Chapter 2

Square linear systems

Not entirely stable? I'm glad you're here to tell us these things. Chewie, take the professor in the back and plug him into the hyperdrive.
—Han Solo, *The Empire Strikes Back*

One of the most frequently encountered tasks in scientific computation is the solution of the linear system of equations $A\mathbf{x} = \mathbf{b}$ for a given square matrix A and vector $\mathbf{b}$. This problem can be solved in a finite number of steps, using an algorithm essentially equivalent to Gaussian elimination. Describing the algorithm is mostly an exercise in organizing some linear algebra.

Because the algorithm takes a finite number of steps, there are no iterations and no truncation error in this chapter, so there is an emphasis on the effect of roundoff error, as described by conditioning. Once we have specified means for measuring the sizes or norms of perturbations to vectors and matrices, we find that the conditioning of the square linear system problem is quite easy to describe. In practice, the values of A and $\mathbf{b}$ are often contaminated by sources of error other than roundoff, such as measurement or modeling error, and our analysis applies equally to those sources.

2.1 Background in linear algebra

If you have taken a course in linear algebra, the material of this section is mostly or entirely a review. Read it carefully anyway, since otherwise our notation may be unfamiliar in a few places.

Terminology

An $m \times n$ matrix A is a rectangular array of entries (also called elements or components). The numbers m and n are called the **row dimension** and the **column dimension**, respectively; collectively they describe the **size** of A. We say A belongs to $\mathbf{R}^{m\times n}$ if its entries are real or $\mathbf{C}^{m\times n}$ if they are complex-valued. A **square** matrix has equal row and column dimensions. A **row vector** has dimension $1 \times n$, while a **column vector** has dimension $m \times 1$. By default, a vector is understood to be a column vector, and we use $\mathbf{R}^n$ or $\mathbf{C}^n$ to represent spaces of vectors. An ordinary number in $\mathbf{R}$ or $\mathbf{C}$ may be called a **scalar**.

We shall use capital letters to refer to matrices and lowercase bold letters for vectors. In addition, the bold symbol $\mathbf{0}$ refers to a vector of all zeros. A **zero matrix** is a matrix of all zeros; we use 0 to mean either a zero matrix or the scalar zero depending on context. To refer to a specific element of a matrix, we use a lowercase letter with subscripts, as in a_{24}. The first subscript is always a row index. To refer to an element of a vector, we use just one subscript, as in x_3.

We shall have frequent need to refer to the individual columns of a matrix. By convention, we use a lowercase bold version of the matrix name, with an index to represent the column number. Thus, $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$ are the columns of the $m \times n$ matrix A. (Note the distinction in typeface between the vector $\mathbf{a}_j$ and the scalar element a_j of a vector named simply $\mathbf{a}$.) Conversely, whenever we define a sequence of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_p$, we can implicitly consider them to be columns of a matrix V.

The **diagonal** (main diagonal) of an $n \times n$ matrix A refers to the entries a_{ii}, $i = 1, \ldots, n$. The entries a_{ij} where $j - i = k$ are on a **superdiagonal** if $k > 0$ and a **subdiagonal** if $k < 0$. The diagonals are numbered as suggested here:

$$\begin{bmatrix} 0 & 1 & 2 & \cdots & n-1 \\ -1 & 0 & 1 & \cdots & n-2 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ -n+2 & \cdots & -1 & 0 & 1 \\ -n+1 & \cdots & -2 & -1 & 0 \end{bmatrix}.$$

A **diagonal** matrix is one whose entries are all zero off the main diagonal. An **upper triangular** matrix U has entries u_{ij} with $u_{ij} = 0$ if $i > j$, and a **lower triangular** matrix L has $\ell_{ij} = 0$ if $i < j$.

The **transpose** of $A \in \mathbf{C}^{m \times n}$ is the matrix $A^T \in \mathbf{C}^{n \times m}$ given by

$$A^T = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ \vdots & \vdots & & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix}.$$

Throughout the rest of these notes, vectors will be implicitly assumed to be column vectors, and the transpose will be used if a row vector is needed. The **conjugate transpose** is given by $A^* = \overline{A^T}$, where the bar denotes taking a complex conjugate.[1] If A is real, then $A^* = A^T$. A square matrix is **symmetric** if $A^T = A$ and **hermitian** if $A^* = A$.

Algebra

Matrices of the same size may be added componentwise. Multiplication by a scalar is also defined componentwise. These operations obey the familiar laws of commutativity, associativity, and distributivity. The multiplication of two matrices, on the other hand, is much less straightforward.

[1]The conjugate of a complex number is found by replacing all references to the imaginary unit i by $-i$. We will not see much of complex numbers until Chapter 4.

In order for matrices A and B to be multiplied, it is necessary that their "inner" dimensions match—i. e., A is $m \times p$ and B is $p \times n$. Note that even if AB is defined, BA may not be, unless $m = n$. In terms of scalar components, the (i, j) entry of $C = AB$ is given by

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}. \tag{2.1}$$

As one consequence, $AB = BA$ is *not* true in general, even when both products are defined. One can prove, however, that when AB is defined, $(AB)^T = B^T A^T$.

It is worth reinterpreting (2.1) at a vector level. If A has dimensions $m \times n$, it can be multiplied on the right by an $n \times 1$ column vector $\mathbf{v}$ to produce an $m \times 1$ column vector $A\mathbf{v}$, which satisfies

$$A\mathbf{v} = \begin{bmatrix} \sum_k a_{1k} v_k \\ \sum_k a_{2k} v_k \\ \vdots \\ \sum_k a_{mk} v_k \end{bmatrix} = v_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix} + v_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix} + \cdots + v_n \begin{bmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{bmatrix} = v_1 \mathbf{a}_1 + \cdots + v_n \mathbf{a}_n. \tag{2.2}$$

In words, we say that $A\mathbf{v}$ is a **linear combination** of the columns of A. Equation (2.2) is very important: *Multiplying a matrix on the right by a column vector produces a linear combination of the columns of the matrix*. There is a similar interpretation of multiplying A on the left by a row vector. Keeping to our convention that boldface letters represent *column* vectors, we write, for $\mathbf{v} \in \mathbf{R}^m$,

$$\begin{aligned} \mathbf{v}^T A &= \begin{bmatrix} \sum_k v_k a_{k1} & \sum_k v_k a_{k2} & \cdots & \sum_k v_k a_{kn} \end{bmatrix} \\ &= v_1 \begin{bmatrix} a_{11} & \cdots & a_{1n} \end{bmatrix} + v_2 \begin{bmatrix} a_{21} & \cdots & a_{2n} \end{bmatrix} + \cdots + v_m \begin{bmatrix} a_{m1} & \cdots & a_{mn} \end{bmatrix}. \end{aligned} \tag{2.3}$$

Thus *multiplying a matrix on the left by a row vector produces a linear combination of the rows of the matrix.*

These two observations extend to more general matrix-matrix multiplications. One can show that (here again A is $m \times p$ and B is $p \times n$)

$$AB = A \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_n \end{bmatrix} = \begin{bmatrix} A\mathbf{b}_1 & A\mathbf{b}_2 & \cdots & A\mathbf{b}_n \end{bmatrix} \tag{2.4}$$

In words, a matrix-matrix product is a horizontal concatenation of matrix-vector products involving the columns of the right-hand matrix. Equivalently, if we write A in terms of rows, then

$$A = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \vdots \\ \mathbf{w}_m^T \end{bmatrix} \qquad \text{implies} \qquad AB = \begin{bmatrix} \mathbf{w}_1^T B \\ \mathbf{w}_2^T B \\ \vdots \\ \mathbf{w}_m^T B \end{bmatrix}. \tag{2.5}$$

Thus, a matrix-matrix product is also a vertical concatenation of vector-matrix products involving the rows of the left-hand matrix. All of our representations of matrix multiplication are equivalent, so whichever one is most convenient at any moment can be used.

Example 2.1. Let

$$A = \begin{bmatrix} 1 & -1 \\ 0 & 2 \\ -3 & 1 \end{bmatrix}, \qquad B = \begin{bmatrix} 2 & -1 & 0 & 4 \\ 1 & 1 & 3 & 2 \end{bmatrix}.$$

Then, going by (2.1), we get

$$\begin{aligned} AB &= \begin{bmatrix} (1)(2)+(-1)(1) & (1)(-1)+(-1)(1) & (1)(0)+(-1)(3) & (1)(4)+(-1)(2) \\ (0)(2)+(2)(1) & (0)(-1)+(2)(1) & (0)(0)+(2)(3) & (0)(4)+(2)(2) \\ (-3)(2)+(1)(1) & (-3)(-1)+(1)(1) & (-3)(0)+(1)(3) & (-3)(4)+(1)(2) \end{bmatrix} \\ &= \begin{bmatrix} 1 & -2 & -3 & 2 \\ 2 & 2 & 6 & 4 \\ -5 & 4 & 3 & -10 \end{bmatrix}. \end{aligned}$$

But note also, for instance, that

$$A \begin{bmatrix} 2 \\ 1 \end{bmatrix} = 2 \begin{bmatrix} 1 \\ 0 \\ -3 \end{bmatrix} + 1 \begin{bmatrix} -1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ -5 \end{bmatrix},$$

and so on, as according to (2.4).

The **identity matrix of size** n, called I (or sometimes I_n), is a diagonal $n \times n$ matrix with every diagonal entry equal to one. As can be seen from (2.4) and (2.5), it satisfies $AI = A$ for $A \in \mathbf{C}^{m \times n}$ and $IB = B$ for $B \in \mathbf{C}^{n \times p}$. It is therefore the matrix analog of the number 1.

Example 2.2. Let

$$B = \begin{bmatrix} 2 & 1 & 7 & 4 \\ 6 & 0 & -1 & 0 \\ -4 & -4 & 0 & 1 \end{bmatrix}.$$

Suppose we want to create a zero in the (2,1) entry by adding -3 times the first row to the second row, leaving the other rows unchanged. We can express this operation as a product AB as follows. From dimensional considerations alone, A will need to be 3×3. According to (2.3), we get "-3 times row one plus row two" from left-multiplying B by the vector $\begin{bmatrix} -3 & 1 & 0 \end{bmatrix}$. Equation (2.5) tells us that this must be the second row of A. Since the first and third rows of AB are the same as those of B, similar logic tells us that the first and third rows of A are the same as the identity matrix:

$$\begin{bmatrix} 1 & 0 & 0 \\ -3 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} B = \begin{bmatrix} 2 & 1 & 7 & 4 \\ 0 & -3 & -22 & -12 \\ -4 & -4 & 0 & 1 \end{bmatrix}.$$

This can be verified directly using (2.1).

Note that a square matrix A can always be multiplied by itself to get a matrix of the same size. Hence we can define the integer powers $A^2 = A \cdot A$, $A^3 = A^2 \cdot A = A \cdot A^2$ (by associativity), and so on. By definition, $A^0 = I$.

According to the rules for multiplying matrices, there are two ways for vectors to be multiplied together. If $\mathbf{v}$ and $\mathbf{w}$ are in $\mathbf{C}^n$, their **inner product** is

$$\mathbf{v}^*\mathbf{w} = \sum_{k=1}^{n} \overline{v_k}\, w_k.$$

Trivially, one finds that $\mathbf{w}^*\mathbf{v} = \overline{(\mathbf{v}^*\mathbf{w})}$. Additionally, *any* two vectors $\mathbf{v} \in \mathbf{C}^m$ and $\mathbf{w} \in \mathbf{C}^n$ have an **outer product**, which is an $m \times n$ matrix:

$$\mathbf{v}\mathbf{w}^* = \begin{bmatrix} v_1\,\overline{w_1} & v_1\,\overline{w_2} & \cdots & v_1\,\overline{w_n} \\ v_2\,\overline{w_1} & v_2\,\overline{w_2} & \cdots & v_2\,\overline{w_n} \\ \vdots & \vdots & & \vdots \\ v_m\,\overline{w_1} & v_m\,\overline{w_2} & \cdots & v_m\,\overline{w_n} \end{bmatrix}.$$

Problems

2.1.1. In racquetball, the winner of a rally serves the next rally. Generally, the server has an advantage. Suppose that when Ashley and Barbara are playing racquetball, Ashley wins 60% of the rallies she serves and Barbara wins 70% of the rallies she serves. If $\mathbf{x} \in \mathbf{R}^2$ is such that x_1 is the probability that Ashley serves first and $x_2 = 1 - x_1$ is the probability that Barbara serves first, define a matrix A such that $A\mathbf{x}$ gives the probabilities that Ashley and Barbara serve the second rally. What is the meaning of $A^{10}\mathbf{x}$?

2.1.2. Suppose we have lists of n terms and m documents. We can define an $m \times n$ matrix A such that $a_{ij} = 1$ if term j appears in document i, and $a_{ij} = 0$ otherwise. Now suppose that the term list is

```
'numerical', 'analysis', 'more', 'cool', 'accounting'
```

and that $\mathbf{x} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \end{bmatrix}^T$. Give an interpretation of the product $A\mathbf{x}$.

2.1.3. Let

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

Show that $A^n = 0$ when $n \geq 4$.

2.1.4. Suppose B is an arbitrary 4×3 matrix. In each part below a matrix A is described in terms of B. Express A as a product of B with one or more other matrices.

(a) $A \in \mathbf{R}^{4\times 1}$ is the result of adding the first column of B to -2 times the last column of B.

(b) The rows of $A \in \mathbf{R}^{4\times 3}$ are the rows of B in reverse order.

(c) The first column of $A \in \mathbf{R}^{4\times 3}$ is 1 times the first column of B, the second column of A is 2 times the second column of B, and the third column of A is 3 times the third column of B.

(d) $A \in \mathbf{R}$ is the sum of all elements of B.

2.1.5. Find two matrices A and B, neither of which is the zero matrix, such that $AB = 0$.

2.1.6. Prove that when AB is defined, $B^T A^T$ is defined too, and use equation (2.1) to show that $(AB)^T = B^T A^T$.

2.1.7. Prove true, or give a counterexample: The product of upper triangular square matrices is upper triangular.

2.1.8. Prove true, or give a counterexample: The product of symmetric matrices is symmetric.

2.1.9. (a) Prove that for real vectors $\mathbf{v}$ and $\mathbf{w}$ of the same length, the inner products $\mathbf{v}^T\mathbf{w}$ and $\mathbf{w}^T\mathbf{v}$ are equal.

(b) Prove true, or give a counterexample for, the equivalent statement about outer products, $\mathbf{v}\mathbf{w}^T$ and $\mathbf{w}\mathbf{v}^T$.

2.2 Triangular linear systems

With the definitions of section 2.1, we are now able to define the central problem of this chapter: Given $A \in \mathbf{R}^{n\times n}$ and $\mathbf{b} \in \mathbf{R}^n$, find $\mathbf{x} \in \mathbf{R}^n$ such that $A\mathbf{x} = \mathbf{b}$. Writing out these equations scalarwise, we obtain

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n. \end{aligned}$$

Another way of stating the problem is as finding a linear combination of the columns of A that equals $\mathbf{b}$.

Mathematically, we say that A is **nonsingular** or **invertible** if there exists another $n \times n$ matrix A^{-1}, the **inverse** of A, such that $AA^{-1} = A^{-1}A = I$, the identity matrix. Otherwise, A is **singular**. If a matrix is invertible, its inverse is unique. This and the following facts are proved in any elementary text on linear algebra.

Theorem 2.1. *The following statements are equivalent:*

1. *A is nonsingular.*
2. *$(A^{-1})^{-1} = A$.*
3. *$A\mathbf{x} = \mathbf{0}$ implies that $\mathbf{x} = \mathbf{0}$.*
4. *$A\mathbf{x} = \mathbf{b}$ has a unique solution, $\mathbf{x} = A^{-1}\mathbf{b}$, for any $\mathbf{b} \in \mathbf{R}^n$.*

Example 2.3. If we define

$$S = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix},$$

then it is easy to check that for *any* real value of α we have

$$S \begin{bmatrix} \alpha \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Hence the linear system $S\mathbf{x} = \mathbf{b}$ with $\mathbf{b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ has infinitely many solutions. For other choices of $\mathbf{b}$ the system can be proven to have no solutions.

While matrix inverses are very valuable mathematically, they almost never appear in computational algorithms, because there are more efficient alternatives. The solution process is especially easy to demonstrate for systems with triangular matrices. For example, consider the lower triangular system

$$\begin{bmatrix} 4 & 0 & 0 & 0 \\ 3 & -1 & 0 & 0 \\ -1 & 0 & 3 & 0 \\ 1 & -1 & -1 & 2 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 8 \\ 5 \\ 0 \\ 1 \end{bmatrix}.$$

The first row of this system demands only that $4x_1 = 8$, which is easily solved as $x_1 = 8/4 = 2$. Now, the second row states that $3x_1 - x_2 = 5$. As x_1 is already known, it can be replaced to find that $x_2 = -(5 - 3 \cdot 2) = 1$. Similarly, the third row gives $x_3 = (0 + 1 \cdot 2)/3 = 2/3$, and the last row yields $x_4 = (1 - 1 \cdot 2 + 1 \cdot 1 + 1 \cdot 2/3)/2 = 1/3$. Hence the solution is

$$\mathbf{x} = \begin{bmatrix} 2 \\ 1 \\ 2/3 \\ 1/3 \end{bmatrix}.$$

The process just described is called **forward substitution**. In the 4×4 lower triangular case of $L\mathbf{x} = \mathbf{b}$ it leads to the formulas

$$\begin{aligned} x_1 &= \frac{b_1}{\ell_{11}} \\ x_2 &= \frac{b_2 - \ell_{21}x_1}{\ell_{22}} \\ x_3 &= \frac{b_3 - \ell_{31}x_1 - \ell_{32}x_2}{\ell_{33}} \\ x_4 &= \frac{b_4 - \ell_{41}x_1 - \ell_{42}x_2 - \ell_{43}x_3}{\ell_{44}}. \end{aligned}$$

Note that in the last formula, for example, we can express $\ell_{41}x_1 + \ell_{42}x_2 + \ell_{43}x_3$ as the inner product between $\begin{bmatrix} \ell_{41} & \ell_{42} & \ell_{43} \end{bmatrix}$ and the "partial vector" $\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T$.

Our MATLAB implementation for forward substitution in the $n \times n$ case is shown in Function 2.1. Line 10 may look unnecessary, but without it the resulting `x` will be a row vector rather than a column vector as we desire. The reference in line 12 to `L(i,1:1-i)` is a row vector consisting of the first $i - 1$ columns of the ith row of L. (When $i = 1$, the reference is understood to be empty since `1:i-1` is empty.) One way to understand how this function works is to use the MATLAB debugging tools to step through it, line by line, for the example above. During the debugging you can check the values of all variables and evaluate expressions like `L(i,1:1-i)` to be sure they

Function 2.1 Solution of a lower triangular linear system.

```
function x = forwardsub(L,b)
% FORWARDSUB   Forward substitution for lower-triangular linear systems.
% Input:
%   L    lower triangular square matrix (n by n)
%   b    right-hand side vector (n by 1)
% Output:
%   x    solution of Lx=b (n by 1 vector)

n = length(L);
x = zeros(n,1);
for i = 1:n
  x(i) = ( b(i) - L(i,1:i-1)*x(1:i-1) ) / L(i,i);
end
```

give what you expect. Or, you can make yourself the computer and verify the function on paper by hand.

For upper triangular systems $U\mathbf{x} = \mathbf{b}$ an analogous process of **backward substitution** begins by solving for the last component $x_n = b_n/u_{nn}$ and working backward. For the 4×4 case we have

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix} \mathbf{x} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}.$$

Solving the system backward, starting with x_4 first and then proceeding in descending order, gives

$$\begin{aligned} x_4 &= \frac{b_4}{u_{44}} \\ x_3 &= \frac{b_3 - u_{34}x_4}{u_{33}} \\ x_2 &= \frac{b_2 - u_{23}x_3 - u_{24}x_4}{u_{22}} \\ x_1 &= \frac{b_1 - u_{12}x_2 - u_{13}x_3 - u_{14}x_4}{u_{11}}. \end{aligned}$$

It should be clear that forward or backward substitution fails if, and only if, one of the diagonal entries of the system matrix is zero. This condition is equivalent to the matrix of the system being singular.

Problems

2.2.1. (a) Suppose that $A \in \mathbf{R}^{n\times n}$ and $B \in \mathbf{R}^{n\times p}$, where the columns of B are $\mathbf{b}_1, \dots, \mathbf{b}_p$. Show that if $X \in \mathbf{R}^{n\times p}$ with columns $\mathbf{x}_1, \dots, \mathbf{x}_p$ exists such that $AX = B$, then $A\mathbf{x}_j = \mathbf{b}_j$, for $j = 1, \dots, p$.

(b) Describe how to find A^{-1} by solving n linear systems with the matrix A. (Hint: Consider the equation $AA^{-1} = I$.)

2.2.2. Prove that if A and B are invertible, then $(AB)^{-1} = B^{-1}A^{-1}$ (and thus AB is invertible as well).

2.2.3. Show that if A is invertible, then $(A^T)^{-1} = (A^{-1})^T$.

2.2.4. Find a right-hand side vector $\mathbf{b}$ such that the system $S\mathbf{x} = \mathbf{b}$, where S is defined as in Example 2.3, has no solution.

2.2.5. Solve the following triangular systems by hand.

(a)
$$\begin{aligned} 3x_1 + 2x_2 + x_3 &= 1 \\ x_2 - x_3 &= 2 \\ -2x_3 &= -4 \end{aligned}$$

(b)
$$\begin{bmatrix} 4 & 0 & 0 & 0 \\ 1 & -2 & 0 & 0 \\ -1 & 4 & 4 & 0 \\ 2 & -5 & 5 & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} -4 \\ 1 \\ -3 \\ 5 \end{bmatrix}$$

2.2.6. Use the `forwardsub` function to solve the systems from the previous problem. Verify that the solution is correct by computing $L\mathbf{x}$ and comparing with $\mathbf{b}$.

2.2.7. Implement a function `backsub` for backward substitution to solve $U\mathbf{x} = \mathbf{b}$. Write the function so that it is used in a manner similar to `forwardsub`; usage should be `x = backsub(U,b)`. Test it by creating your own 4×4 U and exact solution $\mathbf{x}$ to define $\mathbf{b}$, then applying your function and checking that $\mathbf{x}$ is returned.

2.2.8. Use the `backsub` function from the previous problem to solve the following systems. Verify that the solution is correct by computing $U\mathbf{x}$ and comparing with $\mathbf{b}$.

(a)
$$\begin{bmatrix} 3 & 1 & 0 \\ 0 & -1 & -2 \\ 0 & 0 & 3 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 1 \\ 1 \\ 6 \end{bmatrix}$$

(b)
$$\begin{bmatrix} 3 & 1 & 0 & 6 \\ 0 & -1 & -2 & 7 \\ 0 & 0 & 3 & 4 \\ 0 & 0 & 0 & 5 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 4 \\ 1 \\ 1 \\ 5 \end{bmatrix}$$

2.3 Gaussian elimination and LU factorization

Every first linear algebra course introduces the technique of **Gaussian elimination** for a general square system $A\mathbf{x} = \mathbf{b}$. In Gaussian elimination one uses row operations on an augmented matrix $\widetilde{A} = [A\ \mathbf{b}]$ to reduce it to an equivalent triangular system (usually upper triangular). Rather than writing out the process in full generality, we use an example to refresh your memory.

Example 2.4. Suppose

$$A = \begin{bmatrix} 2 & 0 & 4 & 3 \\ -4 & 5 & -7 & -10 \\ 1 & 15 & 2 & -4.5 \\ -2 & 0 & 2 & -13 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 4 \\ 9 \\ 29 \\ 40 \end{bmatrix}.$$

We augment A with $\mathbf{b}$ to get

$$\widetilde{A}_1 = \begin{bmatrix} 2 & 0 & 4 & 3 & 4 \\ -4 & 5 & -7 & -10 & 9 \\ 1 & 15 & 2 & -4.5 & 29 \\ -2 & 0 & 2 & -13 & 40 \end{bmatrix}.$$

We add multiples of the first row to rows 2–4 in order to put some zeros in the first column, leading to

$$\widetilde{A}_2 = \begin{bmatrix} 2 & 0 & 4 & 3 & 4 \\ 0 & 5 & 1 & -4 & 17 \\ 0 & 15 & 0 & -6 & 27 \\ 0 & 0 & 6 & -10 & 44 \end{bmatrix}.$$

Continuing to put zeros in columns two and three in a triangular fashion, we obtain

$$\widetilde{A}_3 = \begin{bmatrix} 2 & 0 & 4 & 3 & 4 \\ 0 & 5 & 1 & -4 & 17 \\ 0 & 0 & -3 & 6 & -24 \\ 0 & 0 & 6 & -10 & 44 \end{bmatrix}, \text{ and then } \widetilde{A}_4 = \begin{bmatrix} 2 & 0 & 4 & 3 & 4 \\ 0 & 5 & 1 & -4 & 17 \\ 0 & 0 & -3 & 6 & -24 \\ 0 & 0 & 0 & 2 & -4 \end{bmatrix}.$$

We now have the triangular system

$$\begin{bmatrix} 2 & 0 & 4 & 3 \\ 0 & 5 & 1 & -4 \\ 0 & 0 & -3 & 6 \\ 0 & 0 & 0 & 2 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 4 \\ 17 \\ -24 \\ -4 \end{bmatrix}, \tag{2.6}$$

which is easily solved by backward substitution:

$$\begin{aligned} x_4 &= \frac{-4}{2} = -2 \\ x_3 &= \frac{-24 - 6(-2)}{-3} = 4 \\ x_2 &= \frac{17 + 4(-2) - 1(4)}{5} = 1 \\ x_1 &= \frac{4 - 3(-2) - 4(4) - 0(1)}{2} = -3 \end{aligned}$$

.

The first step in passing from $\widetilde{A}_1$ to $\widetilde{A}_2$ in this example is to add twice the first row to the second one. This is equivalent to multiplying $\widetilde{A}_1$ on the left by

$$L_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.7}$$

This matrix can be derived by applying the desired row operation to the identity matrix.[2] We want to do some algebra with these matrices and so need a way to express them compactly.

In a slight (but very common) deviation from our notational conventions, we write the columns of the $n \times n$ identity matrix I as $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$. These vectors are sometimes called the **standard basis** for $\mathbf{R}^n$. Note that $\mathbf{e}_j$ has a 1 in the jth component and zeros elsewhere. We observe therefore

[2]Such a matrix is called an **elementary matrix** in many textbooks.

that the inner product $\mathbf{e}_i^T\mathbf{e}_j$ is 1 if $i = j$ and zero otherwise, and that the outer product $\mathbf{e}_i\mathbf{e}_j^T$ is an $n \times n$ matrix with a 1 in the (i, j) entry and zeros elsewhere. For example, in $\mathbf{R}^4$ we have

$$\mathbf{e}_2\mathbf{e}_1^T = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix} = \left[\,1\begin{bmatrix}0\\1\\0\\0\end{bmatrix} \quad 0\begin{bmatrix}0\\1\\0\\0\end{bmatrix} \quad 0\begin{bmatrix}0\\1\\0\\0\end{bmatrix} \quad 0\begin{bmatrix}0\\1\\0\\0\end{bmatrix}\,\right] = \begin{bmatrix}0 & 0 & 0 & 0\\1 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\end{bmatrix}.$$

Thus, the matrix in (2.7) can be succintly written as $L_1 = I + 2\mathbf{e}_2\mathbf{e}_1^T$. This expresses "add two times row 1 to row 2" independently of the dimension n. The overall reduction process in Example 2.4 can be expressed as

$$\begin{aligned}\widetilde{A}_4 &= L_5L_4L_3L_2L_1\widetilde{A}_1\\ &= (I + 2\mathbf{e}_4\mathbf{e}_3^T)(I - 3\mathbf{e}_3\mathbf{e}_2^T)(I + \mathbf{e}_4\mathbf{e}_1^T)(I - \tfrac{1}{2}\mathbf{e}_3\mathbf{e}_1^T)(I + 2\mathbf{e}_2\mathbf{e}_1^T)\widetilde{A}_1.\end{aligned}$$

From now on we consider only the first four columns of the augmented matrices—that is, we exclude the right-hand side $\mathbf{b}$ from the discussion. This streamlines the result to $U = L_5L_4L_3L_2L_1A$, where U is upper triangular. Hence

$$A = (L_5L_4L_3L_2L_1)^{-1}U = L_1^{-1}L_2^{-1}L_3^{-1}L_4^{-1}L_5^{-1}U$$

(see problem 2 in section 2.2). Now we observe that

$$(I - 2\mathbf{e}_2\mathbf{e}_1^T)(I + 2\mathbf{e}_2\mathbf{e}_1^T) = I - 2\mathbf{e}_2\mathbf{e}_1^T I + 2I\mathbf{e}_2\mathbf{e}_1^T - 4\mathbf{e}_2\mathbf{e}_1^T\mathbf{e}_2\mathbf{e}_1^T = I,$$

since $\mathbf{e}_1^T\mathbf{e}_2 = 0$. This shows that $L_1^{-1} = I - 2\mathbf{e}_2\mathbf{e}_1^T$. Each of the other matrices is inverted as easily, so that

$$A = (I - 2\mathbf{e}_2\mathbf{e}_1^T)(I + \tfrac{1}{2}\mathbf{e}_3\mathbf{e}_1^T)(I - \mathbf{e}_4\mathbf{e}_1^T)(I + 3\mathbf{e}_3\mathbf{e}_2^T)(I - 2\mathbf{e}_4\mathbf{e}_3^T)U.$$

Finally, when these matrix products are expanded, we find (just as in the inverse computation above) that the "cross terms" all become zero due to inner products $\mathbf{e}_i^T\mathbf{e}_j$ with $i \neq j$. So

$$A = (I - 2\mathbf{e}_2\mathbf{e}_1^T + \tfrac{1}{2}\mathbf{e}_3\mathbf{e}_1^T - \mathbf{e}_4\mathbf{e}_1^T + 3\mathbf{e}_3\mathbf{e}_2^T - 2\mathbf{e}_4\mathbf{e}_3^T)U = LU,$$

where

$$L = \begin{bmatrix}1 & 0 & 0 & 0\\-2 & 1 & 0 & 0\\\frac{1}{2} & 3 & 1 & 0\\-1 & 0 & -2 & 1\end{bmatrix}$$

is a special type of lower triangular matrix called **unit lower triangular**, because of the ones on the diagonal.

All of the algebra of the preceeding paragraphs illustrates an extremely important and relatively simple principle: Gaussian elimination on $A\mathbf{x} = \mathbf{b}$ produces an ***LU* factorization** $A = LU$, where L is unit lower triangular and U is upper triangular. Moreover, the entries in the lower triangle of L are the negatives of the row elimination multipliers. To solve $A\mathbf{x} = \mathbf{b}$ or $LU\mathbf{x} = \mathbf{b}$, we substitute $\mathbf{z} = U\mathbf{x}$ and get $L\mathbf{z} = \mathbf{b}$. Hence, LU factorization reduces any linear system to two triangular ones. In summary, an algorithm for $A\mathbf{x} = \mathbf{b}$ is:

Function 2.2 *LU* factorization for a square matrix.

```
function [L,U] = lufact(A)
% LUFACT   LU factorization (demo only--not stable!)
% Input:
%   A    square matrix
% Output:
%   L,U  unit lower triangular and upper triangular such that LU=A

n = length(A);
L = eye(n);   % ones on diagonal

% Gaussian elimination
for j = 1:n-1
  for i = j+1:n
    L(i,j) = A(i,j) / A(j,j);   % row multiplier
    A(i,j:n) = A(i,j:n) - L(i,j)*A(j,j:n);
  end
end

U = triu(A);
```

1. Factor $LU = A$ using Gaussian elimination.
2. Solve $L\mathbf{z} = \mathbf{b}$ for $\mathbf{z}$ using forward substitution.
3. Solve $U\mathbf{x} = \mathbf{z}$ for $\mathbf{x}$ using backward substitution.

One of the nice aspects of this algorithm is that the factorization step depends only on the matrix A only; the right-hand side $\mathbf{b}$ is not involved. Thus if one has to solve multiple systems with a single matrix A, the factorization needs to be performed only once for all systems. As we will see in section 2.4, the factorization is the most computationally expensive step, so this note is of more than academic interest.

Implementation

A code for *LU* factorization is given in Function 2.2. The multipliers are stored in the lower triangle of L as they are found. When operations are done to put zeros in column j, they are carried out only in lower rows to create an upper triangular matrix. (Only columns j through n are accessed, since the other entries should already be zero.) At the end the matrix A should be upper triangular, but since roundoff errors could create some small nonzeros the `triu` command is used to make them exactly zero.

Example 2.5. This example validates the consistency of our calculations and implementation. You might want to step through `lufact` using the debugger. (The last line assumes that you successfully created `backsub` in problem 2.2.7.)

```
>> A=[2 0 4 3;-4 5 -7 -10; 1 15 2 -4.5; -2 0 2 -13];
>> [L,U] = lufact(A)

L =

    1.0000         0         0         0
   -2.0000    1.0000         0         0
    0.5000    3.0000    1.0000         0
   -1.0000         0   -2.0000    1.0000


U =

     2     0     4     3
     0     5     1    -4
     0     0    -3     6
     0     0     0     2

>> L*U

ans =

    2.0000         0    4.0000    3.0000
   -4.0000    5.0000   -7.0000  -10.0000
    1.0000   15.0000    2.0000   -4.5000
   -2.0000         0    2.0000  -13.0000

>> backsub( U, forwardsub(L,[4;9;29;40]) )

ans =

    -3
     1
     4
    -2
```

Observe from Function 2.2 that the factorization can fail if $a_{jj} = 0$ when it is put in the demoninator in line 14. This does *not* necessarily mean there is a zero in the diagonal of the original A, because A is changed during the computation. Moreover, there are perfectly good nonsingular matrices for which this type of failure occurs, such as

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Fortunately, this defect can be repaired for all nonsingular matrices, as we shall see in section 2.5.

Problems

2.3.1. For each matrix, perform Gaussian elimination by hand to produce an LU factorization. Write out the L matrix using outer products of standard basis vectors.

$$\text{(a)} \begin{bmatrix} 2 & 3 & 4 \\ 4 & 5 & 10 \\ 4 & 8 & 2 \end{bmatrix} \qquad \text{(b)} \begin{bmatrix} 6 & -2 & -4 & 4 \\ 3 & -3 & -6 & 1 \\ -12 & 8 & 21 & -8 \\ -6 & 0 & -10 & 7 \end{bmatrix}$$

2.3.2. Function 2.2 factors $A = LU$ in such a way that L is a *unit lower triangular matrix*—that is, has all ones on the diagonal. It is also possible to define the factorization so that U is a unit upper triangular matrix instead. (In general, both L and U cannot be unit matrices simultaneously.) Show that Function 2.2 can be used *without modification* to produce this version of the factorization. (Hint: Consider the factorization of A^T.) Demonstrate on a 4×4 example in MATLAB.

2.3.3. In MATLAB, define

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 10^{10} \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 0 \\ 0.2 \\ 0.4 \\ 0.6 \\ 0.8 \\ 1 \end{bmatrix}, \quad \mathbf{b} = A\mathbf{y}.$$

(a) Using Function 2.2 and triangular substitutions, solve the linear system $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$. Compute the difference between $\mathbf{x}$ and the exact solution $\mathbf{y}$. Has $\mathbf{x}$ been computed to full machine precision (about 16 decimal digits)?

(b) Repeat part (a) using 10^8 and 10^{12} in place of 10^{10}. (Don't forget to recompute $\mathbf{b}$ after A is changed.) Based on all of your results, hypothesize about the accuracy obtained by the algorithm as a function of the large corner element of A.

(c) Repeat part (a) with 10^{20} as the corner element, and comment on the result in light of your explanation for part (b).

2.4 Operation counts

An important consideration when comparing algorithms is the time they take to execute. This can be a complex and deep issue, dependent on computer architectures, operating systems, and programming—to name just a few confounding details. It is traditional, especially in computational linear algebra, to sacrifice considerable realism for simplicity by comparing algorithms based on a very specific criterion: the number of **flops** (short for "floating point operations") required. In our interpretation each addition, subtraction, multiplication, division, and square root counts as one flop.

The most reliable information found from this type of analysis is the dependence of the flop count on the problem size n (here, the row/column size of the matrix A). In particular, the dependence is usually expressed in terms of the dominant term when n is large. For positive functions $f(n)$ and $g(n)$, we say $f(n) = O\big(g(n)\big)$ ("f is **big-O** of g") as $n \to \infty$ if $f(n)/g(n)$ is bounded above for all sufficiently large n. We say $f(n) \sim g(n)$ ("f is **asymptotic** to g") as $n \to \infty$ if $f(x)/g(x) \to 1$

as $n \to \infty$. Clearly, $f \sim g$ implies $f = O(g)$; asymptotic notation is more specific than big-O notation.[3]

Example 2.6. Consider the functions $f(n) = a_1 n^3 + b_1 n^2 + c_1 n$ and $g(n) = a_2 n^3$ in the limit $n \to \infty$. Then

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{a_1 + b_1 n^{-1} + c_1 n^{-2}}{a_2} = \frac{a_1}{a_2}.$$

Since a_1/a_2 is a constant, $f(n) = O(g(n))$; if $a_1 = a_2$, then $f \sim g$.

Example 2.7. Consider $f(n) = \sin(1/n)$, $g(n) = 1/n$ and $h(n) = 1/n^2$. For large n, Taylor's theorem with remainder implies that $f(n) = 1/n - \cos(1/\xi)/(6n^3)$, where $n < \xi < \infty$. But $\lim_{n\to\infty}(f/g) = \lim_{n\to\infty} 1 - \cos(1/\xi)/(6n^2) = 1$, and so $f \sim g$. On the other hand, comparing f and h, we find $\lim_{n\to\infty}(f/h) = \lim_{n\to\infty} n - \cos(1/\xi)/(6n)$ which diverges, and we cannot say that $f = O(h)$. Considering the same limit for h/f will show that $h = O(f)$, however.

We can count flops for *LU* factorization from the listing of Function 2.2 in section 2.3. Line 14 requires one division, while line 15 requires $n - j + 1$ multiplications and the same number of subtractions. Summing over the two loops, the exact flop count is

$$\sum_{j=1}^{n-1} \sum_{i=j+1}^{n} \left[2(n-j)+3\right] = \sum_{j=1}^{n-1} (n-j)\left[2(n-j)+3\right]. \tag{2.8}$$

If we transform the index using $k = n - j$, this becomes

$$\sum_{k=1}^{n-1} k(2k+3). \tag{2.9}$$

It is possible to find an exact formula for this sum, but we will make use of a simpler result. It is not hard to show using calculus that

$$\begin{aligned}
\sum_{k=1}^{n} k &\sim \frac{n^2}{2} \text{ as } n \to \infty,\\
\sum_{k=1}^{n} k^2 &\sim \frac{n^3}{3} \text{ as } n \to \infty,\\
&\vdots\\
\sum_{k=1}^{n} k^p &\sim \frac{n^{p+1}}{p+1} \text{ as } n \to \infty,
\end{aligned}$$

[3]Really, $O(g)$ and $\sim g$ are *sets* of functions, and $\sim g$ is a subset of $O(g)$. That we write $f = O(g)$ rather than $f \in O(g)$ is a quirk of tradition.

which holds for any positive integer p and has a memorable similarity to the formula for $\int x^p \, dx$.

Now we can simplify (2.9) to

$$2\sum_{k=1}^{n-1} k^2 + 3\sum_{k=1}^{n-1} k \sim \frac{2}{3}(n-1)^3 + \frac{3}{2}(n-1)^2 \sim \frac{2}{3}n^3. \tag{2.10}$$

In conclusion, then, LU factorization takes $\sim \frac{2}{3}n^3$ flops.

Next, we consider forward and backward substitution. From Function 2.1 in section 2.2 we see an inner product in line 12 that takes $(i-1)+(i-2) = 2i-3$ flops within the loop. Summing this and two more operations over the loops, we obtain

$$\sum_{i=1}^{n}(2i-1) \sim n^2$$

flops for forward substitution. The count for backward substitution is exactly the same.

Finally, we consider the complete linear system problem. Recall that if $A = LU$, then $A\mathbf{x} = \mathbf{b}$ is equivalent to the two triangular systems $L\mathbf{z} = \mathbf{b}$ and $U\mathbf{x} = \mathbf{z}$. The LU factorization takes $\sim \frac{2}{3}n^3$ flops, while the two triangular solves take $\sim 2n^2$ flops combined. In the asymptotic sense, the triangular solutions take insignificant time compared to the factorization step. Hence we are justified in saying that solving linear systems by LU factorization takes $\frac{2}{3}n^3$ flops asymptotically.

Problems

2.4.1. The following are asymptotic assertions about the limit $n \to \infty$. In each case, prove the statement true or false.

(a) $n^2 = O(\log n)$ (b) $n^a = O(n^b)$ if $a \le b$

(c) $e^n \sim e^{2n}$ (d) $n + \sqrt{n} \sim n + 2\sqrt{n}$

2.4.2. Show that the inner product of two n-vectors takes $2n-1$ flops.

2.4.3. Show that the multiplication of two $n \times n$ matrices takes $\sim 2n^3$ flops.

2.4.4. The exact sums for terms like those in (2.9) are as follows:

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}, \qquad \sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}.$$

Find the exact result for (2.9) and plot this result on the same log-log plot as the approximate result $2n^3/3$ for `n=logspace(1,4)`. (Note that we are assuming that we can treat n as a continuous variable; assumptions of this sort will be profitable for analyzing numerical methods.) Plot the ratio of the two functions as a function of n on a `semilogx` plot, verifying that the two functions are asymptotic.

2.4.5. Problem 1 of section 2.2 suggests a way to find the inverse of a matrix A by solving n linear systems $A\mathbf{x}_j = \mathbf{e}_j$. Find the asymptotic flop count for this algorithm. (Important note: The special structure of the standard basis vectors make the answer a bit smaller than if the systems being solved were completely general.)

2.4.6. The goal in this problem is to compare the practical performance of *LU* factorization in MATLAB to the theoretical $O(n^3)$ operation count.

Let `n=round( logspace(1,2.5) )`. For each value of n in this vector, generate a random matrix of size n and compute its decomposition using Function 2.2. Time each decomposition using the function `cputime` or the Profiler. Graph the time $\tau(n)$ (vertical) versus n (horizontal) on a log-log scale (using `loglog`). Since the predicted time is $O(n^3)$, add a straight line to your graph representing $\tau(n) = n^3$, and comment on how well the asymptotic flop count reflects the measurements.

2.5 Row pivoting

As mentioned in section 2.3, the $A = LU$ factorization does not work for every nonsingular A.

Example 2.8. Suppose we rearrange Example 2.4 by changing the order in which the equations are given:

$$A = \begin{bmatrix} 2 & 0 & 4 & 3 \\ -2 & 0 & 2 & -13 \\ 1 & 15 & 2 & -4.5 \\ -4 & 5 & -7 & -10 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 4 \\ 40 \\ 29 \\ 9 \end{bmatrix}.$$

This does nothing to change the solution of the system. But now, if we attempt *LU* factorization (ignoring the right-hand side),

$$A_1 = \begin{bmatrix} 2 & 0 & 4 & 3 \\ -2 & 0 & 2 & -13 \\ 1 & 15 & 2 & -4.5 \\ -4 & 5 & -7 & -10 \end{bmatrix}$$
$$A_2 = \begin{bmatrix} 2 & 0 & 4 & 3 \\ 0 & 0 & 6 & -10 \\ 0 & 15 & 0 & -6 \\ 0 & 5 & 1 & -4 \end{bmatrix},$$

at which point we are stuck. There is no way to proceed, because trying to find a row multiplier means division by zero.

MATLAB has a built-in function for finding the *LU* factorization. Let's try it here.

```
>> A=[2 0 4 3;-2 0 2 -13; 1 15 2 -4.5; -4 5 -7 -10]

A =

    2.0000         0    4.0000    3.0000
   -2.0000         0    2.0000  -13.0000
    1.0000   15.0000    2.0000   -4.5000
   -4.0000    5.0000   -7.0000  -10.0000

>> [L,U] = lu(A)

L =
   -0.5000    0.1538    0.0833    1.0000
    0.5000   -0.1538    1.0000         0
   -0.2500    1.0000         0         0
    1.0000         0         0         0


U =

   -4.0000    5.0000   -7.0000  -10.0000
         0   16.2500    0.2500   -7.0000
         0         0    5.5385   -9.0769
         0         0         0   -0.1667
```

The `lu` function returns successfully, but the L it gives is not lower triangular! What's going on here?

The only difference between the systems in Examples 2.4 and 2.8 is that the order of the rows—i. e., the equations of the system—has been switched. From the standpoint of solutions of the system, this ordering is arbitrary, so we are free to change it at any point in the process. MATLAB has represented these row changes by changing the order of the rows in L. Since the rows can still be rearranged to give a lower triangular matrix, one can use forward substitution to solve a system involving L.

How are the rows reordered? When elimination tells us to divide by zero, clearly it's time to make a change. The number in the (k,k) position of A_k, called the **pivot element**, is the key. If it is zero, we need to swap row k with another row in order to create a nonzero pivot. But we don't want to swap with a row above row k, because that would destroy the upper triangular structure we have partially built. (Look again at A_2 in Example 2.8.) So we search only in rows $k+1$ through n of column k to find a nonzero element to serve as pivot. This process is called **row pivoting** or **partial pivoting**.

What if no nonzero pivot can be found? The following theorem ties up this loose end neatly.

Theorem 2.2. *If elimination is to be done in column k, but all elements in rows k through n of that column are zero, then the original matrix A is singular.*

In other words, row pivoting is always successful for a nonsingular matrix. Since with a singular matrix we cannot expect a solution to the system to exist, the theorem cannot be improved.

So far, this notion is quite familiar from the way Gaussian elimination is usually introduced. However, the possibility of inexact solutions—say, due to roundoff error—introduces a major new wrinkle: one may be required to swap rows even if a nonzero pivot is already in place. The following example illustrates why.

Example 2.9. Let

$$A = \begin{bmatrix} -\epsilon & 1 \\ 1 & -1 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1-\epsilon \\ 0 \end{bmatrix},$$

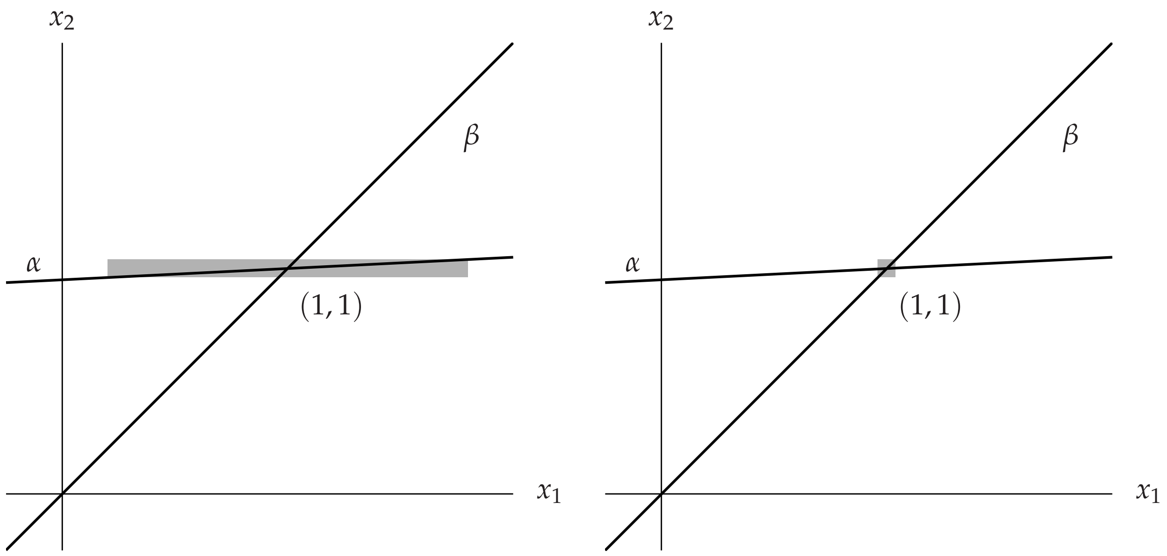


Figure 2.1: Illustration of the need for pivoting. Each row of a 2×2 system in Example 2.9 represents a line, and the exact solution is their intersection. On the left, the shaded box shows that on line α, any uncertainty in x_2 becomes amplified for x_1. Hence α is an unstable choice for eliminating x_1 from the system. On the right, the corresponding box for β is far narrower, so it provides a better pivot.

where ϵ is a small positive number. This can be solved as given, without pivoting.

$$\begin{bmatrix} -\epsilon & 1 & 1-\epsilon \\ 0 & -1+\epsilon^{-1} & \epsilon^{-1}-1 \end{bmatrix} \quad \Rightarrow \quad \begin{aligned} x_2 &= 1 \\ x_1 &= \frac{(1-\epsilon)-1}{-\epsilon}. \end{aligned}$$

In exact arithmetic, this produces the correct solution $x_1 = x_2 = 1$. But look at how x_1 is computed. It involves the subtraction of nearby numbers, which is sure to lead to cancellation in floating point arithmetic. That will make the solution unstable.

On the other hand, suppose we flipped the rows of the matrix before elimination.

$$\begin{bmatrix} 1 & -1 & 0 \\ 0 & 1-\epsilon & 1-\epsilon \end{bmatrix} \quad \Rightarrow \quad \begin{aligned} x_2 &= 1 \\ x_1 &= \frac{0-(-1)}{1}. \end{aligned}$$

There is no reason to think that instability is introduced here.

Figure 2.1 illustrates the difficulty. The solution of the system is the intersection of lines α and β, corresponding to the two rows of the linear system ($x_2 = \epsilon x_1 + (1-\epsilon)$ and $x_2 = x_1$). In the first procedure above we find x_2 first and then read off x_1 from the nearly horizontal α. Because of the small slope, any uncertainty in x_2 becomes much larger in x_1. In the second case we use line β to find x_1 from x_2, and any error in x_2 is not amplified.

We can now say that Gaussian elimination without pivoting is unstable for this problem. This is clear because another algorithm—namely, row pivoting—can produce results that are more accurate to an extent that can be made arbitrarily large as $\epsilon \to 0$.

To summarize what was illustrated by Example 2.9: A small pivot element means weak dependence on the variable that is about to be eliminated, and this makes the elimination inaccurate. We can now view a zero pivot as the extreme case for which elimination is even theoretically impossible. This observation suggests that for elimination in column k, we should swap row k with the row below it that gives the largest (in absolute value) candidate pivot.

We now know that row swapping can always lead to an upper triangular matrix when A is nonsingular. But we must reconsider our derivation of the LU factorization. First, we express the row swap operation as left-multiplication by a **permutation matrix**, defined as a rearrangement of the rows of the identity matrix. For example,

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \end{bmatrix} = \begin{bmatrix} 21 & 22 & 23 & 24 \\ 41 & 42 & 43 & 44 \\ 31 & 32 & 33 & 34 \\ 11 & 12 & 13 & 14 \end{bmatrix}.$$

Schematically, the row operations used to reduce A to upper triangular U are (in the 4×4 case)

$$L_3 P_3 L_2 P_2 L_1 P_1 A = U,$$

where P_k swaps row k with some row greater than or equal to k, and L_k is unit lower triangular and has elimination multipliers in column k only. Because of the row pivoting strategy, these multipliers are less than or equal to one in absolute value. A little more algebra can be used to show that

$$\widetilde{L}_3 \widetilde{L}_2 \widetilde{L}_1 P_3 P_2 P_1 A = U,$$

where $\widetilde{L}_k$ is equal to L_k with the elimination multipliers reordered. Along the same lines as the discussion in section 2.3, this is equivalent to

$$PA = LU, \tag{2.11}$$

where P is a permutation matrix resulting from all of the row swaps in the factorization, and L is unit lower triangular.

Since one can show that any permutation matrix is invertible, with $P^{-1} = P^T$ (see problem 6), equation (2.11) is equivalent to $A = P^T LU$, and we can speak of a $P^T LU$ factorization. Furthermore, the linear system $A\mathbf{x} = \mathbf{b}$ becomes $P^T LU\mathbf{x} = \mathbf{b}$, which is equivalent to $L(U\mathbf{x}) = (P\mathbf{b})$. Hence after the factorization, the entries of $\mathbf{b}$ are permuted according to P, and then one backward and one forward substitution finish the job as before.

Row pivoting does not add any flops to the factorization process. However, the algorithm does need to compare $n - k + 1$ numbers before doing elimination in column k. Since (using $j = n - k + 1$)

$$\sum_{k=1}^{n-1} (n - k + 1) = \sum_{j=2}^{n} j \sim \frac{1}{2} n^2,$$

the total number of comparisons is insignificant next to the $\sim \frac{2}{3} n^3$ flops required. For better efficiency, practical implementations may avoid actual row swaps in memory and instead maintain a list of the new row ordering.

Solving linear systems in MATLAB

The MATLAB function `lu` returns $P^T L$ and U when called with two outputs, so what was seen in Example 2.8 was a row-permuted version of the lower triangular L. However, if the goal of factoring is to solve a linear system $A\mathbf{x} = \mathbf{b}$, normally one does not need to find the factors explicitly. Instead one uses the special backslash operator, \, as in the statement `x=A\b`. From a mathematical point of view, `A\` is a synonym for left-multiplication by A^{-1}.

Example 2.10. In this example we create a small linear system with a known solution, then solve it and compare to the exact answer.

```
>> A = magic(5)
A =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9


>> x_exact = randn(5,1)
x_exact =
   -0.6231
   -0.9737
   -0.2263
   -0.8913
   -0.5329

>> b = A*x_exact;
>> x = A\b
x =
   -0.6231
   -0.9737
   -0.2263
   -0.8913
   -0.5329

>> x - x_exact
ans =

   1.0e-15 *

         0
   -0.2220
    0.1110
   -0.2220
    0.1110
```

The result is as good as could be expected in floating-point arithmetic.

While the effect of the backslash is easy to describe, its methods are not. Using backslash invokes an expert system that examines the coefficient matrix, looking for properties that make the system solution faster. One such property is triangularity: If the backslash is given a triangular (or permuted triangular) matrix, it employs forward or backward substitution to solve the system. If the system fails to be triangular and also does not meet certain other criteria, then the generic method is to find a $P^T LU$ factorization, followed by a permutation of $\mathbf{b}$ and two triangular solves.

Problems

2.5.1. What are the aspect ratios of the shaded boxes in Figure 2.1? Give exact answers in terms of the parameter ϵ from Example 2.9.

2.5.2. Repeat Example 2.10 with the number 4 replacing 5 in the first call to `magic`. What explanation does MATLAB give for the poor accuracy? Also find the **residual** vector $\mathbf{b} - A\mathbf{x}$; the residual is the amount by which the equation is not satisfied. Is the result surprising?

2.5.3. Suppose that `A` is a square matrix and `b` is a column vector of compatible length. On the left is correct MATLAB code to solve $A\mathbf{x} = \mathbf{b}$; on the right is similar but incorrect code. Explain using mathematical notation exactly what vector is found by the right-hand version.

```
[L,U] = lu(A);                [L,U] = lu(A);
x = U \ (L\b);                x = U \ L \ b;
```

2.5.4. Suppose a string is stretched with tension τ horizontally between two anchors at $x = 0$ and $x = 1$. At each of the $n-1$ equally spaced positions $x_k = k/n$, $k = 1, \ldots, n-1$, we attach a little mass m_i and allow the string to come to equilibrium. This causes vertical displacement of the string. Let q_k be the amount of displacement at x_k. If the displacements are not too large, then an approximate force balance equation is

$$n\tau(q_k - q_{k-1}) + n\tau(q_k - q_{k+1}) = m_k g, \qquad k = 1, \ldots, n-1,$$

where $g = -9.8\ \text{m/s}^2$ is the acceleration due to gravity, and we naturally define $q_0 = 0$ and $q_n = 0$ due to the anchors.

(a) Show that the force balance equations can be written as a linear system $A\mathbf{q} = \mathbf{f}$, where $\mathbf{q}$ is a vector of displacements and A is a tridiagonal matrix (i. e., $a_{ij} = 0$ if $|i - j| > 1$) of size $(n-1) \times (n-1)$.

(b) Let $\tau = 10$ N, and $m_k = (1/10n)$ kg for every k. Find the displacements in MATLAB when $n = 4$ and $n = 40$, and superimpose plots of $\mathbf{q}$ over $0 \le x \le 1$ for the two cases. (Be sure to include the zero values at $x = 0$ and $x = 1$ in your plots of the string.)

(c) Repeat (b) for the case $m_k = (k/5n^2)$ kg.

2.5.5. This is a continuation of problem 2.3.3. Recall the definitions

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 10^{10} \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 0 \\ 0.2 \\ 0.4 \\ 0.6 \\ 0.8 \\ 1 \end{bmatrix}, \quad \mathbf{b} = A\mathbf{y}.$$

(a) Use the MATLAB backslash to solve $A\mathbf{x} = \mathbf{b}$, and compute the accuracy of $\mathbf{x}$. Has row pivoting improved matters?

(b) Let D be the 6×6 diagonal matrix the same size as A such that

$$d_{ii} = \left[\max_{1 \le j \le 6} |a_{ij}| \right]^{-1}.$$

Consequently DA has a maximum entry of 1 (absolute value) in each row. Solve the mathematically equivalent linear system $(DA)\mathbf{x} = D\mathbf{b}$ using row pivoting, and again compute the accuracy of $\mathbf{x}$. (This is called **scaled row pivoting**.)

(c) Write a function `x=srpsolve(A,b)` that solves $A\mathbf{x} = \mathbf{b}$ using scaled row pivoting for a general square matrix A. Your function should use backslash for the row-pivoted factorization step. With clever use of the commands `abs`, `max`, and `diag`, you can construct D in one line.

2.5.6. Suppose an $n \times n$ permutation matrix P has the effect of moving rows $1, 2, \dots, n$ to new positions $i_1, i_2, \dots, i_n$.

(a) Explain why $P = \mathbf{e}_{i_1}\mathbf{e}_1^T + \mathbf{e}_{i_2}\mathbf{e}_2^T + \cdots + \mathbf{e}_{i_n}\mathbf{e}_n^T$.

(b) Show that $P^{-1} = P^T$.

2.6 Vector and matrix norms

The manipulations on matrices and vectors so far in this chapter have been algebraic, much like those in a first linear algebra course. In order to progress to the analysis of these algorithms on a computer, we need a way to measure the size of matrix and vector perturbations. This is a matter of analysis rather than algebra. Our tool will be the idea of a norm, which plays the role of the absolute value for scalar quantities. Our definitions in this section are for real matrices. For complex matrices and vectors, the only change needed is to replace transpose ^{T} with conjugate transpose *.

A vector **norm** $\|\cdot\|$ is a function $\mathbf{R}^n \mapsto \mathbf{R}$ with the following properties:

1. $\|\mathbf{x}\| \ge 0$ for all $\mathbf{x} \in \mathbf{R}^n$

2. $\|\mathbf{x}\| = 0$ if and only if $\mathbf{x} = \mathbf{0}$

3. $\|\alpha\mathbf{x}\| = |\alpha| \, \|\mathbf{x}\|$ for any $\mathbf{x} \in \mathbf{R}^n$ and $\alpha \in \mathbf{R}$

4. $\|\mathbf{x} + \mathbf{y}\| \le \|\mathbf{x}\| + \|\mathbf{y}\|$ for all $\mathbf{x}, \mathbf{y} \in \mathbf{R}^n$ (the triangle inequality)

A norm is essentially a rule for assigning distances in n-space. Just as $\|\mathbf{x}\| = \|\mathbf{x} - \mathbf{0}\|$ is the distance from $\mathbf{x}$ to the origin, $\|\mathbf{x} - \mathbf{y}\|$ is the distance from $\mathbf{x}$ to $\mathbf{y}$. The three most common vector norms in

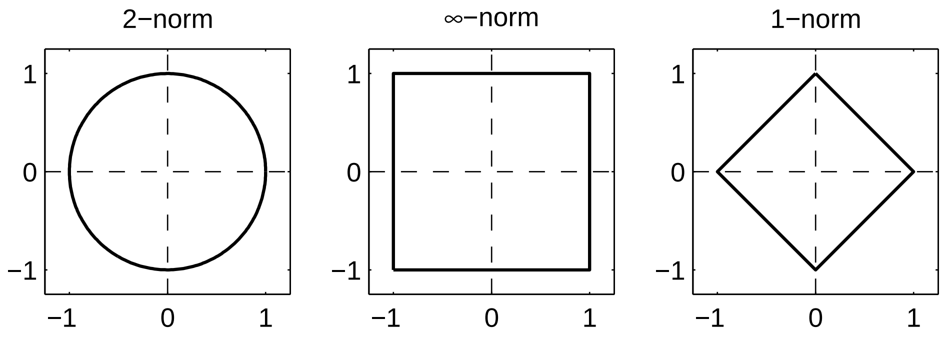


Figure 2.2: Unit "circles" in various vector norms.

practice are

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} |x_i|^2\right)^{\frac{1}{2}} = \sqrt{\mathbf{x}^T\mathbf{x}} \qquad \text{2-norm} \qquad (2.12)$$

$$\|\mathbf{x}\|_\infty = \max_{i=1,\ldots,n} |x_i| \qquad \infty\text{-norm, or max-norm} \qquad (2.13)$$

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i| \qquad \text{1-norm} \qquad (2.14)$$

We refer to a vector $\mathbf{x}$ satisfying $\|\mathbf{x}\| = 1$ as a **unit vector**. In $\mathbf{R}^2$, we can define a unit "circle" as the set of all unit vectors. The resulting graphs for our three major norms are shown in Figure 2.2. The 2-norm corresponds to usual Euclidean distance (or in some physical applications, energy) and is the most common choice. Outside of this section, norm symbols without a subscript in this text are understood to mean the 2-norm, unless the context says otherwise.

Example 2.11. Given the vector $\mathbf{x} = \begin{bmatrix} 2 & -3 & 1 & -1 \end{bmatrix}^T$, we have

$$\|\mathbf{x}\|_2 = \sqrt{4+9+1+1} = \sqrt{15}, \qquad \|\mathbf{x}\|_\infty = \max\{2,3,1,1\} = 3, \qquad \|\mathbf{x}\|_1 = 2+3+1+1 = 7.$$

In MATLAB one could use `norm(x,2)` or simply `norm(x)`, `norm(x,inf)`, and `norm(x,1)` to get these numerical values.

Using a vector norm we can say that a sequence of vectors $\mathbf{x}_n$ **converges** to $\mathbf{x}$ if

$$\lim_{n\to\infty} \|\mathbf{x}_n - \mathbf{x}\| = 0. \qquad (2.15)$$

By definition, a sequence that is convergent in the infinity norm must converge componentwise. However, the same is true for a convergent sequence in *any* norm, thanks to **norm equivalence**. Any two norms $\|\cdot\|_a$ and $\|\cdot\|_b$ on a finite-dimensional space are equivalent, in the sense that there exist constants C_1 and C_2 such that

$$C_1\|\mathbf{x}\|_a \le \|\mathbf{x}\|_b \le C_2\|x\|_a, \qquad (2.16)$$

for any $\mathbf{x} \in \mathbf{R}^n$. ($C_1$ and C_2 may depend on n.) Because of norm equivalence, convergence in any norm implies convergence in all norms.

Matrix norms

The most useful matrix norms are derived from or **induced** by vector norms. Using a vector norm $\|\cdot\|_a$ in $\mathbf{R}^m$ and a vector norm $\|\cdot\|_b$ in $\mathbf{R}^n$, we define for any $A \in \mathbf{R}^{m\times n}$

$$\|A\|_{a,b} = \max_{\|\mathbf{x}\|_b=1} \|A\mathbf{x}\|_a = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|A\mathbf{x}\|_a}{\|\mathbf{x}\|_b}. \tag{2.17}$$

(The last equality follows from linearity.) When $m = n$ and the matrix is square, usually the two vector norms $\|\cdot\|_a$ and $\|\cdot\|_b$ are the same norm, and we use just one subscript for the matrix norm. Thus,

$$\|A\|_2 = \max_{\|\mathbf{x}\|_2=1} \|A\mathbf{x}\|_2,$$

and so on. For the rest of this discussion we will work only with square matrices and omit subscripts when we want to refer to an unspecified norm; after this section, an unsubscripted norm is the 2-norm.

Theorem 2.3. *For any $n \times n$ matrix A and induced matrix norm,*

$$\|A\mathbf{x}\| \le \|A\| \cdot \|\mathbf{x}\|, \qquad \textit{for any } \mathbf{x} \in \mathbf{R}^n, \tag{2.18a}$$
$$\|AB\| \le \|A\| \cdot \|B\|, \qquad \textit{for any } B \in \mathbf{R}^{n\times n}, \tag{2.18b}$$
$$\|A^k\| \le \|A\|^k, \qquad \textit{for any integer } k \ge 0. \tag{2.18c}$$

Proof. The first is a direct consequence of the definition of $\|A\|$. It is trivial if $\mathbf{x} = \mathbf{0}$; otherwise,

$$\frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} \le \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = \|A\|.$$

Inequality (2.18b) then follows because

$$\|AB\mathbf{x}\| = \|A(B\mathbf{x})\| \le \|A\| \cdot \|B\mathbf{x}\| \le \|A\| \cdot \|B\| \cdot \|\mathbf{x}\|$$

and then

$$\|AB\| = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|AB\mathbf{x}\|}{\|\mathbf{x}\|} \le \max_{\mathbf{x}\neq\mathbf{0}} \|A\| \cdot \|B\| = \|A\| \cdot \|B\|.$$

Finally, (2.18c) results from repeated application of (2.18b). □

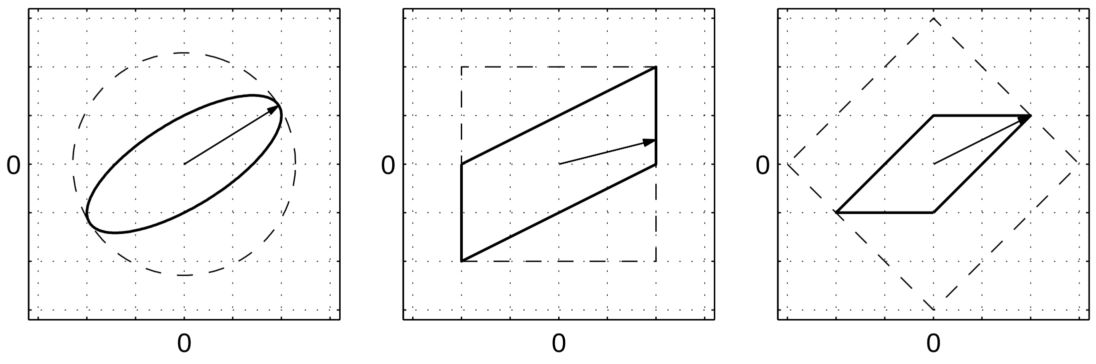
One can interpret the definition of a matrix norm geometrically. Each vector $\mathbf{x}$ on the unit "sphere" (as defined by the chosen vector norm) is mapped to its image $A\mathbf{x}$, and the norm of A is the radius of the smallest "sphere" that encloses all such images. In addition, two of the vector norms we have encountered induce matrix norms that are easy to compute from the matrix elements:

$$\|A\|_\infty = \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}|, \tag{2.19}$$

$$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}|. \tag{2.20}$$

In words, the infinity norm is the maximum *row* sum, and the 1-norm is the maximum *column* sum. (One way to keep this straight: the horizontal orientation of the ∞ suggests the row sum for the ∞-norm, while the vertical orietion of the 1 suggests the column sum of the 1-norm.) The 2-norm of a matrix, however, cannot easily be computed from the matrix elements.

Example 2.12. Let $A = \begin{bmatrix} 2 & 0 \\ 1 & -1 \end{bmatrix}$. Each of the three unit "circles" in Figure 2.2 is mapped under A to the following three shapes (solid curves):



The dashed curves show the smallest enclosing circles in each case, and the arrows indicate vectors whose images just touch the enclosing circles. Using MATLAB, we get

```
>> norm(A)

ans =
   2.2882e+00

>> norm(A,inf)

ans =
     2

>> norm(A,1)

ans =
     3
```

Problems

2.6.1. Why is the vector 1-norm also called the "taxicab norm?"

2.6.2. Show that for all vectors $\mathbf{x} \in \mathbf{R}^n$,

$$\text{(a) } \|\mathbf{x}\|_\infty \le \|\mathbf{x}\|_2, \qquad \text{(b) } \|\mathbf{x}\|_2 \le \|\mathbf{x}\|_1.$$

2.6.3. Show that for any vectors $\mathbf{x}$, $\mathbf{y}$ in $\mathbf{R}^n$, $|\mathbf{x}^T\mathbf{y}| \le \|\mathbf{x}\|_1\|\mathbf{y}\|_\infty$.

2.6.4. Let A be the matrix in Example 2.12.

(a) Find all unit vectors $\mathbf{x}$ such that $\|A\mathbf{x}\|_\infty = \|A\|_\infty$.

(b) Find all unit vectors $\mathbf{y}$ such that $\|A\mathbf{y}\|_1 = \|A\|_1$.

2.6.5. Prove the equivalence of the two formulas for a matrix norm in (2.17).

2.6.6. Show that for any permutation matrix P, $\|P\|_2 = 1$.

2.6.7. Show that for any induced matrix norm and nonsingular matrix A, $\|A^{-1}\| \ge (\|A\|)^{-1}$.

2.6.8. (a) Show that for any $\mathbf{v} \in \mathbf{R}^n$,

$$\|\mathbf{v}\|_p \ge \max_{i=1,\ldots,n} |v_i|,$$

where $p = 1, 2,$ or ∞.

(b) Show that for any $A \in \mathbf{R}^{n\times n}$,

$$\|A\|_p \ge \max_{i,j=1,\ldots,n} |a_{ij}|,$$

where $p = 1$, 2, or ∞. (Hint: For $p = 2$, rearrange (2.18a) for a well-chosen particular value of $\mathbf{x}$.)

2.6.9. Show that if D is a diagonal matrix, then $\|D\|_2 = \max_i |d_{ii}|$. You may assume the matrix is real and square, but that does not affect the result or the proof in any significant way. (Hint: Let $M = \max_i |d_{ii}|$. Proceed in two stages, showing that $\|D\|_2 \ge M$ and separately that $\|D\|_2 \le M$.)

2.6.10. Suppose that A is $n \times n$ and that $\|A\| < 1$ in some induced matrix norm.

(a) Show that $(I - A)$ is nonsingular. (Hint: Show that $(I - A)\mathbf{x} = \mathbf{0}$ for nonzero $\mathbf{x}$ implies that $\|A\| \ge 1$, using the definition of an induced matrix norm.)

(b) Show that $\lim_{m\to\infty} A^m = 0$. (For matrices as with vectors, we say $B_m \to L$ if $\|B_m - L\| \to 0$.)

(c) Use (a) and (b) to show the "geometric series"

$$(I - A)^{-1} = \sum_{k=0}^{\infty} A^k.$$

(Hint: Start with $\left(\sum_{k=0}^{m} A^k\right)(I - A)$ and take the limit.)

2.7 Conditioning and stability for linear systems

The stage is now set to consider the conditioning of solving the square linear system $A\mathbf{x} = \mathbf{b}$. Recall that the condition number of a problem is the relative change in the solution divided by a relative change in the data. In this case the data are A and $\mathbf{b}$, and the solution is $\mathbf{x}$.

For simplicity we start by perturbing $\mathbf{b}$ only. This means that $A\mathbf{x} = \mathbf{b}$ is perturbed to

$$A(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{b} + \Delta\mathbf{b}.$$

We seek to bound $\Delta\mathbf{x}$ in terms of $\Delta\mathbf{b}$. Manipulating the perturbed equation gives

$$\begin{aligned} A(\mathbf{x} + \Delta\mathbf{x}) &= \mathbf{b} + \Delta\mathbf{b} \\ A\mathbf{x} + A(\Delta\mathbf{x}) &= \mathbf{b} + \Delta\mathbf{b}; \end{aligned}$$

using the original equation $Ax = b$ and taking norms yields

$$\begin{aligned} A(\Delta\mathbf{x}) &= \Delta\mathbf{b} \\ \Delta\mathbf{x} &= A^{-1}(\Delta\mathbf{b}) \\ \|\Delta\mathbf{x}\| &\le \|A^{-1}\|\,\|\Delta\mathbf{b}\| \end{aligned}$$

Since $\mathbf{b} = A\mathbf{x}$ and hence $\|\mathbf{b}\| \le \|A\| \cdot \|\mathbf{x}\|$, we derive the condition number bound

$$\frac{\dfrac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|}}{\dfrac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|}} = \frac{\|\Delta\mathbf{x}\| \cdot \|\mathbf{b}\|}{\|\Delta\mathbf{b}\| \cdot \|\mathbf{x}\|} \le \frac{\Big(\|A^{-1}\|\,\|\Delta\mathbf{b}\|\Big)\Big(\|A\|\,\|\mathbf{x}\|\Big)}{\|\Delta\mathbf{b}\|\,\|\mathbf{x}\|} = \|A^{-1}\|\,\|A\|.$$

It is not hard to show that this bound is "tight," in the sense that the inequalities are in fact equalities for some choices of the data and the perturbation. This motivates the definition

$$\kappa(A) = \|A^{-1}\|\,\|A\| \tag{2.21}$$

as the **condition number** of the matrix A. Note that $\kappa(A)$ depends on the norm chosen; a subscript on κ such as 1, 2, or ∞ is used if clarification is needed.

So far we have only perturbed $\mathbf{b}$, but the derivation with respect to perturbations in A is similar. Now

$$\begin{aligned} (A + \Delta A)(\mathbf{x} + \Delta\mathbf{x}) &= \mathbf{b} \\ A\mathbf{x} + (\Delta A)\mathbf{x} + A(\Delta\mathbf{x}) + (\Delta A)(\Delta\mathbf{x}) &= \mathbf{b}. \end{aligned}$$

Assuming the perturbations are small (in the limit, infinitesimal), we ignore the doubly small $(\Delta A)(\Delta\mathbf{x})$. Now, using $A\mathbf{x} = \mathbf{b}$, we have $(\Delta A)x + A(\Delta\mathbf{x}) = \mathbf{0}$, or

$$\begin{aligned} \Delta\mathbf{x} &= -A^{-1}(\Delta A)\mathbf{x} \\ \|\Delta\mathbf{x}\| &\le \|A^{-1}\|\,\|\Delta A\|\,\|\mathbf{x}\|. \end{aligned}$$

Thus the condition number is

$$\frac{\|\Delta\mathbf{x}\|\,\|A\|}{\|\Delta A\|\,\|\mathbf{x}\|} \le \frac{\left(\|A^{-1}\|\,\|\Delta A\|\,\|\mathbf{x}\|\right)\|A\|}{\|\Delta A\|\,\|\mathbf{x}\|} = \kappa(A).$$

We conclude that the condition number of solving $A\mathbf{x} = \mathbf{b}$ is the same regardless of whether $\mathbf{b}$ or A is perturbed:

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \kappa(A)\frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|} \tag{2.22a}$$

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \kappa(A)\frac{\|\Delta A\|}{\|A\|} \tag{2.22b}$$

Equation (2.22a) is true without qualification; equation (2.22b) is strictly true only for infinitesimal perturbations ΔA.

Observe that for any induced matrix norm,

$$1 = \|I\| = \|AA^{-1}\| \le \|A\|\,\|A^{-1}\| = \kappa(A).$$

A condition number of 1 is the best we can hope for—the relative perturbation of the solution has the same size as that of the data. On the other hand, a larger condition number of size 10^t indicates that in floating point arithmetic, roughly t digits are "lost" in computing the solution $\mathbf{x}$. If $\kappa(A) > \epsilon_M^{-1}$ on a machine with unit roundoff ϵ_M, then for computational purposes the matrix is singular, because all of the available digits in the solution are expected to be lost. If A is exactly singular, it is customary to say that $\kappa(A) = \infty$.

MATLAB has a function `cond` to compute $\kappa_2(A)$; the algorithm for this is discussed more in section 4.6. More generally, `cond(A,p)` computes $\kappa_p(A)$. Since finding the condition number can be very time-consuming for large matrices, there is also a function `condest` that estimates $\kappa_1(A)$ quickly. Whenever MATLAB solves a linear system, it performs this estimate and prints a warning if the value is near `1/eps`.

Example 2.13. We study the bound offered by the matrix condition number. It so happens that random upper triangular matrices tend to have large condition numbers on average.

```
>> randn('state',0)
>> A = triu(randn(16));
>> x_exact = (1:16)';
>> b = A*x_exact;
>> x = A\b;
>> norm(x-x_exact)/norm(x_exact)

ans =

   1.7413e-10
```

We see that about six digits were lost in finding $\mathbf{x}$. We can show that this is well within the amount suggested by (2.22) for perturbations by machine roundoff:

```
>> cond(A)

ans =

   3.5188e+08

>> cond(A)*eps

ans =

   7.8133e-08
```

The error is well within the bound provided by the condition number.

Now we use a larger matrix that is so ill-conditioned that it might as well be singular.

```
>> A = triu(randn(64));
>> x_exact = (1:64)'; b = A*x_exact;
>> x = A\b;
>> norm(x-x_exact)/norm(x_exact)

ans =

  349.4345

>> cond(A)

ans =

   1.8658e+17
```

In this case the computed $\mathbf{x}$ held essentially no correct information about the true solution.

As always with conditioning, it is an inherent property of the problem, not an effect created by a particular algorithm. If you are trying to solve a linear system with a matrix whose condition number is 10^{16}, you must expect that the solution will be up to 10^{16} times less accurate than the data in the problem, for all sources of error.

Residuals

Suppose $\tilde{\mathbf{x}}$ is an estimate to the true solution of $A\mathbf{x} = \mathbf{b}$. If we knew the error $\mathbf{x} - \tilde{\mathbf{x}}$, then we would know the solution $\mathbf{x}$ as well. As a practical matter, a more constructive way to estimate the error is the **residual**, defined as

$$\mathbf{r} = \mathbf{b} - A\tilde{\mathbf{x}}. \qquad (2.23)$$

Given $\tilde{\mathbf{x}}$, we can always compute the residual, and obviously a zero residual means that $\tilde{\mathbf{x}} = \mathbf{x}$, the exact solution. But does a small residual mean that $\tilde{\mathbf{x}}$ is close to $\mathbf{x}$?

First, note that

$$\mathbf{r} = A(A^{-1}\mathbf{b} - \tilde{\mathbf{x}}) = A(\mathbf{x} - \tilde{\mathbf{x}}),$$

so the residual is A times the error. From this we obtain $\mathbf{x} - \tilde{\mathbf{x}} = A^{-1}\mathbf{r}$ and then the bound

$$\|\tilde{\mathbf{x}} - \mathbf{x}\| \le \|A^{-1}\| \, \|\mathbf{r}\|.$$

We can reconnect with (2.22) by the definition $\Delta x = \tilde{\mathbf{x}} - \mathbf{x}$, in which case $\Delta \mathbf{b} = A(x + \Delta x) - \mathbf{b} = A\Delta x = -\mathbf{r}$. Hence (2.22) is equivalent to

$$\frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \le \kappa(A) \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}. \tag{2.24}$$

Equation (2.24) says that there is a potential gap, given by the matrix condition number, between the relative residual and the relative error. Ill-conditioned systems are precisely ones for which a small residual may not mean small error. (Look again at Figure 2.1 in this light.) To put it another way: In solving a linear system, all that can be expected is that the *residual*, not the error, be small.

Stability

How stable is *LU* factorization? We already know that if pivoting is not done, the method can completely fail even for perfectly conditioned problems. Hence we must require row pivoting for stability. Even with this in place, though, there is another alarming example.

Example 2.14. Let

$$A = \begin{bmatrix} 1 & & & & 1 \\ -1 & 1 & & & 1 \\ -1 & -1 & 1 & & 1 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ -1 & \cdots & -1 & -1 & 1 \end{bmatrix}.$$

We try this matrix in MATLAB for the 30×30 case:

```
>> A = toeplitz([1 -ones(1,29)],[1 zeros(1,29)]); A(:,30)=1;
>> x_exact = randn(30,1);  b = A*x_exact;
>> x = A\b;
>> norm(x-x_exact)/norm(x_exact)

ans =
   2.2607e-09
```

About seven digits of accuracy have been lost. But this is *not* due to conditioning, nor is the residual small.

```
>> cond(A)

ans =
   1.3318e+01

>> norm(A*x-b)

ans =
   1.9838e-08
```

It is not hard to show in the $n \times n$ case that the LU factorization process requires no row swaps for pivoting and produces

$$L = \begin{bmatrix} 1 & & & & 0 \\ -1 & 1 & & & 0 \\ -1 & -1 & 1 & & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ -1 & \cdots & -1 & -1 & 1 \end{bmatrix}, \qquad U = \begin{bmatrix} 1 & & & & 1 \\ 0 & 1 & & & 2 \\ 0 & 0 & 1 & & 4 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 2^{n-1} \end{bmatrix}.$$

One can verify from these that

$$\kappa_1(L) = n2^{n-1}, \qquad \kappa_1(U) = 2^n - 1, \qquad \kappa_1(A) = n.$$

Hence the factorization algorithm for $A\mathbf{x} = \mathbf{b}$, which solves $L\mathbf{z} = \mathbf{b}$ and $U\mathbf{x} = \mathbf{z}$, can amplify relative errors by factors on the order of $2^{30} \approx 10^9$. Or from a more intuitive point of view, the product LU relies on subtractive cancellation of the large elements of U to produce A.

This example suggests that row pivoted LU factorization (Gaussian elimination) can be extremely unstable. In the earliest days of digital computation, many experts were pessimistic about the prospects for Gaussian elimination for all but the smallest systems. However, decades of experience have erased all traces of such doubts. It turns out that examples like Example 2.14, which represent the worst case, are stupendously rare both among randomly chosen matrices and those occurring in practice. As an everyday matter, the level of concern is so low that MATLAB and other software skip the easy step of checking for the element growth in the LU factors that signals instabiity.

Problems

2.7.1. The purpose of this problem is to verify, as in Example 2.13, the error bound

$$\frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \le \kappa(A) \frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|}.$$

Here $\tilde{\mathbf{x}}$ is a numerical approximation to the exact solution $\mathbf{x}$, and $\Delta\mathbf{b}$ is an unknown perturbation caused by machine roundoff. We will assume that the relative perturbation of $\mathbf{b}$ is of size roughly `eps` in MATLAB.

For each $n = 10, 20, \dots, 70$ let

```
A = gallery('prolate',n,0.4);
```

The exact solution $\mathbf{x}$ should have components $x_k = k/n$ for $k = 1, \dots, n$, and you should define $\mathbf{b}$ as $A\mathbf{x}$. Then $\tilde{\mathbf{x}}$ is the solution produced numerically by the backslash.

Make a table of the results, including columns for n, the relative error in $\tilde{\mathbf{x}}$, the condition number of A, and the right-hand side of the inequality above. Does the inequality hold in every case?

2.7.2. (a) Show that for $n \times n$ matrices A and B, $\kappa(AB) \le \kappa(A)\kappa(B)$. (Both A and B may be singular.)

(b) Show by means of an example that the result of part (a) cannot be an equality in general.

2.7.3. Let D be a diagonal $n \times n$ matrix, not necessarily invertible. Show that

$$\kappa_2(D) = \frac{\max_i |d_{ii}|}{\min_i |d_{ii}|}.$$

(Hint: See problem 9 in section 2.6.)

2.7.4. Verify the assertions about $\kappa_1(L)$, $\kappa_1(U)$, and $\kappa_1(A)$ in Example 2.14. You may use `inv` in MATLAB on small instances in order to produce the inverses of the matrices.

2.8 Special types of matrices

There are certain important special types of matrices for which we can take significant shortcuts in the LU factorization process.

Diagonally dominant matrices

An $n \times n$ matrix A is **diagonally dominant** if

$$|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}|, \qquad \text{for each } i = 1, \dots, n.$$

For example, the matrix

$$A = \begin{bmatrix} 3 & 1 & 0 & 0 \\ -1 & 3 & 1 & 0 \\ 0 & -1 & 3 & 1 \\ 0 & 0 & 1 & 3 \end{bmatrix}$$

is diagonally dominant, but

$$B = \begin{bmatrix} 2 & 1 & 1 \\ -1 & 3 & 4 \\ 0 & 0 & 1 \end{bmatrix}$$

is not. Diagonally dominant matrices do arise in important applications. It turns out that a diagonally dominant matrix is guaranteed to be nonsingular. Furthermore, for such matrices row pivoting is not required; $A = LU$ is just as good as $A = P^T LU$.

Banded matrices

We say that the $n \times n$ matrix A has **upper bandwidth** p and **lower bandwidth** q if $a_{ij} = 0$ whenever $j - i \geq p$ or $i - j \geq q$. The **bandwidth** of A is $p + q - 1$. If p and q are much smaller than n, we say that A is **banded**. All the nonzero entries of a banded matrix are close to the diagonal. The most important type of banded matrix occurs when $p = q = 2$ and is called **tridiagonal**. Such a matrix has the form

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \cdots & 0 \\ 0 & a_{32} & a_{33} & a_{34} & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & \cdots & 0 & 0 & a_{n,n-1} & a_{nn} \end{bmatrix}.$$

If we forgo pivoting, Gaussian elimination can exploit bandedness. Consider the tridiagonal case. In the first stage, only one nonzero multiplier is needed to get rid of a_{21}. After this stage the nonzero pattern will be

$$\begin{bmatrix} \times & \times & 0 & 0 & \cdots & 0 \\ 0 & \times & \times & 0 & \cdots & 0 \\ 0 & \times & \times & \times & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \times & \times & \times \\ 0 & \cdots & 0 & 0 & \times & \times \end{bmatrix}$$

Now again only one nonzero multiplier is needed, to eliminate the $(3,2)$ entry. This pattern persists all the way to the end. Since the multipliers are the elements of L in the LU factorization, this means that L will have lower bandwidth equal to 2. It is also clear that U will have upper bandwidth 2. This observation holds for a general banded matrix:

Theorem 2.4. *A matrix with upper and lower bandwidths p and q has an LU factorization (without pivoting) in which L has lower bandwidth p and U has upper bandwidth q.*

When arithmetic operations on zeros are skipped, the operation count for factoring banded matrices drops dramatically. In fact if p and q are small numbers then the whole process of factorization and triangular substitution takes just $O(n)$ flops. Keep in mind, however, that the savings for banded matrices are based on the absence of pivoting. If the matrix is also diagonally dominant, then we will be safe, but otherwise the method could be arbitrarily unstable.

Matrices in which many or most entries are zero in a reliable pattern are called **sparse** (as opposed to **dense** or full). As in the special case of banded matrices, the zero structure of a sparse matrix can often be exploited to save tremendously on arithmetic operations. Sparse matrices are intrinsically supported in MATLAB and will play a major role in Chapter 5.

Symmetric positive definite matrices

A real matrix A is called **symmetric positive definite** (or SPD, or just positive definite) if it satisfies

1. $A = A^T$, and
2. $\mathbf{x}^T A\mathbf{x} > 0$ whenever $\mathbf{x} \in \mathbf{R}^n$ and $\mathbf{x} \neq \mathbf{0}$.

Note in the second condition that $\mathbf{x}^T A\mathbf{x}$ is the product of $1 \times n$, $n \times n$, and $n \times 1$ matrices, and hence is a scalar. This condition is usually difficult to check directly.

An SPD matrix is in some ways analogous to a positive number. For instance, it has a "square root."

Theorem 2.5. *An SPD matrix A can be factored as $A = R^T R$, where R is upper triangular with positive diagonal entries.*

This is often called the **Cholesky factorization**. Alternatively we may write $A = LDL^T$ for a unit lower triangular L and a diagonal matrix D with positive numbers on the diagonal.

Since there are basically only half as many entries to find in the factorization of an SPD matrix, Cholesky factorization takes $\sim \frac{1}{3}n^3$ flops. One nice feature of the algorithm is that it fails if and only if the matrix A is not SPD, so it serves as a practical test of positive definiteness. The built-in function `chol` computes a Cholesky factorization (or reports failure) in MATLAB. This function does not check symmetry, however.

Problems

2.8.1. Which of the following matrices is/are diagonally dominant?

$$A = \begin{bmatrix} 3 & 1 & 0 & 0 & 1 \\ 0 & -2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 4 & -1 \\ 0 & 0 & 3 & 0 & 6 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

2.8.2. Which of the following matrices is/are SPD? You may use MATLAB.

$$A = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 4 & 5 \\ -1 & 5 & 10 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 4 & 5 \\ -1 & 5 & 10 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 4 & 5 \\ 1 & 5 & 1 \end{bmatrix}.$$

2.8.3. Rewrite Function 2.2 to accept arguments `p` and `q` for upper and lower bandwidth, and to perform banded LU factorization (without pivoting). Create a nontrivial diagonally dominant 6×6 matrix to test your function against the built-in `lu`.

2.8.4. In this problem you will explore the backslash and how it handles banded matrices. To do this you will generate tridiagonal matrices using the following code:

```
A = triu( tril(rand(n),1), -1);
A(1:n+1:end) = a;
```

The resulting matrix is $n \times n$, with each entry on the sub- and superdiagonals chosen randomly from $(0, 1)$ and each diagonal entry equalling `a`.

(a) Write a script that solves 200 linear systems whose matrices are generated as above, with $n = 1000$ and `a` $= 2$. (Use randomly generated right-hand side vectors.) Record the total time used by the solution process `A\b` only, using the built-in `cputime`.

(b) Repeat the experiment of part (a), but add the command `A=sparse(A);` right after the two lines above. How does the timing change?

(c) Repeat parts (a) and (b) with `a` $= 1$. In which case is there a major change?

(d) Based on these observations, state a hypothesis on how backslash solves tridiagonal linear systems given in standard dense form and in sparse form. (Hint: What is the mathematical significance of `a` $= 2$ versus `a` $= 1$?)

2.8.5. Prove that if A is any real nonsingular square matrix, then $A^T A$ is SPD.