CHAPTER 3

SOME FUNDAMENTAL TOOLS AND CONCEPTS FROM NUMERICAL LINEAR ALGEBRA

Topics covered

- Floating Point Numbers and Errors in Computations
- LU Factorization Using Gaussian Elimination
- QR Factorization Using Householder and Givens Matrices
- Numerical Solution of the Algebraic Linear Systems and Least-Squares Problems
- The Singular Value Decomposition (SVD)

3.1 INTRODUCTION

In this chapter, we introduce some fundamental concepts and techniques of numerical linear algebra which, we *believe, are essential for in-depth understanding of computational algorithms for control problems, discussed in this book.* The basic concepts of floating point operations, numerical stability of an algorithm, conditioning of a computational problem, and their effects on the accuracy of a solution obtained by a certain algorithm are introduced first.

Three important matrix factorizations: ***LU***, ***QR***, and the **singular value decomposition** (SVD), and their applications to solutions of algebraic linear systems, linear least-squares problems, and eigenvalue problems are next described in details.

The method of choice for the linear system problem is the LU factorization technique obtained by Gaussian elimination with partial pivoting (Section 3.4). The method of choice for the symmetric positive definite system is the **Cholesky factorization technique (Algorithm 3.4.1).**

The *QR* factorization of a matrix is introduced in the context of the least-squares solution of a linear system; however, it also forms the core of the ***QR* iteration technique, which is the method of choice for eigenvalue computation**. The *QR* iteration technique itself for eigenvalue computation is described in **Chapter 4**. Two numerically stable methods for the *QR* factorization, namely, **Householder's and Givens' methods are described in Section 3.6**. Householder's method is slightly cheaper than Givens' method for sequential computations, but the latter has computational advantages in parallel computation setting.

The SVD has nowadays become an essential tool for determining the **numerical rank**, the **distance of a matrix from a matrix of immediate lower rank, finding the orthonormal basis and projections**, etc. This important matrix factorization is described in **Section 3.9. The SVD is also a reliable tool for computing the least-squares solution to $Ax = b$.**

A reliable and widely used computational technique for computing the SVD of a matrix is described in **Chapter 4**.

3.2 FLOATING POINT NUMBERS AND ERRORS IN COMPUTATIONS

3.2.1 Floating Point Numbers

Most scientific and engineering computations on a computer are performed using **floating point arithmetic**. Computers may have different bases, though base 2 is most common.

A t-digit **floating point** number in base β has the form:

$$x = m \cdot \beta^e,$$

where m is a t-digit fraction called **mantissa** and e is called **exponent**.

If the first digit of the mantissa is different from zero, then the floating point number is called **normalized**. Thus, 0.3457×10^5 is a 4-digit normalized decimal floating number, whereas 0.03475×10^6 is a five-digit unnormalized decimal floating point number.

The number of digits in the mantissa is called **precision**. On many computers, it is possible to manipulate floating point numbers so that a number can be represented with about twice the usual precision. Such a precision is called **double precision**.

Most computers nowadays conform to the IEEE floating point standard (ANSI/IEEE standard 754-1985). For a single-precision, IEEE standard recommends about 24 binary digits and for a double precision, about 53 binary digits. Thus, IEEE standard for **single precision provides approximately 7 decimal digits of accuracy**, since $2^{-23} \cong 1.2 \times 10^{-7}$, and **double precision provides approximately 16 decimal digits of accuracy**, since $2^{-52} \approx 2.2 \times 10^{-16}$.

Note: Although computations with double precision increase accuracy, they require more computer time and storage.

On each computer, there is an allowable range of the exponent e: L, the minimum; U, the maximum. L **and** U **vary from computer to computer.**

If, during computations, the computer produces a number whose exponent is too large (too small), that is, it is outside the permissible range, then we say that an **overflow** (**underflow**) has occurred.

Overflow is a serious problem; for most systems, the result of an overflow is $\pm\infty$. Underflow is usually considered less serious. On most computers, when an underflow occurs, the computed value is set to zero, and then computations proceed. **Unless otherwise stated, we will use only decimal arithmetic.**

3.2.2 Rounding Errors

If a computed result of a given real number is not machine representable, then there are two ways it can be represented in the machine. Consider the machine representation of the number

$$\pm \cdot d_1 \cdots d_t d_{t+1} \cdots .$$

Then the first method, **chopping**, is the method in which the digits from d_{t+1} on are simply chopped off. The second method is **rounding**, in which the digits d_{t+1} through the rest are not only chopped off, but the digit d_t is also rounded up or down depending on whether $d_{t+1} \geq 5$ or $d_{t+1} < 5$.

We will denote the floating point representation of a real number x by $\text{fl}(x)$.

Example 3.2.1. (*Rounding*) Let $x = 3.141596$.

$$t = 2\text{: } \text{fl}(x) = 3.1,$$
$$t = 3\text{: } \text{fl}(x) = 3.14,$$
$$t = 4\text{: } \text{fl}(x) = 3.142.$$

A useful measure of error in computation is the **relative error**.

Definition 3.2.1. *Let $\hat{x}$ denote an approximation of x. Then the relative error is $|\hat{x} - x|/|x|$, $x \neq 0$.*

We now give an expression for the relative error in representing a real number x by its floating point representation $\text{fl}(x)$. Proof of Theorem 3.2.1 can be found in Datta (1995, p. 47).

Theorem 3.2.1. *Let* $\mathrm{fl}(x)$ *denote the floating point representation of a real number* x *in base* β. *Then,*

$$\frac{|\mathrm{fl}(x) - x|}{|x|} \leq \mu = \begin{cases} \frac{1}{2}\beta^{1-t} \text{ for rounding,} \\ \beta^{1-t} \text{ for chopping.} \end{cases} \tag{3.2.1}$$

Definition 3.2.2. *The number* μ *in the above theorem is called the* **machine precision, computer epsilon**, *or* **unit roundoff error**. *It is the smallest positive floating point number such that*

$$\mathrm{fl}(1 + \mu) > 1.$$

The number μ **is usually of the order** 10^{-16} **and** 10^{-7} *(on most machines) for double and single precisions computations, respectively. For example, for the IBM* 360 *and* 370, $\beta = 16, t = 6, \mu = 4.77 \times 10^{-7}$.

Definition 3.2.3. *The* ***significant digits*** *in a number are the number of digits starting with the first nonzero digit.*

For example, the number 1.5211 *has five significant digits, whereas the number* 0.0231 *has only three.*

3.2.3 Laws of Floating Point Arithmetic

The formula (3.2.1) can be written as

$$\mathrm{fl}(x) = x(1 + \delta),$$

where $|\delta| \leq \mu$.

Assuming that the IEEE standard holds, we can easily derive the following simple **laws of floating point arithmetic**.

Theorem 3.2.2. *Laws of Floating Point Arithmetic. Let* x *and* y *be two floating point numbers, and let* $\mathrm{fl}(x + y)$, $\mathrm{fl}(x - y)$, $\mathrm{fl}(xy)$, *and* $\mathrm{fl}(x/y)$ *denote, respectively, the computed sum, difference, product, and quotient. Then,*

1. $\mathrm{fl}(x \pm y) = (x \pm y)(1 + \delta)$, *where* $|\delta| \leq \mu$.
2. $\mathrm{fl}(xy) = (xy)(1 + \delta)$, *where* $|\delta| \leq \mu$.
3. *if* $y \neq 0$, *then* $\mathrm{fl}(x/y) = (x/y)(1 + \delta)$, *where* $|\delta| \leq \mu$.

On computers that do not use the IEEE standard, the following floating point law of addition might hold:

4. $\mathrm{fl}(x + y) = x(1 + \delta_1) + y(1 + \delta_2)$, where $|\delta_1| \leq \mu$ and $|\delta_2| \leq \mu$.

Example 3.2.2. Let $\beta = 10, t = 4$.

$$x = 0.1112, y = 0.2245 \times 10^5,$$
$$xy = 0.24964 \times 10^4,$$
$$\text{fl}(xy) = 0.24960 \times 10^4.$$

Thus, $|\text{fl}(xy) - xy| = 0.4000$ and $|\delta| = 1.7625 \times 10^{-4} < \frac{1}{2} \times 10^{-3}$.

3.2.4 Catastrophic Cancellation

A phenomenon, called **catastrophic cancellation**, occurs when two numbers of approximately the same size are subtracted. Very often significant digits are lost in the process.

Consider the example of computing $f(x) = e^x - 1 - x$ for $x = 0.01$. In five digit arithmetic $a = e^x - 1 = 1.0101 - 1 = 0.0101$. Then the computed value of $f(x) = a - x = 0.0001$, whereas the true answer is 0.000050167.

Note that even though the subtraction was done accurately, the final result was wrong. Indeed, subtractions in most cases can be done exactly, cancellation only signals that the error must have occurred in previous steps. **Fortunately, often cancellation can be avoided by rearranging computations**. For the example under consideration, if e^x were computed using the convergent series $e^x = 1 + x + x^2/2! + x^3/3! + \cdots$, then the result would have been 0.000050167, which is correct up to five significant digits.

For details and examples, see Datta (1995, pp. 43–61). See also Stewart (1998, pp. 136–138) for an illuminating discussion on this topic.

3.3 CONDITIONING, EFFICIENCY, STABILITY, AND ACCURACY

3.3.1 Algorithms and Pseudocodes

Definition 3.3.1. *An* **algorithm** *is an ordered set of operations, logical and arithmetic, which when applied to a computational problem defined by a given set of data, called the* **input data**, *produces a solution to the problem. A solution comprises of a set of data called the* **output data**.

In this book, for the sake of convenience and simplicity, we will very often describe algorithms by means of **pseudocodes**, which can easily be translated into computer codes. Here is an illustration.

3.3.2 Solving an Upper Triangular System

Consider the system

$$Ty = b,$$

where $T = (t_{ij})$ is a nonsingular upper triangular matrix and $y = (y_1, y_2, \ldots, y_n)^T$ and $b = (b_1, \ldots, b_n)^T$.

Algorithm 3.3.1. *Back Substitution Method for Upper Triangular System*
Input. *T—An $n \times n$ nonsingular upper triangular matrix, b—An n-vector.*
Output. *The vector $y = (y_1, \ldots, y_n)^T$ such that $Ty = b$.*

Step 1. *Compute* $y_n = \dfrac{b_n}{t_{nn}}$

Step 2. *For* $i = n-1, n-2, \ldots, 2, 1$ *do*

$$y_i = \frac{1}{t_{ii}} \left(b_i - \sum_{j=i+1}^{n} t_{ij} y_j \right)$$

End

Note: When $i = n$, the summation $(\sum)$ is skipped.

3.3.3 Solving a Lower Triangular System

A lower triangular system $Ly = b$ can be solved in an analogous manner. The process is known as the **forward substitution method**. Let $L = (l_{ij})$, and $b = (b_1, b_2, \ldots, b_n)^T$. Then starting with y_1, y_2 through y_n are computed recursively.

3.3.4 Efficiency of an Algorithm

Two most desirable properties of an algorithm are: **Efficiency and Stability**.

The *efficiency of an algorithm* is measured by the amount of computer time consumed in its implementation.

A theoretical and very crude measure of efficiency is the number of floating point operations (**flops**) needed to implement the algorithm. Too much emphasis should not be placed on exact flop-count when comparing the efficiency of two algorithms.

Definition 3.3.2. *A floating point operation of* flop *is a floating point operation: $+, -, *,$ or $/$.*

The Big O Notation

An algorithm will be called an $O(n^p)$ algorithm if the dominant term in the operations count of the algorithm is a multiple of n^p. Thus, the solution of a triangular system is an $O(n^2)$ algorithm; because it requires n^2 flops.

Notation for Overwriting and Interchange

We will use the notation:

$$a \equiv b$$

to denote that "***b* overwrites *a***". Similarly, if two computed quantities a and b are interchanged, they will be written symbolically

$$a \leftrightarrow b.$$

3.3.5 The Concept of Numerical Stability

The accuracy or the inaccuracy of the computed solution of a problem usually depends upon two important factors: **the stability or the instability of the algorithm used to solve the problem** and **the conditioning of the problem** (i.e., how sensitive the problem is to small perturbations).

We first define the concept of stability of an algorithm. In the next section, we shall talk about the conditioning of a problem.

The study of stability of an algorithm is done by means of **roundoff error analysis**. There are two types: **backward error analysis** and **forward error analysis**.

In forward analysis, an attempt is made to see how the computed solution obtained by the algorithm differs from the exact solution based on the same data.

On the other hand, backward analysis relates the error to the data of the problem rather than to the problem's solution.

Definition 3.3.3. *An algorithm is called* **backward stable** *if it produces an exact solution to a* **nearby** *problem; that is, a backward algorithm exactly solves a problem whose data are close to the original data.*

Backward error analysis, popularized in the literature by J.H. Wilkinson (1965), is now widely used in matrix computations and using this analysis, the stability (or instability) of many algorithms in numerical linear algebra has been established in recent years. **In this book, by "stability" we will imply "backward stability," unless otherwise stated.**

As a simple example of backward stability, consider again the problem of computing the sum of two floating point numbers x and y. We have seen before that

$$\text{fl}(x + y) = (x + y)(1 + \delta) = x(1 + \delta) + y(1 + \delta) = x' + y'.$$

Thus, the computed sum of two floating point numbers x and y is the exact sum of another two floating point numbers x' and y'. Because $|\delta| \leq \mu$, both x' and y' are close to x and y, respectively. Thus, we conclude that **the operation of adding two floating-point numbers is stable**. Similarly, *floating-point subtraction, multiplication, and division are also backward stable.*

Example 3.3.1. (*A Stable Algorithm for Linear Systems*) Solution of an upper triangular system by Back substitution.

The back-substitution method for solving an upper triangular system $Tx = b$ is backward stable. It can be shown that the computed solution $\hat{x}$ satisfies

$$(T + E)\hat{x} = b,$$

where $|e_{ij}| \leq c\mu|t_{ij}|$, $i, j = 1, \ldots, n$ and c is a constant of order unity. Thus, the computed solution $\hat{x}$ solves exactly a nearby system. The back-substitution process is, therefore, backward stable.

Remark

- The forward substitution method for solving a lower triangular system has the same numerical stability property as above. **This algorithm is also stable.**

Example 3.3.2. (*An Unstable Algorithm for Linear Systems*) Gaussian elimination without pivoting.

It can be shown (**see Section 3.5.2**) that Gaussian elimination without pivoting applied to the linear system $Ax = b$ produces a solution $\hat{x}$ such that

$$(A + E)\hat{x} = b$$

with $\|E\|_\infty \leq cn^3\rho\|A\|_\infty\mu$. The number ρ above, called the **growth factor**, can be arbitrarily very large. When it happens, the computed solution $\hat{x}$ does not solve a nearby problem.

Example 3.3.3. (*An Unstable Algorithm for Eigenvalue Computations*) Finding the eigenvalues of a matrix via its characteristic polynomial. **The process is numerically unstable.**

There are two reasons: First, the characteristic polynomial of a matrix may not be obtained in a numerically stable way (**see Chapter 4**); second, the zeros of a polynomial can be extremely sensitive to small perturbations of the coefficients.

A well-known example of zero-sensitivity is the Wilkinson polynomial $P_n(x) = (x - 1)(x - 2)\cdots(x - 20)$. A small perturbation of 2^{-23} to the coefficient of x^{19} changes some of the zeros significantly: some of them even become complex. See Datta (1995, pp. 81–82) for details.

Remark

- This example shows that the **eigenvalues of a matrix should not be computed by finding the roots of its characteristic polynomial.**

3.3.6 Conditioning of the Problem and Perturbation Analysis

From the preceding discussion, we should not form the opinion that if a stable algorithm is used to solve a problem, then the computed solution will be accurate.

As said before, a property of the problem called **conditioning** also contributes to the accuracy or inaccuracy of the computed result.

The conditioning of a problem is a property of the problem itself. It is concerned with how the solution of the problem will change if the input data contains some impurities. This concern arises from the fact that in practical applications, the data very often come from some experimental observations where the measurements can be subjected to disturbances (or "noises") in the data. There are other sources of error also, such as **roundoff errors, discretization errors**, and so on. Thus, when a numerical analyst has a problem in hand to solve, he or she must frequently solve the problem not with the original data, but with data that have been perturbed. The question naturally arises: **What effects do these perturbations have on the solution?**

A theoretical study done by numerical analysts to investigate these effects, which is independent of the particular algorithm used to solve the problem, is called **perturbation analysis**. This study helps us detect whether a given problem is "**bad**" or "**good**" in the sense of whether small perturbations in the data will create a large or small change in the solution. Specifically we use the following standard definition.

Definition 3.3.4. *A problem (with respect to a given set of data) is called an* **ill-conditioned** *or* **badly conditioned** *problem if a small relative error in data can cause a large relative error in the computed solution, regardless of the method of solution. Otherwise, it is called* **well-conditioned**.

Suppose a problem P is to be solved with an input c. Let $P(c)$ denote the value of the problem with the input c. Let δ_c denote the perturbation in c. Then P is said to be ill-conditioned for the input data c if the relative error $|P(c + \delta_c) - P(c)|/|P(c)|$ is much larger than the relative error in the data $|\delta_c|/|c|$

Note: The definition of conditioning is data-dependent. Thus, a problem that is ill-conditioned for *one* set of data could be well-conditioned for *another* set.

3.3.7 Conditioning of the Problem, Stability of the Algorithm, and Accuracy of the Solution

As stated in the previous section, the conditioning of a problem is a property of the problem itself, and has nothing to do with the algorithm used to solve the problem. To a user, of course, the accuracy of the computed solution is of primary importance. However, the accuracy of a computed solution by a given algorithm is directly connected with both the stability of the algorithm and the conditioning of the problem. **If the problem is ill-conditioned, no matter how stable the algorithm is, the accuracy of the computed result cannot be guaranteed.**

On the other hand, if a backward stable algorithm is applied to a well-conditioned problem, the computed result will be accurate.

Backward Stability and Accuracy

Stable Algorithm $\rightarrow$ Well-conditioned Problem $\equiv$ Accurate Result.
Stable Algorithm $\rightarrow$ Ill-conditioned Problem $\equiv$ Possibly Inaccurate Result (inaccuracy depends upon how ill-conditioned the problem is).

3.3.8 Conditioning of the Linear System and Eigenvalue Problems

The Condition Number of a Problem

Numerical analysts usually try to associate a number called the **condition number** with a problem. The condition number indicates whether the problem is ill- or well-conditioned. More specifically, the condition number gives a bound for the relative error in the solution when a small perturbation is applied to the input data.

We will now give results on the conditions of the linear system and eigenvalue problems.

Theorem 3.3.1. *General Perturbation Theorem. Let $\triangle A$ and δb, be the perturbations, respectively, of the data A and b, and δx be the error in x. Assume that A* **is nonsingular** *and $\| \triangle A \| < 1/\| A^{-1} \|$. Then,*

$$\frac{\| \delta x \|}{\| x \|} \leq \frac{\| A \| \| A^{-1} \|}{(1 - \| \triangle A \| \| A^{-1} \|)} \left(\frac{\| \triangle A \|}{\| A \|} + \frac{\| \delta b \|}{\| b \|} \right).$$

Interpretation of the theorem: The above theorem says that if the relative perturbations in A and b are small, then the number $\| A \| \| A^{-1} \|$ is the dominating factor in determining how large the relative error in the solution can be.

Definition 3.3.5. *The number $\| A \| \| A^{-1} \|$ is called the* **condition number** *of the linear system problem $Ax = b$ or just the condition number of A, and is denoted by* Cond(A).

From the theorem above, it follows that **if Cond(A) is large, then the system $Ax = b$ is ill-conditioned**; otherwise it is **well-conditioned**.

The condition number of a matrix certainly depends upon the norm of the matrix. However, roughly, if a matrix is ill-conditioned in one type of norm, it is ill-conditioned in other types as well. This is because the condition numbers in different norms are related. For example, for an $n \times n$ real matrix A, one can

show that

$$\frac{1}{n}\text{Cond}_2(A) \leq \text{Cond}_1(A) \leq n\text{Cond}_2(A),$$
$$\frac{1}{n}\text{Cond}_\infty(A) \leq \text{Cond}_2(A) \leq n\text{Cond}_\infty(A),$$
$$\frac{1}{n^2}\text{Cond}_1(A) \leq \text{Cond}_\infty(A) \leq n^2\text{Cond}_1(A),$$

where $\text{Cond}_p(A)$, $p = 1, 2, \infty$ denotes the condition number in p-norm.

Next, we present the proof of the above theorem in the case $\triangle A = 0$. For the proof in the general case, see Datta (1995, pp. 249–250). We first restate the theorem in this special case.

Theorem 3.3.2. *Right Perturbation Theorem. If δb and δx, are, respectively, the perturbations of b and x in the linear system $Ax = b$, and, A is assumed to be nonsingular and $b \neq 0$, then*

$$\frac{\|\delta x\|}{\|x\|} \leq \text{Cond}(A)\,\frac{\|\delta b\|}{\|b\|}.$$

Proof. We have

$$Ax = b \quad \text{and} \quad A(x + \delta x) = b + \delta b.$$

The last equation can be written as $Ax + A\delta x = b + \delta b$, or

$$A\delta x = \delta b \quad (\text{since } Ax = b) \quad \text{that is, } \delta x = A^{-1}\delta b.$$

Taking a subordinate matrix-vector norm, we get

$$\|\delta x\| \leq \|A^{-1}\|\|\delta b\|. \tag{3.3.1}$$

Again, taking the same norm on both sides of $Ax = b$, we get $\|Ax\| = \|b\|$ or

$$\|b\| = \|Ax\| \leq \|A\|\|x\|. \tag{3.3.2}$$

Combining (3.3.1) and (3.3.2), we have

$$\frac{\|\delta x\|}{\|x\|} \leq \|A\|\|A^{-1}\|\frac{\|\delta b\|}{\|b\|}. \quad \blacksquare \tag{3.3.3}$$

Interpretation of Theorem 3.3.2: Theorem 3.3.2 says that a relative error in the solution can be as large as $\text{Cond}(A)$ multiplied by the relative perturbation in the vector b. Thus, if the condition number is not too large, then a small perturbation in the vector b will have very little effect on the solution. **On the other hand, if the condition number is large, then even a small perturbation in b might change the solution drastically.**

Example 3.3.4. (*An Ill-Conditioned Linear System Problem*)

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4.0001 & 2.002 \\ 1 & 2.002 & 2.004 \end{pmatrix}, \quad b = \begin{pmatrix} 4 \\ 8.0021 \\ 5.006 \end{pmatrix}.$$

The exact solution $x = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$. Change b to $b' = \begin{pmatrix} 4 \\ 8.0020 \\ 5.0061 \end{pmatrix}$.

Then the relative perturbation in b:

$$\frac{\|b' - b\|}{\|b\|} = \frac{\|\delta b\|}{\|b\|} = 1.379 \times 10^{-5} \text{ (small)}.$$

If we solve the system $Ax' = b'$, we get

$$x' = x + \delta x = \begin{pmatrix} 3.0850 \\ -0.0436 \\ 1.0022 \end{pmatrix}.$$

(x' **is completely different from** x).

Note that the relative error in x: $\frac{\|\delta x\|}{\|x\|} = 1.3461$ (quite large!).

It is easily verified that the inequality in the above theorem is satisfied:

$$\text{Cond}(A) \cdot \frac{\|\delta b\|}{\|b\|} = 4.4434, \quad \text{Cond}(A) = 3.221 \times 10^5.$$

However, the predicted change is overly estimated.

Conditioning of Eigenvalues

Like the linear system problem, the eigenvalues and the eigenvectors of a matrix A can be ill-conditioned too.

The following result gives an overall sensitivity of the eigenvalues due to perturbations in the entries of A. For a proof, see Datta (1995) or Golub and Van Loan (1996).

Theorem 3.3.3. *Bauer-Fike. Let* $X^{-1}AX = D = \text{diag}(\lambda_1, \ldots, \lambda_n)$. *Then for any eigenvalue* λ *of* $A + E \in \mathbb{C}^{n \times n}$, *we have*

$$\min |\lambda_i - \lambda| \leq \text{Cond}_p(X) \parallel E \parallel,$$

where $\| \cdot \|_p$ *is a p-norm.*

The result says that the eigenvalues of A might be sensitive to small perturbations of the entries of A if the transforming matrix X is ill-conditioned.

Analysis of the conditioning of the individual eigenvalues and eigenvectors are rather involved. We just state here the conditioning of simple eigenvalues of a matrix.

Let λ_i be a **simple** eigenvalue of A. Then the condition number of λ_i, denoted by $\text{Cond}(\lambda_i)$, is defined to be: $\text{Cond}(\lambda_i) = 1/|y_i^T x_i|$, where y_i and x_i are, respectively, the unit **left** and **right** eigenvectors associated with λ_i.

A well-known example of eigenvalue sensitivity is the **Wilkinson bidiagonal matrix**:

$$A = \begin{pmatrix} 20 & & 20 & & \\ & 19 & & 20 & 0 \\ 0 & & \ddots & & \ddots \\ & & & \ddots & 20 \\ & & & & 1 \end{pmatrix}.$$

The eigenvalues of A are $1, 2, \ldots, 20$.

A small perturbation E of the $(20, 1)$th entry of A (say $E = 10^{-10}$) changes some of the eigenvalues drastically: **they even become complex** (see Datta (1995, pp. 84–85)).

The above matrix A is named after the famous British numerical analyst **James H. Wilkinson**, who computed the condition numbers of the above eigenvalues and found that some of the condition numbers were quite large, explaining the fact why they changed so much due to a small perturbation of just one entry of A.

Note: Though the eigenvalues of a nonsymmetric matrix can be ill-conditioned-the **eigenvalues of a symmetric matrix are well-conditioned (see Datta (1995, pp. 455–456)).**

3.4 LU FACTORIZATION

In this section, we describe a well-known matrix factorization, called the **LU factorization of a matrix** and in the next section, we will show how the LU factorization is used to solve an algebraic linear system.

3.4.1 LU Factorization using Gaussian Elimination

An $n \times n$ matrix A having nonsingular principal minors can be factored into LU: $A = LU$, where L is a lower triangular matrix with 1s along the diagonal (unit lower triangular) and U is an $n \times n$ upper triangular matrix. This factorization is known as an **LU factorization of A**. A classical elimination technique, called **Gaussian elimination**, is used to achieve this factorization.

If an LU factorization exists and A is nonsingular, then the LU factorization is unique (see Golub and Van Loan (1996), pp. 97–98).

Gaussian Elimination

There are $(n-1)$ steps in the process. Beginning with $A^{(0)} = A$, the matrices $A^{(1)}, \ldots, A^{(n-1)}$ are constructed such that $A^{(k)}$ has zeros below the diagonal in the kth column. The final matrix $A^{(n-1)}$ will then be an upper triangular matrix U. Denote $A^{(k)} = (a_{ij}^{(k)})$. The matrix $A^{(k)}$ is obtained from the previous matrix $A^{(k-1)}$ by multiplying the entries of the row k of $A^{(k-1)}$ with $m_{ik} = -(a_{ik}^{(k-1)})/(a_{kk}^{(k-1)})$, $i = k+1, \ldots, n$ and adding them to those of $(k+1)$ through n. In other words,

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} + m_{ik} a_{kj}^{(k-1)}, \quad i = k+1, \ldots, n; \quad j = k+1, \ldots, n. \tag{3.4.1}$$

The entries m_{ik} are called **multipliers**. The entries $a_{kk}^{(k-1)}$ are called the **pivots**.

To see how an LU factorization, when it exists, can be obtained, we note (which is easy to see using the above relations) that

$$A^{(k)} = M_k A^{(k-1)}, \tag{3.4.2}$$

where M_k is a unit lower triangular matrix formed out of the multipliers. The matrix M_k is known as the **elementary lower triangular matrix**. The matrix M_k can be written as:

$$M_k = I + m_k e_k^{\mathrm{T}},$$

where e_k is the kth unit vector, $e_i^{\mathrm{T}} m_k = 0$ for $i \leq k$, and $m_k = (0, \ldots, 0, m_{k+1,k}, \ldots, m_{n,k})^{\mathrm{T}}$.

Furthermore, $M_k^{-1} = I - m_k e_k^{\mathrm{T}}$.

Using (3.4.2), we see that

$$\begin{aligned} U = A^{(n-1)} &= M_{n-1} A^{(n-2)} = M_{n-1} M_{n-2} A^{(n-3)} \\ &= \cdots = M_{n-1} M_{n-2} \cdots M_2 M_1 A \end{aligned}$$

Thus, $A = (M_{n-1} M_{n-2} \cdots M_2 M_1)^{-1} U = LU$,

where $L = (M_{n-1} M_{n-2} \cdots M_2 M_1)^{-1}$.

Since each of the matrices M_1 through M_{n-1} is a unit upper triangular matrix, so is L (*Note:* The product of two unit upper triangular matrix is an upper triangular matrix and the inverse of a unit upper triangular matrix is an upper triangular matrix).

Constructing L: The matrix L can be formed just from the multipliers, as shown below. **No explicit matrix inversion is needed.**

$$L = \begin{pmatrix} 1 & 0 & 0 & \cdots & \cdots & 0 \\ -m_{21} & 1 & 0 & \cdots & \cdots & 0 \\ -m_{31} & -m_{32} & 1 & \cdots & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \ddots & 0 \\ -m_{n1} & -m_{n2} & -m_{n3} & \cdots & -m_{n,n-1} & 1 \end{pmatrix}.$$

Difficulties with Gaussian Elimination without Pivoting

Gaussian elimination, as described above, fails if any of the pivots is zero, **it is worse yet if any pivot becomes close to zero**. In this case, the method can be carried to completion, but the obtained results may be totally wrong.

Consider the following simple example: Let Gaussian elimination without pivoting be applied to

$$A = \begin{pmatrix} 0.0001 & 1 \\ 1 & 1 \end{pmatrix},$$

using three decimal digit floating point arithmetic.

There is only one step. The multiplier $m_{21} = -1/10^{-4} = -10^4$. Let $\hat{L}$ and $\hat{U}$ be the computed versions of L and U. Then,

$$\hat{U} = A^{(1)} = \begin{pmatrix} 0.0001 & 1 \\ 0 & 1 - 10^4 \end{pmatrix} = \begin{pmatrix} 0.0001 & 1 \\ 0 & -10^4 \end{pmatrix}.$$

(Note that $(1 - 10^4)$ gives -10^4 in three-digit arithmetic). The matrix $\hat{L}$ formed out the multiplier m_{21} is

$$\hat{L} = \begin{pmatrix} 1 & 0 \\ 10^4 & 1 \end{pmatrix}.$$

The product of the computed $\hat{L}$ and $\hat{U}$ is:

$$\hat{L}\hat{U} = \begin{pmatrix} 0.0001 & 1 \\ 1 & 0 \end{pmatrix},$$

which is different from A.

Note that the pivot $a_{11}^{(1)} = 0.0001$ is very close to zero (in three-digit arithmetic). This small pivot gave a large multiplier. This large multiplier, when used to update the entries of A, the number 1, which is much smaller compared to 10^4, got wiped out in the subtraction of $1 - 10^4$ and the result was -10^4.

Gaussian Elimination with Partial Pivoting

The above example suggests that disaster in Gaussian elimination without pivoting in the presence of a small pivot can perhaps be avoided by identifying a **"good pivot"** (a pivot as large as possible) at each step, before the process of elimination is applied. The good pivot may be located among the entries in a column or among all the entries in a submatrix of the current matrix. In the former case, since the search is only partial, the method is called **partial pivoting**; in the latter case, the method is called **complete pivoting**. *It is important to note that the purpose of pivoting is to prevent large growth in the reduced matrices, which can wipe out the original data.* One way to do this is to keep multipliers less than 1 in magnitude, and this is exactly what is accomplished by pivoting.

We will discuss here only Gaussian elimination with partial pivoting, which also consists of $(n-1)$ steps.

In fact, the process is just a slight modification of Gaussian elimination in the following sense: At each step, the largest entry (in magnitude) is identified among all the entries in the pivot column. This entry is then brought to the diagonal position of the current matrix by interchange of suitable rows and then, using that entry as "pivot," the elimination process is performed.

Thus, if we set $A^{(0)} = A$, at step k $(k = 1, 2, \ldots, n-1)$, first, the largest entry (in magnitude) $a_{r_k,k}^{(k-1)}$ is identified among all the entries of the column k (below the row $(k-1)$) of the matrix $A^{(k-1)}$, this entry is then brought to the diagonal position by interchanging the rows k and r_k, and then the elimination process proceeds with $a_{r_k,k}^{(k-1)}$ as the pivot.

LU Factorization from Gaussian Elimination with Partial Pivoting

Since the interchange of two rows of a matrix is equivalent to premultiplying the matrix by a permutation matrix, the matrix $A^{(k)}$ is related to $A^{(k-1)}$ by the following relation:

$$A^{(k)} = M_k P_k A^{(k-1)}, \quad k = 1, 2, \ldots, n-1,$$

where P_k is the permutation matrix obtained by interchanging the rows k and r_k of the identity matrix, and M_k is an elementary lower triangular matrix resulting from the elimination process. So,

$$\begin{aligned} U = A^{(n-1)} &= M_{n-1}P_{n-1}A^{(n-2)} = M_{n-1}P_{n-1}M_{n-2}P_{n-2}A^{(n-3)} \\ &= \cdots = M_{n-1}P_{n-1}M_{n-2}P_{n-2}\cdots M_2P_2M_1P_1A. \end{aligned}$$

Setting $M = M_{n-1}P_{n-1}M_{n-2}P_{n-2}\cdots M_2P_2M_1P_1$, we have the following factorization of A:

$$U = MA.$$

The above factorization can be written in the form: $PA = LU$, where $P = P_{n-1}P_{n-2}\cdots P_2P_1$, $U = A^{(n-1)}$, and the matrix L is a unit lower triangular matrix formed out of the multipliers. For details, see Golub and Van Loan (1996, pp. 99).

For $n = 4$, the reduction of A to the upper triangular matrix U can be schematically described as follows:

$$1.\quad A \xrightarrow{P_1} P_1A \xrightarrow{M_1} M_1P_1A = \begin{pmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{pmatrix} = A^{(1)}.$$

2. $A^{(1)} \xrightarrow{P_2} P_2A^{(1)} \xrightarrow{M_2} M_2P_2A^{(1)} = M_2P_2M_1P_1A = \begin{pmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{pmatrix} = A^{(2)}.$

3. $A^{(2)} \xrightarrow{P_3} P_3A^{(2)} \xrightarrow{M_3} M_3P_3A^{(2)} = M_3P_3M_2P_2M_1P_1A = \begin{pmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{pmatrix}$

$= A^{(3)} = U.$

The only difference between L here and the matrix L from Gaussian elimination without pivoting is that the multipliers in the kth column are now permuted according to the permutation matrix $\tilde{P}_k = P_{n-1}P_{n-2}\cdots P_{k+1}$.

Thus, to construct L, again no explicit products or matrix inversions are needed. We illustrate this below.

Consider the case $n = 4$, and suppose P_2 interchanges rows 2 and 3, and P_3 interchanges rows 3 and 4.

The matrix L is then given by:

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -m_{31} & 1 & 0 & 0 \\ -m_{21} & -m_{42} & 1 & 0 \\ -m_{41} & -m_{32} & -m_{34} & 1 \end{pmatrix}.$$

Example 3.4.1.

$$A = \begin{pmatrix} 1 & 2 & 4 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}.$$

$k = 1$

1. The pivot entry is 7: $r_1 = 3$.
2. Interchange rows 3 and 1.

$$P_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \qquad P_1A = \begin{pmatrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 4 \end{pmatrix}.$$

3. Form the multipliers: $a_{21} \equiv m_{21} = -\frac{4}{7}$, $a_{31} \equiv m_{31} = -\frac{1}{7}$.

4. $A^{(1)} = M_1P_1A = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{4}{7} & 1 & 0 \\ -\frac{1}{7} & 0 & 1 \end{pmatrix} \begin{pmatrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 4 \end{pmatrix} \equiv \begin{pmatrix} 7 & 8 & 9 \\ 0 & \frac{3}{7} & \frac{6}{7} \\ 0 & \frac{6}{7} & \frac{19}{7} \end{pmatrix}.$

$k = 2$

1. The pivot entry is $\frac{6}{7}$, $r_2 = 3$.

2. Interchange rows 2 and 3.

$$P_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \qquad P_2 A^{(1)} = \begin{pmatrix} 7 & 8 & 9 \\ 0 & \frac{6}{7} & \frac{19}{7} \\ 0 & \frac{3}{7} & \frac{6}{7} \end{pmatrix}.$$

3. Form the multiplier: $m_{32} = -\frac{1}{2}$

$$A^{(2)} = M_2 P_2 A^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} 7 & 8 & 9 \\ 0 & \frac{6}{7} & \frac{19}{7} \\ 0 & \frac{3}{7} & \frac{6}{7} \end{pmatrix} = \begin{pmatrix} 7 & 8 & 9 \\ 0 & \frac{6}{7} & \frac{19}{7} \\ 0 & 0 & -\frac{1}{2} \end{pmatrix}.$$

$$\text{Form } L = \begin{pmatrix} 1 & 0 & 0 \\ -m_{31} & 1 & 0 \\ -m_{21} & -m_{32} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{7} & 1 & 0 \\ \frac{4}{7} & \frac{1}{2} & 1 \end{pmatrix}.$$

$$P = P_2 P_1 = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}.$$

Verify. $PA = \begin{pmatrix} 7 & 8 & 9 \\ 1 & 2 & 4 \\ 4 & 5 & 6 \end{pmatrix} = LU.$

Flop-count. Gaussian elimination with partial pivoting requires only $\frac{2}{3}n^3$ flops. Furthermore, the process with partial pivoting requires at most $O(n^2)$ comparisons for identifying the pivots.

Stability of Gaussian Elimination

The stability of Gaussian elimination algorithms is better understood by measuring the growth of the elements in the reduced matrices $A^{(k)}$. **(Note that although pivoting keeps the multipliers bounded by unity, the elements in the reduced matrices still can grow arbitrarily.)**

Definition 3.4.1. *The* **growth factor** ρ *is the ratio of the largest element (in magnitude) of* $A, A^{(1)}, \ldots, A^{(n-1)}$ *to the largest element (in magnitude) of* A*:* $\rho = (\max(\alpha, \alpha_1, \alpha_2, \ldots, \alpha_{n-1}))/\alpha$*, where* $\alpha = \max_{i,j} |a_{ij}|$*, and* $\alpha_k = \max_{i,j} |a_{ij}^{(k)}|$.

The growth factor ρ **can be arbitrarily large for Gaussian elimination without pivoting**. Note that ρ for the matrix

$$A = \begin{pmatrix} 0.0001 & 1 \\ 1 & 1 \end{pmatrix}$$

without pivoting is 10^4.

Thus, Gaussian elimination without pivoting is, in general, unstable.

Note: Though Gaussian elimination without pivoting is unstable for arbitrary matrices, there are two classes of matrices, the **diagonally dominant matrices** and the **symmetric positive definite matrices**, for which the process can be shown to be stable. The growth factor of a diagonally dominant matrix is bounded by 2 and that of a symmetric positive definite matrix is 1.

The next question is: How large can the growth factor be for Gaussian elimination with partial pivoting?

The growth factor ρ for Gaussian elimination with partial pivoting can be as large as 2^{n-1}: $\rho \leq 2^{n-1}$.

Though matrices for which this bound is attained can be constructed (see Datta 1995), such matrices are rare in practice. Indeed, in many practical examples, the elements of the matrices $A^{(k)}$ very often continue to decrease in size. **Thus, Gaussian elimination with partial pivoting is not unconditionally stable in theory; in practice, however, it can be considered as a stable algorithm.**

MATLAB note: The MATLAB command $[L, U, P] = \text{lu}\ (A)$ returns lower triangular matrix L, upper triangular matrix U, and permutation matrix P such that $PA = LU$.

3.4.2 The Cholesky Factorization

Every symmetric positive definite matrix A can be factored into

$$A = HH^{\mathrm{T}},$$

where H is a lower triangular matrix with positive diagonal entries.

This factorization of A is known as the **Cholesky factorization**. Since, the growth factor for Gaussian elimination of a symmetric positive definite matrix is 1, **Gaussian elimination can be safely used to compute the Cholesky factorization of a symmetric positive definite matrix**. Unfortunately, no advantage of symmetry of the matrix A can be taken in the process.

In practice, the entries of the lower triangular matrix H, called the **Cholesky factor**, are computed directly from the relation $A = HH^{\mathrm{T}}$. The matrix H is computed row by row. The algorithm is known as the Cholesky algorithm. See Datta (1995, pp. 222–223) for details.

Algorithm 3.4.1. *The Cholesky Algorithm*
Input. *A—A symmetric positive definite matrix*
Output. *H—The Cholesky factor*
For $k = 1, 2, \ldots, n$ *do*
For $i = 1, 2, \ldots, k-1$ *do*

$$h_{ki} = \frac{1}{h_{ii}} \left(a_{ki} - \sum_{j=1}^{i-1} h_{ij} h_{kj} \right)$$

$$h_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} h_{kj}^2}$$

End
End

Flop-count and numerical stability. Algorithm 3.4.1 requires only $n^3/3$ flops. The algorithm is **numerically stable**.

MATLAB and MATCOM notes: Algorithm 3.4.1 has been implemented in MATCOM program **choles**. MATLAB function **chol** also can be used to compute the Cholesky factor. However, note that $L = \mathbf{chol}(A)$ computes an upper triangular matrix R such that $A = R^{\mathrm{T}}R$.

3.4.3 LU Factorization of an Upper Hessenberg Matrix

Recall that $H = (h_{ij})$ is an upper Hessenberg matrix if $h_{ij} = 0$ whenever $i > j+1$. Thus, Gaussian elimination scheme applied to an $n \times n$ upper Hessenberg matrix requires zeroing of only the nonzero entries on the subdiagonal. This means at each step, after a possible interchange of rows, just a multiple of the row containing the pivot has to be added to the next row.

Specifically, Gaussian elimination scheme with partial pivoting for an $n \times n$ upper Hessenberg matrix $H = (h_{ij})$ is as follows:

Algorithm 3.4.2. *LU Factorization of an Upper Hessenberg Matrix*
Input. *H—An $n \times n$ upper Hessenberg matrix*
Output. *U—The upper triangular matrix U of LU factorization of H, stored over the upper part of H. The subdiagonal entries of H contain the multipliers.*
For $k = 1, 2, \ldots, n-1$ do

1. *Interchange $h_{k,j}$ and $h_{k+1,j}$, if $|h_{k,k}| < |h_{k+1,k}|$, $j = k, \ldots, n$.*
2. *Compute the multiplier and store it over $h_{k+1,k}$: $h_{k+1,k} \equiv -\dfrac{h_{k+1,k}}{h_{k,k}}$.*
3. *Update $h_{k+1,j}$: $h_{k+1,j} \equiv h_{k+1,j} + h_{k+1,k} \cdot h_{k,j}$, $j = k+1, \ldots, n$.*

End.

Flop-count and stability. The above algorithm requires n^2 flops.

It can be shown Wilkinson (1965, p. 218); Higham (1996, p. 182), that the growth factor ρ of a Hessenberg matrix for Gaussian elimination with partial pivoting is less than or equal to n. **Thus, computing LU factorization of a Hessenberg**

matrix using Gaussian elimination with partial pivoting is an efficient and a numerically stable procedure.

3.5 NUMERICAL SOLUTION OF THE LINEAR SYSTEM $Ax=b$

Given an $n \times n$ matrix A and the n-vector b, the algebraic linear system problem is the problem of finding an n-vector x such that $Ax = b$.

The principal uses of the LU factorization of a matrix A are: **solving the algebraic linear system** $Ax = b$, **finding the determinant of a matrix**, and **finding the inverse of** A.

We will discuss first how $Ax = b$ can be solved using the LU factorization of A.

The following theorem gives results on the existence and uniqueness of the solution x of $Ax = b$. Proof can be found in any linear algebra text.

Theorem 3.5.1. *Existence and Uniqueness Theorem. The system $Ax = b$ has a solution if and only if rank (A) = rank(A, b). The solution is unique if and only if A is invertible.*

3.5.1 Solving $Ax = b$ using the Inverse of A

The above theorem suggests that the unique solution x of $Ax = b$ be computed as $x = A^{-1}b$.

Unfortunately, **computationally this is not a practical idea. It generally involves more computations and gives less accurate answers.**

This can be illustrated by the following trivial example:

Consider solving $3x = 27$.

The exact answer is: $x = 27/3 = 9$. Only one flop (one division) is needed in this process. On the other hand, if the problem is solved by writing it in terms of the inverse of A, we then have $x = \frac{1}{3} \times 27 = 0.3333 \times 27 = 8.9991$ (in four digit arithmetic), a less accurate answer. Moreover, the process will need two flops: one division and one multiplication.

3.5.2 Solving $Ax = b$ using Gaussian Elimination with Partial Pivoting

Since Gaussian elimination without pivoting does not always work and, even when it works, might give an unacceptable answer in certain instances, we only discuss solving $Ax = b$ using Gaussian elimination with partial pivoting.

We have just seen that Gaussian elimination with partial pivoting, when used to triangularize A, yields a factorization $PA = LU$. In this case, the system $Ax = b$ is equivalent to the two triangular systems:

$$Ly = Pb = b' \quad \text{and} \quad Ux = y.$$

Thus, to solve $Ax = b$ using Gaussian elimination with partial pivoting, the following two steps need to be performed in the sequence.

Step 1. Find the factorization $PA = LU$ using Gaussian eliminating with partial pivoting.

Step 2. Solve the lower triangular system: $Ly = Pb = b'$ first, followed by the upper triangular system: $Ux = y$.

Forming the vector b'. The vector b' is just the permuted version of b. So, to obtain b', all that needs to be done is to permute the entries of b in the same way as the rows of the matrices $A^{(k)}$ have been interchanged. This is illustrated in the following example.

Example 3.5.1. *Solve the following system using Gaussian elimination with partial pivoting*:

$$\begin{aligned} x_1 + 2x_2 + 4x_3 &= 7, \\ 4x_1 + 5x_2 + 6x_3 &= 15, \\ 7x_1 + 8x_2 + 9x_3 &= 24. \end{aligned}$$

Here

$$A = \begin{pmatrix} 1 & 2 & 4 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \qquad b = \begin{pmatrix} 7 \\ 15 \\ 24 \end{pmatrix}.$$

Using the results of Example 3.4.1, we have

$$L = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{7} & 1 & 0 \\ \frac{4}{7} & \frac{1}{2} & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} 7 & 8 & 9 \\ 0 & \frac{6}{7} & \frac{19}{7} \\ 0 & \frac{6}{7} & \frac{19}{7} \\ 0 & 0 & -\frac{1}{2} \end{pmatrix}.$$

Since $r_1 = 3$, and $r_2 = 3$,

$$b' = \begin{pmatrix} 24 \\ 7 \\ 15 \end{pmatrix}.$$

Note that to obtain b', first the 1st and 3rd components of b were permuted, according to $r_1 = 3$ (which means the interchange of rows 1 and 3), followed by the

permutation of the components 2 and 3, according to $r_2 = 3$ (which means the interchange of the rows 2 and 3). $Ly = b'$ gives

$$y = \begin{pmatrix} 24 \\ 3.5714 \\ -0.5000 \end{pmatrix},$$

and $Ux = y$ gives

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

Flop-count. The factorization process requires about $\frac{2}{3}n^3$ flops. The solution of each of the triangular systems $Ly = b'$ and $Ux = y$ requires n^2 flops. Thus, the solution of the linear system $Ax = b$ using Gaussian elimination with partial pivoting requires about $\frac{2}{3}n^3 + O(n^2)$ flops. Also, the process requires $O(n^2)$ comparisons for pivot identifications.

Stability of Gaussian Elimination Scheme for $Ax = b$

We have seen that the growth factor ρ determines the stability of the triangularization procedure. Since solutions of triangular systems are numerically stable procedures, the growth factor is still the dominating factor for solving linear systems with Gaussian elimination.

The large growth factor ρ for Gaussian elimination with partial pivoting is rare in practice. **Thus, for all practical purposes, Gaussian elimination with partial pivoting for the linear system $Ax = b$ is a numerically stable procedure.**

3.5.3 Solving a Hessenberg Linear System

Certain control computations such as **computing the frequency response of a matrix** (see Chapter 5) require solution of a Hessenberg linear algebraic system. We have just seen that the LU factorization of a Hessenberg matrix requires only $O(n^2)$ flops and Gaussian elimination with partial pivoting is safe, because, the growth factor in this case is at most n. Thus, **a Hessenberg system can be solved using Gaussian elimination with partial pivoting using $O(n^2)$ flops and in a numerically stable way.**

3.5.4 Solving $AX = B$

In many practical situations, one faces the problem of solving multiple linear systems: $AX = B$. Here A is $n \times n$ and nonsingular and B is $n \times p$. Since each

of the systems here has the same coefficient matrix A, to solve $AX = B$, we need to factor A just once. The following scheme, then, can be used.

Partition $B = (b_1, \ldots, b_p)$.

Step 1. Factorize A using Gaussian elimination with partial pivoting: $PA = LU$

Step 2. For $k = 1, \ldots, p$ do

Solve $Ly = Pb_k$

Solve $Ux_k = y$

End

Step 3. Form $X = (x_1, \ldots, x_p)$.

3.5.5 Finding the Inverse of A

The inverse of an $n \times n$ nonsingular matrix A can be obtained as a special case of the above method. Just set $B = I_{n\times n}$. Then, $X = A^{-1}$.

3.5.6 Computing the Determinant of A

The determinant of matrix A can be immediately computed, once the LU factorization of A is available. Thus, if Gaussian elimination with partial pivoting is used giving $PA = LU$, then $\det(A) = (-1)^r \prod_{i=1}^{n} u_{ii}$, where r is the number of row interchanges in the partial pivoting process.

3.5.7 Iterative Refinement

Once the system $Ax = b$ is solved using Gaussian elimination, it is suggested that the computed solution be refined iteratively to a desired accuracy using the following procedure. The procedure is fairly inexpensive and requires only $O(n^2)$ flops for each iteration.

Let x be the computed solution of $Ax = b$ obtained by using Gaussian elimination with partial pivoting factorization: $PA = LU$.

For $k = 1, 2, \ldots,$ do until desired accuracy.

1. Compute the residual $r = b - Ax$ (in double precision).
2. Solve $Ly = Pr$ for y.
3. Solve $Uz = y$ for z.
4. Update the solution $x \equiv x + z$.

3.6 THE QR FACTORIZATION

Recall that a square matrix O is said to be an **orthogonal matrix** if $OO^{\mathrm{T}} = O^{\mathrm{T}}O = I$. Given an $m \times n$ matrix A there exist an $m \times m$ orthogonal matrix Q

and an $m \times n$ upper triangular matrix R such that $A = QR$. Such a factorization of A is called the **QR factorization**. If $m \geq n$, and if the matrix Q is partitioned as $Q = [Q_1, Q_2]$, where Q_1 is the matrix of the first n columns of Q, and if R_1 is defined by

$$R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix},$$

where R_1 is $n \times n$ upper triangular, then $A = Q_1 R_1$. This QR factorization is called the **"economy size"** or the **"thin"** QR factorization of A. The following theorem gives condition for uniqueness of the "thin" QR factorization. For a proof of the theorem, see Golub and Van Loan (1996, p. 230).

Theorem 3.6.1. *Let $A \in \mathbb{R}^{m \times n}, m \geq n$ have full rank. Then the thin QR factorization*

$$A = Q_1 R_1$$

is unique. Furthermore, the diagonal entries of R_1 are all positive.

There are several ways to compute the QR factorization of a matrix. Householder's and Givens' methods can be used to compute both types of QR factorizations. On the other hand, the classical Gram–Schmidt (CGS) and the modified Gram–Schmidt (MGS) compute $Q \in \mathbb{R}^{m \times n}$ and $R \in \mathbb{R}^{n \times n}$ such that $A = QR$.

The MGS has better numerical properties than the CGS. We will not discuss them here. The readers are referred to the book Datta (1995, pp. 339–343). We will discuss Householder's and Givens' methods in the sequel.

3.6.1 Householder Matrices

Definition 3.6.1. *A matrix of the form $H = I - 2uu^{\mathrm{T}}/u^{\mathrm{T}}u$, where u is a nonzero vector, is called a* **Householder matrix**, *after the celebrated American numerical analyst Alston Householder.*

A Householder matrix is also known as an **Elementary Reflector** or a **Householder transformation**.

It is easy to see that a Householder matrix H is **symmetric** and **orthogonal**.

A Householder matrix H is an important tool to create zeros in a vector:

Given $x = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$, the Householder matrix $H = I - 2(uu^{\mathrm{T}}/u^{\mathrm{T}}u)$, where $u = x + \mathrm{sgn}(x_1) \| x \|_2 e_1$ is such that $Hx = (\sigma, 0, \ldots, 0)^{\mathrm{T}}$, where $\sigma = -\mathrm{sgn}(x_1) \| x \|_2$.

Schematically, $x \xrightarrow{H} Hx = (\sigma, 0, \ldots, 0)^{\mathrm{T}}$.

Forming Matrix–Vector and Matrix–Matrix Products With a Householder Matrix

A remarkable computational advantage involving Householder matrices is that neither a matrix–vector product with a Householder matrix H nor the matrix product HA (or AH) needs to be explicitly formed, as can be seen from the followings:

1. $Hx = \left(I - 2\dfrac{uu^{\mathrm{T}}}{u^{\mathrm{T}}u}\right)x = x - \beta u(u^{\mathrm{T}}x)$, where $\beta = \dfrac{2}{u^{\mathrm{T}}u}$.
2. $HA = (I - \beta uu^{\mathrm{T}})A = A - \beta uu^{\mathrm{T}}A = A - \beta uv^{\mathrm{T}}$, where $v = A^{\mathrm{T}}u$.
3. $AH = A(I - \beta uu^{\mathrm{T}}) = A - \beta vu^{\mathrm{T}}$, where $v = Au$.

From above, we immediately see that the matrix product HA or AH requires only $O(n^2)$ flops, a substantial saving compared to $2n^3$ flops that are required to compute the product of two arbitrary matrices.

3.6.2 The Householder QR Factorization

Householder's method for the QR factorization of matrix $A \in \mathbb{R}^{m\times n}$ with $m \geq n$, consists of constructing Householder matrices $H_1, H_2, \ldots, H_n$ successively such that

$$H_nH_2 \cdots H_1A = R$$

is an $m \times n$ upper triangular matrix. If $H_1H_2 \cdots H_n = Q$, then Q is an orthogonal matrix (since each H_i is orthogonal) and from above, we have $Q^{\mathrm{T}}A = R$ or $A = QR$. Note that

$$R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix},$$

where $R_1 \in \mathbb{R}^{n\times n}$ and is upper triangular. The matrices H_i are constructed such that $A^{(i)} = H_iA^{(i-1)}$ (with $A^{(0)} = A$) has zeros below the diagonal in the ith column (see **Example 3.6.1**).

Flop-count. The Householder QR factorization method requires approximately $2n^2(m - (n/3))$ flops just to compute the triangular matrix R.

Note: The matrix Q can be computed, if required, as $Q = H_1 \cdots H_n$ by forming the product implicitly, as shown in Section 3.6.1.

It should be noted that in a majority of practical applications, it is sufficient to have Q in this factored form; in many applications, Q is not needed at all. If Q is needed explicitly, about another $4(m^2n - mn^2 + (n^3/3))$ flops will be required.

Numerical stability: The Householder QR factorization method computes the QR factorization of a slightly perturbed matrix. Specifically, it can be shown Wilkinson (1965, p. 236) that, if $\hat{R}$ denotes the computed R, then there exists

an orthogonal $\hat{Q}$ such that

$$A + E = \hat{Q}\hat{R}, \quad \text{where } \| E \|_2 \approx \mu \| A \|_2 .$$

The algorithm is thus stable.

MATLAB notes: $[Q, R] = \boldsymbol{qr}(A)$ computes the QR factorization of A, using Householder's method.

Example 3.6.1.

$$A = \begin{pmatrix} 1 & 1 \\ 0.0001 & 0 \\ 0 & 0.0001 \end{pmatrix}$$

$k = 1$

Form H_1:

$$u_1 = \begin{pmatrix} 1 \\ 0.0001 \\ 0 \end{pmatrix} + \sqrt{1 + (0.0001)^2} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 0.0001 \\ 0 \end{pmatrix}.$$

$$\text{Update } A \equiv A^{(1)} = H_1 A = \left(I - \frac{2u_1 u_1^{\mathrm{T}}}{u_1^{\mathrm{T}} u_1} \right) A = \begin{pmatrix} -1 & -1 \\ 0 & -0.0001 \\ 0 & 0.0001 \end{pmatrix}.$$

$k = 2$

Form H_2:

$$u_2 = \begin{pmatrix} -0.0001 \\ 0.0001 \end{pmatrix} - \sqrt{(-0.0001)^2 + (0.0001)^2} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 10^{-4} \begin{pmatrix} -2.4141 \\ 0.1000 \end{pmatrix}.$$

$$\hat{H}_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - 2\frac{u_2 u_2^{\mathrm{T}}}{u_2^{\mathrm{T}} u_2} = \begin{pmatrix} -0.7071 & 0.7071 \\ 0.7071 & 0.7071 \end{pmatrix},$$

$$H_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -0.7071 & 0.7071 \\ 0 & 0.7071 & 0.7071 \end{pmatrix}.$$

$$\text{Update } A \equiv A^{(2)} = H_2 A^{(1)} = \begin{pmatrix} -1 & -1 \\ 0 & 0.0001 \\ 0 & 0 \end{pmatrix}.$$

Form Q and R:

$$Q = H_1H_2 = \begin{pmatrix} -1 & 0.0001 & -0.0001 \\ -0.0001 & -0.7071 & 0.7071 \\ 0 & 0.7071 & 0.7071 \end{pmatrix}.$$

$$R = \begin{pmatrix} -1 & -1 \\ 0 & 0.0001 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}, \quad \text{where } R_1 = \begin{pmatrix} -1 & -1 \\ 0 & 0.0001 \end{pmatrix}.$$

Complex QR Factorization

If $x \in \mathbb{C}^n$ and $x_1 = re^{i\theta}$, then it is easy to see that the Householder matrix $H = I - \beta vv^*$, where $v = x \pm e^{i\theta} \| x \|_2 e_1$ and $\beta = 2/v^*v$, is such that $Hx = \mp ve^{i\theta} \| x \|_2 e_1$.

Using the above formula, the Householder QR factorization method for a real matrix A, described in the last section, can be easily adapted to a complex matrix. The details are left to the readers.

The process of complex QR factorization of an $m \times n$ matrix, $m \geq n$, using Householder's method requires $8n^2(m - (n/3))$ real flops.

3.6.3 Givens Matrices

Definition 3.6.2. *A matrix of the form*

$$J(i, j, c, s) = \begin{array}{c} \begin{matrix} & & & & i\text{th} & & j\text{th} & & \\ & & & & \downarrow & & \downarrow & & \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \ddots & & & & & \cdots & \vdots \\ \vdots & \vdots & & & & & & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & \vdots & \vdots & & & & & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & \vdots & \vdots & & & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 & \cdots & 1 \end{pmatrix} \end{array} \begin{matrix} \\ \\ \\ \\ \leftarrow i\text{th} \\ \\ \leftarrow j\text{th} \\ \\ \end{matrix}$$

where $c^2 + s^2 = 1$, is called a **Givens matrix**, *after the name of the numerical analyst Wallace Givens.*

Since one can choose $c = \cos\theta$ and $s = \sin\theta$ for some θ, the above Givens matrix can be conveniently denoted by $J(i, j, \theta)$. Geometrically, the matrix $J(i, j, \theta)$ rotates a pair of coordinate axes (ith unit vector as its x-axis and the jth unit vector as its y-axis) through the given angle θ in the (i, j) plane. That is why, the Givens matrix $J(i, j, \theta)$ is commonly known as a **Givens Rotation** or **Plane Rotation in the** (i, j) **plane.**

Thus, when an n-vector $x = (x_1, x_2, \ldots, x_n)^T$ is premultiplied by the Givens rotation $J(i, j, \theta)$, only the ith and jth components of x are affected; the other components remain unchanged.

Also, note that since $c^2 + s^2 = 1$; $J(i, j, \theta) \cdot J(i, j, \theta)^T = I$. **So, the Givens matrix** $J(i, j, \theta)$ **is orthogonal.**

Zeroing the Entries of a 2×2 **Vector Using a Givens Matrix**

If

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

is a vector, then it is a matter of simple verification that, with

$$c = \frac{x_1}{\sqrt{x_1^2 + x_2^2}} \quad \text{and} \quad s = \frac{x_2}{\sqrt{x_1^2 + x_2^2}},$$

the Givens rotation

$$J(1, 2, \theta) = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

is such that

$$J(1, 2, \theta)x = \begin{pmatrix} * \\ 0 \end{pmatrix}.$$

The preceding formula for computing c and s might cause some **underflow** or **overflow**. However, the following simple rearrangement of the formula might prevent that possibility.

1. If $|x_2| \geq |x_1|$, compute $t = x_1/x_2$, $s = 1/\sqrt{1 + t^2}$, and take $c = st$.
2. If $|x_2| < |x_1|$, compute $t = x_2/x_1$, $c = 1/\sqrt{1 + t^2}$, and take $s = ct$.

Implicit Construction of JA

If A is $\mathbb{R}^{m\times n}$ and $J(i, j, c, s) \in \mathbb{R}^{m\times m}$, then the update $A \equiv J(i, j, c, s)A$ can be computed implicitly as follows:

For $k = 1, \ldots, n$ do
$a \equiv a_{ik}$
$b \equiv a_{jk}$
$a_{ik} \equiv ac + bs$
$a_{jk} \equiv -as + bc$
End

MATCOM note: The above computation has been implemented in MATCOM program PGIVMUL.

3.6.4 The QR Factorization Using Givens Rotations

Assume that $A \in \mathbb{R}^{m\times n}$, $m \geq n$. The basic idea is just like Householder's: Compute orthogonal matrices $Q_1, Q_2, \ldots, Q_n$, using Givens rotations such that $A^{(1)} = Q_1 A$ has zeros below the (1, 1) entry in the first column, $A^{(2)} = Q_2 A^{(1)}$ has zeros below the (2, 2) entry in the second column, and so on. Each Q_i is generated as a product of Givens rotations. One way to form $\{Q_i\}$ is:

$$Q_1 = J(1, m, \theta)J(1, m-1, \theta)\cdots J(1, 2, \theta),$$
$$Q_2 = J(2, m, \theta)J(2, m-1, \theta)\cdots J(2, 3, \theta),$$

and so on.

Then,

$$R = A^{(n)} = Q_n A^{(n-1)} = Q_n Q_{n-1} A^{(n-2)} = \cdots$$
$$= Q_n Q_{n-1} \cdots Q_2 Q_1 A = Q^{\mathrm{T}} A, \quad \text{where } Q = Q_1^{\mathrm{T}} Q_2^{\mathrm{T}} \cdots Q_n^{\mathrm{T}}.$$

Algorithm 3.6.1. *Givens QR Factorization*
Input. *A—An m × n matrix*
Outputs. *R—An m × n upper triangular matrix stored over A.*
Q—An m × m orthogonal matrix in factored form defined by the Givens parameters c, s, and the indices k and l.

Step 1. For $k = 1, 2, \ldots, n$ do

For $l = k + 1, \ldots, m$ do

1.1. Find c and s using the formulas given in **Section 3.6.3** so that

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} a_{kk} \\ a_{lk} \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}.$$

1.2. Save the indices k and l and the numbers c and s

1.3. Update A using the implicit construction as shown above:

$A \equiv J(i, j, c, s)A$

End

End

Step 2. Set $R \equiv A$.

Forming the matrix Q. If the orthogonal matrix Q is needed explicitly, then it can be computed from the product $Q = Q_1^{\mathrm{T}} Q_2^{\mathrm{T}} \cdots Q_n^{\mathrm{T}}$, where each Q_i is the product of $m - i$ Givens rotations: $Q_i = J(i, m, \theta)J(i, m-1, \theta) \cdots J(i, i+1, \theta)$.

Flop-count. The algorithm requires $3n^2(m - n/3)$ flops; $m \geq n$. This count, of course, does not include computation of Q.

Numerical stability. **The algorithm is stable.** It can be shown Wilkinson (1965, p. 240) that for $m = n$, the computed $\hat{Q}$ and $\hat{R}$ satisfy $\hat{R} = \hat{Q}^{\mathrm{T}}(A + E)$, where $\| E \|_{\mathrm{F}}$ is small.

MATCOM note: The above algorithm has been implemented in MATCOM program GIVQR.

Q and R have been explicitly computed.

3.6.5 The QR Factorization of a Hessenberg Matrix Using Givens Matrices

From the structure of an upper Hessenberg matrix H, it is easy to see that the ***QR* factorization of *H* takes only $O(n^2)$ flops** either by Householder's or Givens' method, compared to $O(n^3)$ procedure for a full matrix. This is because only one entry from each column has to be made zero. Try this yourself using Algorithm 3.6.1.

3.7 ORTHONORMAL BASES AND ORTHOGONAL PROJECTIONS USING QR FACTORIZATION

The QR factorization of A can be used to compute the orthonormal bases and orthogonal projections associated with the subspaces $R(A)$ and $N(A^{\mathrm{T}})$. Let A be $m \times n$, where $m \geq n$ and have full rank. Suppose $Q^{\mathrm{T}}A = R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$. Partition $Q = (Q_1, Q_2)$, where Q_1 has n columns. **Then the columns of Q_1 form an orthonormal basis for $R(A)$.** Similarly, **the columns of Q_2 form an orthonormal**

basis for the orthogonal complement of $R(A)$. Thus, the matrix $P_A = Q_1 Q_1^T$ is the **orthogonal projection onto** $R(A)$ and the matrix $P_A^\perp = Q_2 Q_2^T$ is the **projection onto the orthogonal complement of** $R(A)$. The above projections can also be computed using the SVD (see **Section 3.9.2**).

MATLAB note: MATLAB function orth(A) computes the orthonormal basis for $R(A)$.

QR Factorization with Column Pivoting

If A is rank-deficient, then QR factorization cannot be used to find a basis for $R(A)$. In this case, one needs to use a modification of the QR factorization process, called *QR factorization with column pivoting.*

We shall not discuss this here. The process finds a permutation matrix P, and the matrices Q and R such that $AP = QR$. The details are given in Golub and Van Loan (1996, pp. 248–250).

MATLAB function $[Q, R, P] = QR(A)$ can be used to compute the QR factorization with column pivoting.

Also, $[Q, R, E] = QR(A, 0)$ produces an economy size QR factorization in which E is a permutation vector so that $Q^*R = A(:, E)$.

3.8 THE LEAST-SQUARES PROBLEM

One of the most important applications of the QR factorization of a matrix A is that it can be effectively used to solve the **least-squares problem** (LSP).

The **linear** LSP is defined as follows:

> Given an $m \times n$ matrix A and a real vector b, find a real vector x such that the function:
>
> $$\| r(x) \|_2 = \| Ax - b \|_2$$
>
> is minimized.

If $m > n$, the problem is called an **overdetermined LSP**, if $m < n$, it is called an **underdetermined problem**.

We will discuss here only the overdetermined problem.

Theorem 3.8.1. *Theorem on Existence and Uniqueness of the LSP. The least-squares solution to $Ax = b$ always exists. The solution is unique if and only if A has full rank. Otherwise, it has infinitely many solutions. The unique solution x is obtained by solving $A^T Ax = A^T b$.*

Proof. See Datta (1995, p. 318). ■

3.8.1 Solving the Least-Squares Problem Using Normal Equations

The expression of the unique solution in Theorem 3.8.1 immediately suggests the following procedure, called the **Normal Equations** method, for solving the LSP:

1. Compute the **symmetric positive definite matrix** A^TA (Note that if A has full rank, A^TA is symmetric positive definite).
2. Solve for x: $A^TAx = A^Tb$.

Computational remarks. The above procedure, though simple to understand and implement, has **serious numerical difficulties**. First, some significant figures may be lost during the explicit formation of the matrix A^TA. Second, **the matrix A^TA will be more ill-conditioned, if A is ill-conditioned**. In fact, it can be shown that $\text{Cond}_2(A^TA) = (\text{Cond}_2(A))^2$. The following simple example illustrates the point.

Let

$$A = \begin{pmatrix} 1 & 1 \\ 10^{-4} & 0 \\ 0 & 10^{-4} \end{pmatrix}.$$

If eight-digit arithmetic is used, then $A^TA = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$, which is **singular**, though the columns of A are linearly independent.

A computationally effective method via the QR factorization of A is now presented below.

3.8.2 Solving the Least-Squares Problem Using *QR* Factorization

Let $Q^TA = R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$ be the QR decomposition of the matrix A. Then, since the length of a vector is preserved by an orthogonal matrix multiplication, we have

$$\begin{aligned} \|Ax - b\|_2^2 &= \|Q^TAx - Q^Tb\|_2^2 \\ &= \|R_1x - c\|_2^2 + \|d\|_2^2, \quad \text{where } Q^Tb = \begin{pmatrix} c \\ d \end{pmatrix}. \end{aligned}$$

Thus, $\|Ax - b\|_2^2$ will be minimized if x is chosen so that $R_1x - c = 0$. The corresponding residual norm then is given by $\|r\|_2 = \|Ax - b\|_2 = \|d\|_2$. This observation immediately suggests the following QR algorithm for solving the LSP:

Algorithm 3.8.1. *Least Squares Solution Using QR Factorization of A*

Inputs. *A—An $m \times n$ matrix ($m \geq n$)*

b—An m-vector.

Output. *The least-squares solution x to the linear system $Ax = b$.*
Step 1. *Decompose $A = QR$, where $Q \in \mathbb{R}^{m\times m}$ and $R \in \mathbb{R}^{m\times n}$.*
Step 2. *Form $Q^{\mathrm{T}}b = \binom{c}{d}$, $c \in \mathbb{R}^{n\times 1}$.*
Step 3. *Obtain x by solving the upper triangular system: $R_1 x = c$ where $R = \binom{R_1}{0}$.*
Step 4. *Obtain the residual norm: $\| r \|_2 = \| d \|_2$.*

Example 3.8.1. Solve $Ax = b$ for x with

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{pmatrix}, \qquad b = \begin{pmatrix} 3 \\ 5 \\ 9 \end{pmatrix}.$$

Step 1. Find the QR factorization of A : $A = QR$

$$Q = \begin{pmatrix} -0.2673 & 0.8729 & 0.4082 \\ -0.5345 & 0.2182 & -0.8165 \\ -0.8018 & -0.4364 & 0.4082 \end{pmatrix},$$

$$R = \left(\begin{array}{cc} -3.7417 & -5.3452 \\ 0 & 0.6547 \\ \hline 0 & 0 \end{array}\right) = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}.$$

Step 2. Form

$$Q^{\mathrm{T}}b = \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} -10.6904 \\ -0.2182 \\ 0.8165 \end{pmatrix}.$$

Step 3. Obtain x by solving $R_1 x = c$: $x = \begin{pmatrix} 3.3532 \\ -0.3333 \end{pmatrix}$.

Step 4. $\| r \|_2 = \| d \|_2 = 0.8165$.

Use of Householder Matrices

Note that if the Householder's or Givens' method is used to compute the QR decomposition of A, then the product $Q^{\mathrm{T}}b$ can be formed from the factored form of Q without explicitly computing the matrix Q.

MATCOM and MATLAB notes: MATCOM function **lsfrqrh** implements the QR factorization method for the full-rank least-squares problem using Householder's method. Alternatively, one can use the MATLAB operator: \. The command $x = A\backslash b$ gives the least-squares solution to $Ax = b$.

Flop-count and numerical stability: The least-squares method, using Householder's QR factorization, requires about $2(mn^2 - (n^3/3))$ flops. **The algorithm is numerically stable** in the sense that the computed solution satisfies a **"nearby"** LSP.

3.9 THE SINGULAR VALUE DECOMPOSITION (SVD)

We have seen two factorizations (decompositions) of A: LU and QR.

In this section we shall study another important decomposition, called the **singular value decomposition** or in short the **SVD** of A. Since $m \geq n$ is the case mostly arising in applications, we **will assume throughout this section that** $m \geq n$.

Theorem 3.9.1. *The SVD Theorem. Given $A \in \mathbb{R}^{m\times n}$, there exist orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$, and a diagonal matrix $\Sigma \in \mathbb{R}^{m\times n}$ with nonnegative diagonal entries such that*

$$A = U\Sigma V^{\mathrm{T}}.$$

Proof. See Datta (1995, pp. 552–554). ■

The diagonal entries of Σ are called the **singular values** of A.

The columns of U are called the **left singular vectors**, and those of V are called the **right singular vectors. The singular values are unique, but U and V are not unique**.

The number of nonzero singular values is equal to the rank of the matrix A.

A convention. The n singular values $\sigma_1, \sigma_2, \ldots, \sigma_n$ of A can be arranged in nondecreasing order: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$. The largest singular value σ_1 is denoted by $\sigma_{\max}$. Similarly, the smallest singular value σ_n is denoted by $\sigma_{\min}$.

The thin SVD. Let $U = (u_1, \ldots, u_m)$.

If $A = U\Sigma V^{\mathrm{T}}$ be the SVD of $A \in \mathbb{R}^{m\times n}$ and if $U_1 = (u_1, \ldots, u_n) \in \mathbb{R}^{m\times n}$, $\Sigma_1 = \mathrm{diag}(\sigma, \ldots, \sigma_n)$, then $A = U_1\Sigma_1 V^{\mathrm{T}}$.

This factorization is known as the **thin SVD of** A. For obvious reasons, the thin SVD is also referred to as the **economic SVD**.

Relationship between eigenvalues and singular values. It can be shown that (see Datta (1995, pp. 555–557)).

1. The singular values $\sigma_1, \ldots, \sigma_n$ of A are the nonnegative square roots of the eigenvalues of the symmetric positive semidefinite matrix $A^{\mathrm{T}}A$.
2. The right singular vectors are the eigenvectors of the matrix $A^{\mathrm{T}}A$, and the left singular vectors are the eigenvectors of the matrix AA^{T}.

Sensitivity of the singular values. A remarkable property of the singular values is that **they are insensitive** to **small perturbations**. In other words, the **singular values are well-conditioned**. Specifically, the following result holds.

Theorem 3.9.2. *Insensitivity of the Singular Values. Let A be an $m \times n$ ($m \geq n$) matrix with the singular values $\sigma_1, \ldots, \sigma_n$, and $B = A+E$ be another slightly perturbed matrix with the singular values $\bar{\sigma}_1, \ldots, \bar{\sigma}_n$, then $| \bar{\sigma}_i - \sigma_i | \leq \| E \|_2$, $i = 1, \ldots, n$.*

Proof. See Datta (1995, pp. 560–561). ■

Example 3.9.1. Let

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}, \qquad E = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 2 \times 10^{-4} \end{pmatrix}.$$

The singular values of A:

$$\sigma_1 = 14.5576, \qquad \sigma_2 = 1.0372, \qquad \sigma_3 = 0.$$

The singular values of $A + E$:

$$\bar{\sigma}_1 = 14.5577, \qquad \bar{\sigma}_2 = 1.0372, \qquad \bar{\sigma}_3 = 2.6492 \times 10^{-5}$$

It is easily verified that the inequalities in the above theorem are satisfied.

3.9.1 The Singular Value Decomposition and the Structure of a Matrix

The SVD is an effective tool in handling several computationally sensitive computations, such as the **rank** and **rank-deficiency** of matrix, the **distance of a matrix from a matrix of immediate lower rank, the orthogonormal basis and projections**, etc. It is also a reliable and numerically stable way of computing the least-squares solution to a linear system. Since these computations need to be performed routinely in control and systems theory, we now discuss them briefly in the following. The results of Theorem 3.9.3 can be easily proved.

Theorem 3.9.3. *Let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ be the n singular values of an $m \times n$ matrix $A (m \geq n)$. Then,*

1. $\|A\|_2 = \sigma_1 = \sigma_{\max}$,
2. $\|A\|_F = (\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_n^2)^{1/2}$,
3. $\|A^{-1}\|_2 = \dfrac{1}{\sigma_n}$, *when A is $n \times n$ and nonsingular,*
4. $\text{Cond}_2(A) = \|A\|_2 \|A^{-1}\|_2 = \dfrac{\sigma_1}{\sigma_n} = \dfrac{\sigma_{\max}}{\sigma_{\min}}$, *when A is $n \times n$ and nonsingular.*

The Condition Number of a Rectangular Matrix

The condition number (with respect to 2-norm) of a rectangular matrix A of order $m \times n$ ($m \geq n$) with full rank is defined to be

$$\text{Cond}_2(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)},$$

where $\sigma_{\max}(A)$ and $\sigma_{\min}(A)$ denote, respectively, the largest and smallest singular value of A.

Remark

- When A is rank-deficient, $\sigma_{\min} = 0$, and we say that $\text{Cond}(A)$ is **infinite**.

3.9.2 Orthonormal Bases and Orthogonal Projections

Let r be the rank of A, that is,

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0,$$
$$\sigma_{r+1} = \cdots = \sigma_n = 0.$$

Let u_j and v_j be the jth columns of U and V in the SVD of A. **Then the set of columns $\{v_j\}$ corresponding to the zero singular values of A form an orthonormal basis for the null-space of** A. This is because, when $\sigma_j = 0$, v_j satisfies $Av_j = 0$ and is therefore in the null-space of A. Similarly, **the set of columns $\{u_j\}$ corresponding to the nonzero singular values is an orthonormal basis for the range of** A. The orthogonal projections now can be easily computed.

Orthogonal Projections

Partition U and V as

$$U = (U_1, U_2), \qquad V = (V_1, V_2),$$

where U_1 and V_1 consist of the first r columns of U and V, then

1. Projection onto $R(A) = U_1 U_1^{\mathrm{T}}$.
2. Projection onto $N(A) = V_2 V_2^{\mathrm{T}}$.
3. Projection onto the orthogonal complement of $R(A) = U_2 U_2^{\mathrm{T}}$.
4. Projection onto the orthogonal complement of $N(A) = V_1 V_1^{\mathrm{T}}$.

Example 3.9.2.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}.$$

$$\sigma_1 = 14.5576, \qquad \sigma_2 = 1.0372, \qquad \sigma_3 = 0.$$

$$U = \begin{pmatrix} 0.2500 & 0.8371 & 0.4867 \\ 0.4852 & 0.3267 & -0.8111 \\ 0.8378 & -0.4379 & 0.3244 \end{pmatrix}.$$

$$V = \begin{pmatrix} 0.4625 & -0.7870 & 0.4082 \\ 0.5706 & -0.0882 & -0.8165 \\ 0.6786 & -0.6106 & 0.4082 \end{pmatrix}.$$

An orthonormal basis for the null-space of A is:

$$V_2 = \begin{Bmatrix} 0.4082 \\ -0.8165 \\ 0.4082 \end{Bmatrix}.$$

An orthonormal basis for the range of A is:

$$U_1 = \begin{Bmatrix} 0.2500 & 0.8371 \\ 0.4852 & 0.3267 \\ 0.8370 & -0.4379 \end{Bmatrix}.$$

(Now compute the four orthogonal projections yourself.)

3.9.3 The Rank and the Rank-Deficiency of a Matrix

The most obvious and the least expensive way of determining the rank of a matrix is, of course, to triangularize the matrix using Gaussian elimination and then to find the rank of the reduced upper triangular matrix. Finding the rank of a triangular matrix is trivial; one can just read it off from the diagonal. Unfortunately, however, this is not a very reliable approach in floating point arithmetic. Gaussian elimination method which uses elementary transformations, may transform a rank-deficient matrix into one having full rank, due to numerical round-off errors. Thus, in practice, it is more important, **to determine if the given matrix is near a matrix of a certain rank and in particular, to know if it is near a rank-deficient matrix. The most reliable way to determine the rank and nearness to rank-deficiency is to use the SVD.**

Suppose that A has rank r, that is, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ and $\sigma_{r+1} = \cdots = \sigma_n = 0$. Then the question is: *How far is A from a matrix of rank $k < r$.* The following theorem can be used to answer the question. We state the theorem below, without proof. For proof, see Datta (1995, pp. 565–566).

Theorem 3.9.4. *Distance to Rank-Deficient Matrices. Let $A = U\Sigma V^{\mathrm{T}}$ be the SVD of A, and let* $\mathrm{rank}(A) = r > 0$. *Let $k < r$. Define $A_k = U\Sigma_k V^{\mathrm{T}}$,*

where

$$\Sigma_k = \left(\begin{array}{ccc|c} \sigma_1 & & 0 & \\ & \ddots & & 0 \\ 0 & & \sigma_k & \\ \hline & & 0 & 0 \end{array}\right), \quad (\sigma_1 \geq \sigma_2 \cdots \geq \sigma_k > 0).$$

1. *Then out of all the matrices of rank $k (k < r)$, the matrix A_k is closest to A.*
2. *Furthermore, the distance of A_k from A: $\|A - A_k\|_2 = \sigma_{k+1}$.*

Corollary 3.9.1. *The relative distance of a nonsingular matrix A to the nearest singular matrix B is $1/\text{Cond}_2(A)$. That is, $\|B - A\|_2/\|A\|_2 = 1/\text{Cond}_2(A)$.*

Implication of the Above Results

Distance of a Matrix to the Nearest Matrix of Lower Rank

The above result states that the smallest nonzero singular value of A gives the 2-norm distance of A to the nearest matrix of lower rank. In particular, for a nonsingular $n \times n$ matrix A, σ_n gives the measures of the distance of A to the nearest singular matrix.

Thus, in order to know if a matrix A of rank r is close enough to a matrix of lower rank, look into the smallest nonzero singular value σ_r. If this is very small, then the matrix is very close to a matrix of rank $r - 1$, because there exists a perturbation of size as small as $|\sigma_r|$ which will produce a matrix of rank $r - 1$. In fact, one such perturbation is $u_r \sigma_r v_r^{\mathrm{T}}$.

Example 3.9.3. Let

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \times 10^{-7} \end{pmatrix},$$

$$\text{Rank}(A) = 3, \qquad \sigma_3 = 0.0000004, \qquad u_3 = v_3 = (0, 0, 1)^{\mathrm{T}}.$$

$$A' = A - u_3 \sigma_3 v_3^{\mathrm{T}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \qquad \text{rank}(A') = 2.$$

The required perturbation $u_3\sigma_3 v_3^T$ to make A singular is:

$$10^{-7}\begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 4\end{pmatrix}.$$

3.9.4 Numerical Rank

The above discussions prompt us to define the concept of **"Numerical Rank"** of a matrix. A has **"numerical rank"** r if the computed singular values $\tilde{\sigma}_1, \tilde{\sigma}_2, \ldots, \tilde{\sigma}_n$ satisfy:

$$\tilde{\sigma}_1 \geq \tilde{\sigma}_2 \geq \cdots \geq \tilde{\sigma}_r > \delta \geq \tilde{\sigma}_{r+1} \geq \cdots \geq \tilde{\sigma}_n, \tag{3.9.1}$$

where δ is an error tolerance.

Thus to determine the numerical rank of a matrix A, count the "large" singular values only. If this number is r, then A has numerical rank r.

Remark

- Note that finding the numerical rank of a matrix will be "tricky" if there is no suitable gap between a set of singular values.

3.9.5 Solving the Least-Squares Problem Using the Singular Value Decomposition

The SVD is also an **effective tool** to solve the LSP, both in the full rank and rank-deficient cases.

Recall that the **linear LSP** is: Find x such that $\|r\|_2 = \|Ax - b\|_2$ is minimum.

Let $A = U\Sigma V^T$ be the SVD of A. Then since U is orthogonal and a vector length is preserved by orthogonal multiplication, we have

$$\|r\|_2 = \|(U\Sigma V^T x - b)\|_2 = \|U(\Sigma V^T x - U^T b)\|_2 = \|\Sigma y - b'\|_2,$$

where $V^T x = y$ and $U^T b = b'$. **Thus, the use of the SVD of A reduces the LSP for a full matrix A to one with a diagonal matrix Σ,** which is almost trivial to solve, as shown in the following algorithm.

Algorithm 3.9.1. *Least Squares Solutions Using the SVD*
Inputs. *A—An $m \times n$ matrix,*
b—An m-vector
Output. *x—The least-squares solution of the system $Ax = b$.*
Step 1. *Find the SVD of A: $A = U\Sigma V^T$.*

Step 2. *Form* $b' = U^{\mathrm{T}}b = \begin{pmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_m \end{pmatrix}$.

Step 3. *Compute*

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

choosing

$$y_i = \begin{Bmatrix} \dfrac{b'_i}{\sigma_i}, \text{ when } \sigma_i \neq 0 \\ \text{arbitrary, when } \sigma_i = 0. \end{Bmatrix}.$$

Step 4. *Compute the family of least squares solutions:* $x = Vy$. **(Note that in the full-rank case, the family has just one number).**

Flop-count. Using the SVD, it takes about $4mn^2 + 8n^3$ flops to solve the LSP, when A is $m \times n$ and $m \geq n$.

An Expression for the Minimum Norm Least Squares Solution

Since a rank-deficient LSP has an infinite number of solutions, it is practical to look for the one that has minimum norm. Such a solution is called the **minimum norm least square solution.**

It is clear from Step 3 above that in the **rank-deficient case**, the minimum 2-norm least squares solution is the one that is obtained by setting $y_i = 0$, whenever $\sigma_i = 0$. Thus, from above, we have the following expression for the minimum 2-norm solution:

$$x = \sum_{i=1}^{k} \frac{u_i^{\mathrm{T}} b'_i}{\sigma_i} v_i, \tag{3.9.2}$$

where k is the **numerical rank** of A, and u_i and v_i, respectively, are the ith columns of U and V.

Example 3.9.4.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 1 & 2 & 3 \end{pmatrix}, \qquad b = \begin{pmatrix} 6 \\ 9 \\ 6 \end{pmatrix}.$$

Step 1. $\sigma_1 = 7.5358$, $\sigma_2 = 0.4597$, $\sigma_3 = 0$.

A is rank-deficient.

$$U = \begin{pmatrix} 0.4956 & 0.5044 & 0.7071 \\ 0.7133 & -0.7008 & 0.0000 \\ 0.4956 & 0.5044 & -0.7071 \end{pmatrix},$$

$$V = \begin{pmatrix} 0.3208 & -0.8546 & 0.4082 \\ 0.5470 & -0.1847 & -0.8165 \\ 0.7732 & 0.4853 & 0.4082 \end{pmatrix}.$$

Step 2. $b' = U^{\mathrm{T}}b = (12.3667, -0.2547, 0)^{\mathrm{T}}$.
Step 3. $y = (1.6411, -0.5541, 0)$.
The minimum 2-norm least-squares solution is $Vy = (1, 1, 1)^{\mathrm{T}}$.

Computing the SVD of A

Since the singular values of a matrix A are the nonnegative square roots of the eigenvalues of $A^{\mathrm{T}}A$, it is natural to think of computing the singular values and the singular vectors, by finding the eigendecomposition of $A^{\mathrm{T}}A$. However, **this is not a numerically effective procedure**.

Some vital information may be lost during the formation of the matrix $A^{\mathrm{T}}A$, as the following example shows.

Example 3.9.5.

$$A = \begin{pmatrix} 1.0001 & 1.000 \\ 1.000 & 1.0001 \end{pmatrix}.$$

The singular values of A are 2.0001 and 0.0001.

$$A^{\mathrm{T}}A = \begin{pmatrix} 2.0002 & 2.0002 \\ 2.0002 & 2.0002 \end{pmatrix}.$$

The eigenvalues of $A^{\mathrm{T}}A$ are 0 and 4.0004 (in four-digit arithmetic). Thus, the singular values computed from the eigenvalues of $A^{\mathrm{T}}A$ are 0 and 2.0002.

A standard algorithm for computing the SVD of A is the **Golub–Kahan–Reinsch** algorithm. The algorithm will be described later in the book in **Chapter 4**.

MATLAB and MATCOM notes: MATLAB function **svd** can be used to compute the SVD. $[U, S, V] = \boldsymbol{svd}(A)$ produces a diagonal matrix S, of the same dimension as A and with nonnegative diagonal entries in decreasing order, and unitary matrices U and V such that $A = USV^*$.

Algorithm 3.9.1 has been implemented in MATCOM program **lsqrsvd**. Also, MATCOM has a program called **minmsvd** to compute the minimum 2-norm least-squares solution using the SVD.

3.10 SUMMARY AND REVIEW

Floating Point Numbers and Errors

1. *Floating-point numbers.* A t-digit floating point number has the form:

$$x = m\beta^e,$$

where e is called exponent, m is a t-digit fraction, and β is the base of the number system.

2. *Errors.* The errors in a computation are measured either by absolute error or relative error. **The relative errors make more sense than absolute errors.** The relative error gives an indication of the number of significant digits in an approximate answer. The relative error in representing a real number x by its floating-point representation $\text{fl}(x)$ is bounded by a number μ, called the **machine precision (Theorem 3.2.1).**

3. *Laws of floating-point arithmetic:*

$$\text{fl}(x * y) = (x * y)(1 + \delta).$$

Conditioning, Stability, and Accuracy

1. *Conditioning of the problem.* The conditioning of the problem is a property of the problem. A problem is said to be **ill-conditioned** if a small change in the data can cause a large change in the solution, otherwise it is **well-conditioned.** The conditioning of a problem is data-dependent. A problem can be ill-conditioned with respect to one set of data but can be quite well-conditioned with respect to another set.

   The condition number of a nonsingular matrix, $\text{Cond}(A) = \|A\| \|A^{-1}\|$ is an indicator of the conditioning of the associated linear system problem: $Ax = b$. If $\text{Cond}(A)$ is large, then the linear system $Ax = b$ is ill-conditioned.

   The well-known examples of ill-conditioned problems are the **Wilkinson polynomial** for the root-finding problem, the **Wilkinson bidiagonal matrix** for the eigenvalue problem, the **Hilbert matrix** for the algebraic linear system problem, and so on.

2. *Stability of an algorithm.* An algorithm is said to be a *backward stable algorithm* if it computes the exact solution of a nearby problem. Some examples of stable algorithms are the methods of back substitution and forward elimination for triangular systems, the QR factorization using Householder and Givens matrices transformations, and the QR iteration algorithm for eigenvalue computations.

The Gaussian elimination algorithm without row changes is unstable for arbitrary matrices. However, Gaussian elimination with partial pivoting can be considered as a stable algorithm in practice.

3. *Effects of conditioning and stability on the accuracy of the solution.* The conditioning of the problem and the stability of the algorithm both have effects on accuracy of the solution computed by the algorithm.

   If a stable algorithm is applied to a well-conditioned problem, it should compute an accurate solution. On the other hand, if a stable algorithm is applied to an ill-conditioned problem, there is no guarantee that the computed solution will be accurate. However, if a stable algorithm is applied to an ill-conditioned problem, it should not introduce more errors than that which the data warrants.

Matrix Factorizations

There are three important matrix factorizations: LU, QR, and SVD.

1. *LU factorization.* A factorization of a matrix A in the form $A = LU$, where L is unit lower triangular and U is upper triangular, is called an LU factorization of A. An LU factorization of A exists if all of its leading principal minors are nonsingular.

   A classical elimination scheme, called **Gaussian elimination**, is used to obtain an LU factorization of A (**Section 3.4.1**).

   The stability of Gaussian elimination is determined by the **growth factor**

$$\rho = \frac{\max(\alpha, \alpha_1, \ldots, \alpha_{n-1})}{\alpha},$$

   where $\alpha = \max_{i,j} |a_{ij}|$ and $\alpha_k = \max_{i,j} |a_{ij}^{(k)}|$.

   If no pivoting is used in Gaussian elimination, ρ can be arbitrarily large. Thus, **Gaussian elimination without pivoting is, in general, an unstable process.**

   If partial pivoting is used, then Gaussian elimination yields the factorization of A in the form $PA = LU$, where P is a perturbation matrix.

   The growth factor ρ for Gaussian elimination with partial pivoting can be as large as 2^{n-1}; however, such a growth is extremely rare in practice. Thus, **Gaussian elimination with partial pivoting is considered to be a stable process in practice.**

2. *The QR factorization.* Given an $m \times n$ matrix A, there exists an orthogonal matrix Q and an upper triangular matrix R such that $A = QR$.

The QR factorization of A can be obtained using **Householder's method, Givens' method, the Gram–Schmidt processes (the CGS and MGS)**.

The Gram–Schmidt processes do not have favorable numerical properties. **Both Householder's and Givens' methods are numerically stable procedures for QR factorization.** They are discussed, respectively, in **Section 3.6.2 and Section 3.6.4 (Algorithm 3.6.1)**. Householder's method is slightly more efficient than Givens' method.

The Algebraic Linear System Problem $Ax = b$

The method of practical choice for the linear system problem $Ax = b$ is Gaussian elimination with partial pivoting (**Section 3.5.2**) followed by iterative refinement procedure (**Section 3.5.7**). A symmetric positive definite system should be solved by computing its Cholesky factor (**Algorithm 3.4.1**) R followed by solving two triangular systems: $Ry = b$ and $R^{\mathrm{T}}x = y$ (**Algorithm 3.3.1** and **Section 3.3.3**).

The Least-Squares Problem

Given an $m \times n$ matrix A, the LSP is the problem of finding a vector x such that $\| Ax - b \|_2$ is minimized. The LSP can be solved using:

- The normal equations method (**Section 3.8.1**): $A^{\mathrm{T}}Ax = A^{\mathrm{T}}b$
- The QR factorization method (**Algorithm 3.8.1**)
- The SVD method (**Algorithm 3.9.1**).

The normal equations method might give numerical difficulties, and should not be used in practice without looking closely at the condition number. Both the QR and SVD methods for the LSP are numerically stable. Though the SVD is more expensive than the QR method, **the SVD method is most reliable and can handle both rank-deficient and full-rank cases very effectively**.

The Singular Value Decomposition

1. *Existence and uniqueness of the SVD.* The SVD of a matrix A always exists (**Theorem 3.9.1**):

   Let $A \in \mathbb{R}^{m \times n}$. Then $A = U\Sigma V^{\mathrm{T}}$, where $U \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{n \times n}$ are orthogonal and Σ is an $m \times n$ diagonal matrix.

   The singular values (the diagonal entries of Σ) are unique, but U and V are not unique.
2. *Relationship between the singular values and the eigenvalues.* The singular values of A are the nonnegative square roots of the eigenvalues of $A^{\mathrm{T}}A$ (or of AA^{T}).

3. *Sensitivity of the singular values.* The singular values are insensitive to small perturbations (**Theorem 3.9.2**).
4. *Applications of the SVD.* The singular values and the singular vectors of a matrix A are useful and are the most reliable tools for determining the (numerical) rank and the rank-deficiency of A; finding the orthonormal bases for range and the null space of A; finding the distance of A from another matrix of lower rank (in particular, the nearness to singularity of a nonsingular matrix); solving both full-rank and the rank-deficient LSPs.

   These remarkable abilities and the fact that the singular values are insensitive to small perturbations have made the SVD an indispensable tool for a wide variety of problems in control and systems theory, as we will see throughout the book.

3.11 CHAPTER NOTES AND FURTHER READING

Material of this chapter has been taken from the recent book of the author (Datta 1995). For the advanced topics on numerical linear algebra, see Golub and Van Loan (1996). The details about stability of various algorithm and sensitivities of problems described in this chapter can be found in the book by Higham (1996). Stewart's (1998) recent book is also an excellent source of knowledge in this area. For details of various MATLAB functions, see MATLAB Users' Guide (1992). *MATCOM* is a MATLAB-based toolbox implementing all the major algorithms of the book "*Numerical Linear Algebra and Applications*" by Datta (1995). MATCOM can be obtained from the book's web page on the web site of MATHWORKS: http://www.mathworks.com/support/books/book1329.jsp. The software (MATCOM) is linked at the bottom.

References

Datta B.N. *Numerical Linear Algebra and Applications*, Brooks/Cole Publishing Company, Pacific Grove, CA, 1995 (*Custom published by Brooks/Cole, 2003*).

Golub G.H. and Van Loan C.F. *Matrix Computations*, 3rd edn, Johns Hopkins University Press, Baltimore, MD, 1996.

Higham N.J. *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, 1996.

MATCOM web site: http://www.mathworks.com/support/books/book1329.jsp.

MATLAB *User's Guide*, The Math Works, Inc., Natick, MA, 1992.

Stewart G.W. *Matrix Algorithms Volume 1: Basic Decompositions*, SIAM, Philadelphia, 1998.

Wilkinson J.H. *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.