Fundamentals of Numerical Linear Algebra

Seongjai Kim

Department of Mathematics and Statistics
Mississippi State University
Mississippi State, MS 39762 USA
Email: skim@math.msstate.edu

Updated: March 21, 2023

Seongjai Kim, Department of Mathematics and Statistics, Mississippi State University, Mississippi State, MS 39762-5921 USA Email: skim@math.msstate.edu. The work of the author is supported in part by NSF grant DMS-1228337.

Prologue

In organizing this lecture note, I am indebted by Demmel [7], Golub and Van Loan [10], Varga [24], and Watkins [26], among others. Currently the lecture note is not fully grown up; other useful techniques would be soon incorporated. Any questions, suggestions, comments will be deeply appreciated.

Contents

CHAPTER 1

Introduction to Matrix Analysis

Contents of Chapter 1

1.1. Basic Algorithms and Notation

1.1.1. Matrix notation

Let $\mathbb{R}$ be the set of real numbers. We denote the vector space of all $m \times n$ matrices by $\mathbb{R}^{m\times n}$:

$$A \in \mathbb{R}^{m\times n} \quad \Leftrightarrow \quad A = (a_{ij}) = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}, \quad a_{ij} \in \mathbb{R}. \tag{1.1}$$

The subscript ij refers to the (i, j) entry.

1.1.2. Matrix operations

- transpose $(\mathbb{R}^{m\times n} \to \mathbb{R}^{n\times m})$

$$C = A^T \quad \Rightarrow \quad c_{ij} = a_{ji}$$

- addition $(\mathbb{R}^{m\times n} \times \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n})$

$$C = A + B \quad \Rightarrow \quad c_{ij} = a_{ij} + b_{ij}$$

- scalar-matrix multiplication $(\mathbb{R} \times \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n})$

$$C = \alpha A \quad \Rightarrow \quad c_{ij} = \alpha\, a_{ij}$$

- matrix-matrix multiplication $(\mathbb{R}^{m\times p} \times \mathbb{R}^{p\times n} \to \mathbb{R}^{m\times n})$

$$C = A\,B \quad \Rightarrow \quad c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}$$

These are the building blocks of matrix computations.

1.1.3. Vector notation

Let $\mathbb{R}^n$ denote the vector space of real n-vectors:

$$x \in \mathbb{R}^n \quad \Leftrightarrow \quad x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad x_i \in \mathbb{R}. \tag{1.2}$$

We refer to x_i as the ith component of x. We identify $\mathbb{R}^n$ with $\mathbb{R}^{n\times 1}$ so that the members of $\mathbb{R}^n$ are *column* vectors. On the other hand, the elements of $\mathbb{R}^{1\times n}$ are *row* vectors:

$$y \in \mathbb{R}^{1\times n} \quad \Leftrightarrow \quad y = (y_1, \cdots, y_n).$$

If $x \in \mathbb{R}^n$, then $y = x^T$ is a row vector.

1.1.4. Vector operations

Assume $a \in \mathbb{R}$ and x, $y \in \mathbb{R}^n$.

- scalar-vector multiplication

$$z = a\,x \quad \Rightarrow \quad z_i = a\,x_i$$

- vector addition

$$z = x + y \quad \Rightarrow \quad z_i = x_i + y_i$$

- dot product (or, inner product)

$$c = x^T y (= x \cdot y) \quad \Rightarrow \quad c = \sum_{i=1}^{n} x_i y_i$$

- vector multiply (or, Hadamard product)

$$z = x.*y \quad \Rightarrow \quad z_i = x_i y_i$$

- saxpy (“scalar a x plus y"): a LAPACK routine

$$z = a\,x + y \quad \Rightarrow \quad z_i = a\,x_i + y_i$$

1.1.5. Matrix-vector multiplication & "gaxpy"

- gaxpy ("generalized saxpy"): a LAPACK routine

$$z = Ax + y \quad \Rightarrow \quad z_i = \sum_{j=1}^{n} a_{ij}x_j + y_i$$

1.1.6. The colon notation

A handy way to specify a column or row of a matrix is with the "colon" notation. Let $A \in \mathbb{R}^{m\times n}$. Then,

$$A(i,:) = [a_{i1}, \cdots, a_{in}], \quad A(:,j) = \begin{bmatrix} a_{1j} \\ \vdots \\ a_{mj} \end{bmatrix}$$

Example 1.1. *With these conventions, the gaxpy can be written as*

```
for i=1:m
   z(i)=y(i)+A(i,:)x                                (1.3)
end
```

or, in column version,

```
z=y
for j=1:n                                           (1.4)
   z=z+x(j)A(:,j)
end
```

Matlab-code 1.2. The gaxpy can be implemented as simple as

```
z=A*x+y;                                            (1.5)
```

1.1.7. Flops

A **flop** (floating point operation) is any mathematical operation (such as +, -, $*$, /) or assignment that involves floating-point numbers Thus, the gaxpy (1.3) or (1.4) requires $2mn$ flops.

The following will be frequently utilized in counting flops:

$$\begin{aligned} \sum_{k=1}^{n} k &= \frac{n(n+1)}{2}, \\ \sum_{k=1}^{n} k^2 &= \frac{n(n+1)(2n+1)}{6}, \\ \sum_{k=1}^{n} k^3 &= \Big(\sum_{k=1}^{n} k\Big)^2 = \frac{n^2(n+1)^2}{4}. \end{aligned} \tag{1.6}$$

1.2. Vector and Matrix Norms

1.2.1. Vector norms

Definition 1.3. A **norm** (or, **vector norm**) on $\mathbb{R}^n$ is a function that assigns to each $x \in \mathbb{R}^n$ a nonnegative real number $\|x\|$, called the norm of x, such that the following three properties are satisfied: for all x, $y \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$,

$$\begin{aligned} &\|x\| > 0 \text{ if } x \neq 0 && \text{(positive definiteness)} \\ &\|\lambda x\| = |\lambda|\,\|x\| && \text{(homogeneity)} \\ &\|x+y\| \le \|x\| + \|y\| && \text{(triangle inequality)} \end{aligned} \tag{1.7}$$

Example 1.4. *The most common norms are*

$$\|x\|_p = \Big(\sum_i |x_i|^p\Big)^{1/p}, \quad 1 \le p < \infty, \tag{1.8}$$

which we call the p-**norms***, and*

$$\|x\|_\infty = \max_i |x_i|, \tag{1.9}$$

which is called the **infinity-norm** *or* **maximum-norm***.*

Two of frequently used p-norms are

$$\|x\|_1 = \textstyle\sum_i |x_i|, \quad \|x\|_2 = \Big(\textstyle\sum_i |x_i|^2\Big)^{1/2}$$

The 2-norm is also called the Euclidean norm, often denoted by $\|\cdot\|$.

Remark 1.5. One may consider the infinity-norm as the limit of p-norms, as $p \to \infty$; see Homework 1.

Theorem 1.6. ***(Cauchy-Schwarz inequality)*** *For all $x,\, y \in \mathbb{R}^n$,*

$$\Big|\sum_{i=1}^n x_i y_i\Big| \le \Big(\sum_{i=1}^n x_i^2\Big)^{1/2} \Big(\sum_{i=1}^n y_i^2\Big)^{1/2} \tag{1.10}$$

Note: (1.10) can be rewritten as

$$|x \cdot y| \le \|x\|\,\|y\| \tag{1.11}$$

which is clearly true.

Example 1.7. *Given a positive definite matrix $A \in \mathbb{R}^{n\times n}$, define the A-norm on $\mathbb{R}^n$ by*

$$\|x\|_A = (x^T A x)^{1/2}$$

Note: When $A = I$, the A-norm is just the Euclidean norm. The A-norm is indeed a norm; see Homework 2.

Lemma 1.8. *All p-norms on $\mathbb{R}^n$ are equivalent to each other. In particular,*

$$\begin{array}{rcccl} \|x\|_2 & \le & \|x\|_1 & \le & \sqrt{n}\,\|x\|_2 \\ \|x\|_\infty & \le & \|x\|_2 & \le & \sqrt{n}\,\|x\|_\infty \\ \|x\|_\infty & \le & \|x\|_1 & \le & n\,\|x\|_\infty \end{array} \tag{1.12}$$

Note: For all $x \in \mathbb{R}^n$,

$$\|x\|_\infty \le \|x\|_2 \le \|x\|_1 \le \sqrt{n}\,\|x\|_2 \le n\,\|x\|_\infty \tag{1.13}$$

1.2.2. Matrix norms

Definition 1.9. *A* **matrix norm** *on $m \times n$ matrices is a vector norm on the mn-dimensional space, satisfying*

$$\begin{aligned} &\|A\| \ge 0, \ \text{and} \ \|A\| = 0 \ \Leftrightarrow \ A = 0 && \text{(positive definiteness)} \\ &\|\lambda A\| = |\lambda|\,\|A\| && \text{(homogeneity)} \\ &\|A+B\| \le \|A\| + \|B\| && \text{(triangle inequality)} \end{aligned} \tag{1.14}$$

Example 1.10.

- $\max_{i,j} |a_{ij}|$ *is called the* **maximum norm***.*
- $\|A\|_F \equiv \Big(\sum_{i,j} |a_{ij}|^2\Big)^{1/2}$ *is called the* **Frobenius norm***.*

Definition 1.11. *Let $\|\cdot\|_{m\times n}$ be a matrix norm on $m \times n$ matrices. The norms are called* **mutually consistent** *if*

$$\|A\,B\|_{m\times p} \le \|A\|_{m\times n} \cdot \|B\|_{n\times p} \tag{1.15}$$

Definition 1.12. *Lat $A \in \mathbb{R}^{m\times n}$ and $\|\cdot\|_{\widehat{n}}$ is a vector norm on $\mathbb{R}^n$. Then*

$$\|A\|_{\widehat{m}\widehat{n}} = \max_{x\in\mathbb{R}^n,\, x\neq 0} \frac{\|Ax\|_{\widehat{m}}}{\|x\|_{\widehat{n}}} \tag{1.16}$$

is called an **operator norm** *or* **induced norm** *or* **subordinate norm***.*

Lemma 1.13. *Let $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$.*

1. *An operator norm of A is a matrix norm.*
2. *For all operator norms and the Frobenius norm,*

$$\begin{aligned} \|Ax\| &\le \|A\|\,\|x\| \\ \|A\,B\| &\le \|A\|\,\|B\| \end{aligned} \tag{1.17}$$

3. *For the induced 2-norm and the Frobenius norm,*

$$\|P A\, Q\| = \|A\| \tag{1.18}$$

   when P and Q are **orthogonal**, *i.e., $P^{-1} = P^T$ and $Q^{-1} = Q^T$.*
4. *The max norm and the Frobenius norm are not operator norms.*
5. $\|A\|_1 \equiv \max\limits_{x\neq 0} \dfrac{\|Ax\|_1}{\|x\|_1} = \max\limits_{j} \sum\limits_{i} |a_{ij}|$
6. $\|A\|_\infty \equiv \max\limits_{x\neq 0} \dfrac{\|Ax\|_\infty}{\|x\|_\infty} = \max\limits_{i} \sum\limits_{j} |a_{ij}|$
7. $\|A\|_2 \equiv \max\limits_{x\neq 0} \dfrac{\|Ax\|_2}{\|x\|_2} = \sqrt{\lambda_{\max}(A^T A)}$,
   where $\lambda_{\max}$ denotes the largest eigenvalue.
8. $\|A\|_2 = \|A^T\|_2$.
9. $\|A\|_2 = \max\limits_{i} |\lambda_i(A)|$, *when $A^T A = A A^T$ (normal).*

Lemma 1.14. *Let $A \in \mathbb{R}^{n\times n}$. Then*

$$\begin{aligned} \frac{1}{\sqrt{n}}\|A\|_2 &\le \|A\|_1 \le \sqrt{n}\,\|A\|_2 \\ \frac{1}{\sqrt{n}}\|A\|_2 &\le \|A\|_\infty \le \sqrt{n}\,\|A\|_2 \\ \frac{1}{n}\|A\|_\infty &\le \|A\|_1 \le n\,\|A\|_\infty \\ \|A\|_1 &\le \|A\|_F \le \sqrt{n}\,\|A\|_2 \end{aligned} \tag{1.19}$$

Example 1.15. *Let $u,\, v \in \mathbb{R}^n$ and let $A = uv^T$. This is a matrix of rank one. Let's prove*

$$\|A\|_2 = \|u\|_2\,\|v\|_2 \tag{1.20}$$

Proof.

1.2.3. Condition numbers

Definition 1.16. *Let $A \in \mathbb{R}^{n\times n}$. Then*

$$\kappa(A) \equiv \|A\|\,\|A^{-1}\|$$

is called the **condition number** *of A, associated to the matrix norm.*

Lemma 1.17.

1. $\kappa(A) = \kappa(A^{-1})$
2. $\kappa(cA) = \kappa(A)$ *for any* $c \neq 0$.
3. $\kappa(I) = 1$ *and* $\kappa(A) \geq 1$, *for any induced matrix norm.*

Theorem 1.18. *If A is nonsingular and*

$$\frac{\|\delta A\|}{\|A\|} \leq \frac{1}{\kappa(A)}, \tag{1.21}$$

then $A + \delta A$ is nonsingular.

Theorem 1.19. *Let $A \in \mathbb{R}^{n\times n}$ be nonsingular, and x and $\widehat{x} = x + \delta x$ be the solutions of*

$$A\,x = b \;\text{ and }\; A\,\widehat{x} = b + \delta b,$$

respectively. Then

$$\frac{\|\delta x\|}{\|x\|} \le \kappa(A)\,\frac{\|\delta b\|}{\|b\|}. \tag{1.22}$$

Proof. The equations

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

imply $A\delta x = \delta b$, that is, $\delta x = A^{-1}\delta b$. Whatever vector norm we have chosen, we will use the induced matrix norm to measure matrices. Thus

$$\|\delta x\| \le \|A^{-1}\|\,\|\delta b\| \tag{1.23}$$

Similarly, the equation $b = A\,x$ implies $\|b\| \le \|A\|\,\|x\|$, or equivalently

$$\frac{1}{\|x\|} \le \|A\|\,\frac{1}{\|b\|}. \tag{1.24}$$

The claim follows from (1.23) and (1.24). □

Example 1.20. Let $A = \begin{bmatrix} 1 & 2 & -2 \\ 0 & 4 & 1 \\ 1 & -2 & 2 \end{bmatrix}$. Then, we have

$$A^{-1} = \frac{1}{20} \begin{bmatrix} 10 & 0 & 10 \\ 1 & 4 & -1 \\ -4 & 4 & 4 \end{bmatrix} \text{ and } A^T A = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 24 & -4 \\ 0 & -4 & 9 \end{bmatrix}.$$

1. Find $\|A\|_1$, $\|A\|_\infty$, and $\|A\|_2$.
2. Compute the ℓ_1-condition number $\kappa_1(A)$.

Solution.

Remark 1.21. *Numerical analysis involving the condition number is a useful tool in scientific computing. The condition number has been involved in analyses of*

- *Accuracy, due to the error involved in the data*
- *Stability of algebraic systems*
- *Convergence speed of iterative algebraic solvers*

The condition number is one of most frequently-used measurements for matrices.

1.3. Numerical Stability

Sources of error in numerical computation:

Example: Evaluate a function $f : \mathbb{R} \to \mathbb{R}$ at a given x.

- x is not exactly known
  - measurement errors
  - errors in previous computations
- The algorithm for computing $f(x)$ is not exact
  - discretization (e.g., it uses a table to look up)
  - truncation (e.g., truncating a Taylor series)
  - rounding error during the computation

The condition of a problem: sensitivity of the solution with respect to errors in the data

- A problem is *well-conditioned* if small errors in the data produce small errors in the solution.
- A problem is *ill-conditioned* if small errors in the data may produce large errors in the solution

1.3.1. Stability in numerical linear algebra

Theorem 1.22. *(Revisit of Theorem 1.19) Let $A \in \mathbb{R}^{n\times n}$ be nonsingular, and x and $\widehat{x} = x + \delta x$ be the solutions of*

$$A\,x = b \;\text{ and }\; A\,\widehat{x} = b + \delta b,$$

respectively. Then

$$\frac{\|\delta x\|}{\|x\|} \le \kappa(A)\,\frac{\|\delta b\|}{\|b\|}, \tag{1.25}$$

where $\kappa(A) \equiv \|A\|\,\|A^{-1}\|$.

Example 1.23. *Let*

$$A = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1+10^{-10} & 1-10^{-10} \end{bmatrix}.$$

Then

$$A^{-1} = \begin{bmatrix} 1-10^{10} & 10^{10} \\ 1+10^{10} & -10^{10} \end{bmatrix}.$$

- *Solution for $b = (1,1)^T$ is $x = (1,1)^T$.*
- *If we change b to $b + \Delta b$, then the change in the solution is*

$$\Delta x = A^{-1}\Delta b = \begin{bmatrix} \Delta b_1 - 10^{10}(\Delta b_1 - \Delta b_2) \\ \Delta b_1 + 10^{10}(\Delta b_1 - \Delta b_2) \end{bmatrix}.$$

  Small Δb may lead to an extremely large Δx

- *The condition number $\kappa(A) = 2 \times 10^{10} + 1$.*

1.3.2. Stability in numerical ODEs/PDEs

Physical Definition: A (FD) scheme is **stable** if a small change in the initial conditions produces a small change in the state of the system.

- Most aspects in the nature are stable.
- Some phenomena in the nature can be represented by differential equations (ODEs and PDEs), while they may be solved through difference equations.
- Although ODEs and PDEs are stable, their approximations (finite difference equations) may not be stable. In this case, the approximation is a failure.

Definition: A differential equation is

- **stable** if for every set of initial data, the solution remains bounded as $t \to \infty$.
- **strongly stable** if the solution approaches zero as $t \to \infty$.

Exercises for Chapter 1

1.1. This problem proves that the infinite-norm is the limit of the p-norms as $p \to \infty$.

(a) Verify that

$$\lim_{p\to\infty} (1+x^p)^{1/p} = 1, \quad \text{for each} \quad |x| \le 1$$

(b) Let $x = (a,b)^T$ with $|a| \ge |b|$. Prove

$$\lim_{p\to\infty} \|x\|_p = |a| \tag{1.26}$$

Equation (1.26) implies $\lim\limits_{p\to\infty} \|x\|_p = \|x\|_\infty$ for $x \in \mathbb{R}^2$.

(c) Generalize the above arguments to prove that

$$\lim_{p\to\infty} \|x\|_p = \|x\|_\infty \quad \text{for} \quad x \in \mathbb{R}^n \tag{1.27}$$

1.2. Let $A \in \mathbb{R}^{n\times n}$ be a positive definite matrix, and R be its Cholesky factor so that $A = R^T R$.

(a) Verify that

$$\|x\|_A = \|Rx\|_2 \quad \text{for all} \quad x \in \mathbb{R}^n \tag{1.28}$$

(b) Using the fact that the 2-norm is indeed a norm on $\mathbb{R}^n$, prove that the A-norm is a norm on $\mathbb{R}^n$.

1.3. Prove (1.13).

1.4. Let A, $B \in \mathbb{R}^{n\times n}$ and $C = AB$. Prove that the Frobenius norm is mutually consistent, i.e.,

$$\|A\,B\|_F \le \|A\|_F\,\|B\|_F \tag{1.29}$$

Hint: Use the definition $c_{ij} = \sum_k a_{ik}b_{kj}$ and the Cauchy-Schwarz inequality.

1.5. The **rank** of a matrix is the dimension of the space spanned by its columns. Prove that A has rank one if and only if $A = uv^T$ for some nonzero vectors u, $v \in \mathbb{R}^n$.

1.6. A matrix is **strictly upper triangular** if it is upper triangular with zero diagonal elements. Show that if $A \in \mathbb{R}^{n\times n}$ is strictly upper triangular, then $A^n = 0$.

1.7. Let $A = \mathrm{diag}_n(d_{-1}, d_0, d_1)$ denote the n-dimensional tri-diagonal matrix with $a_{i,i-k} = d_k$ for $k = -1, 0, 1$. For example,

$$\mathrm{diag}_4(-1,2,-1) = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}.$$

Let $B = \mathrm{diag}_{10}(-1,2,-1)$. Use a computer software to solve the following.

(a) Find condition number $\kappa_2(B)$.

(b) Find the smallest ($\lambda_{\min}$) and the largest eigenvalues ($\lambda_{\max}$) of B to compute the ratio $\dfrac{\lambda_{\max}}{\lambda_{\min}}$.

(c) Compare the above results.

CHAPTER 2

Gauss Elimination and Its Variants

One of the most frequently occurring problems in all areas of scientific endeavor is that of solving a system of n linear equations in n unknowns. The main subject of this chapter is to study the use of Gauss elimination to solve such systems. We will see that there are many ways to organize this fundamental algorithm.

Contents of Chapter 2

2.1. Systems of Linear Equations

Consider a system of n linear equations in n unknowns

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned} \tag{2.1}$$

Given the coefficients a_{ij} and the source b_i, we wish to find $x_1,\ x_2,\ \cdots,\ x_n$ which satisfy the equations.

Since it is tedious to write (2.1) again and again, we generally prefer to write it as a single matrix equation

$$Ax = b, \tag{2.2}$$

where

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \text{and} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

Solvers for Linear Systems

- Direct algebraic solvers
  - LU, LL^T, LDL^T, QR, SVD, SuperLU, $\cdots$
  - A is modified (factorized)
  - Harder to optimize and parallelize
  - Numerically robust, but higher algorithmic complexity
- Iterative algebraic solvers
  - Stationary and Nonstationary methods
    (Jacobi, Gauss-Seidel, SOR, SSOR,
    CG, MINRES, GMRES, BiCG, QMR, $\cdots$)
  - A is not changed (read-only)
  - Easier to optimize and parallelize
  - Low algorithmic complexity, but may not converge

2.1.1. Nonsingular matrices

Definition 2.1. *A square matrix $A \in \mathbb{R}^{n\times n}$ is* **invertible (nonsingular)** *if there is a matrix $B \in \mathbb{R}^{n\times n}$ such that $AB = BA = I$. The matrix B is called the* **inverse** *of A and denoted by A^{-1}.*

Theorem 2.2. (Invertible Matrix Theorem) *Let $A \in \mathbb{R}^{n\times n}$. Then the following are equivalent.*

1. *A^{-1} exists, i.e., A is invertible.*
2. *There is a matrix B such that $AB = I$*
3. *There is a matrix C such that $CA = I$*
4. *$Ay = 0$ implies that $y = 0$.*
5. *Given any vector b, there is exactly one vector x such that $Ax = b$.*
6. *The columns of A are linearly independent.*
7. *The rows of A are linearly independent.*
8. $\det(A) \neq 0$.
9. *Zero (0) is not an eigenvalue of A.*
10. *A is a product of elementary matrices.*

Example 2.3. *Let $A \in \mathbb{R}^{n\times n}$ and eigenvalues of A be λ_i, $i = 1, 2, \cdots, n$. Show that*

$$\det(A) = \prod_{i=1}^{n} \lambda_i. \tag{2.3}$$

Thus we conclude that A is singular if and only if 0 is an eigenvalue of A.

Hint: *Consider the characteristic polynomial of A, $\phi(\lambda) = \det(A - \lambda I)$, and $\phi(0)$.*

2.1.2. Numerical solutions of differential equations

Consider the following differential equation:

$$
\begin{array}{lrcll}
\text{(a)} & -u_{xx} + cu & = & f, & x \in (a_x, b_x), \\
\text{(b)} & -u_x + \beta u & = & g, & x = a_x, \\
\text{(c)} & u_x + \beta u & = & g, & x = b_x.
\end{array}
\tag{2.4}
$$

where

$$c \geq 0 \text{ and } \beta \geq 0 \quad (c + \beta > 0).$$

Select n_x equally spaced grid points on the interval $[a_x, b_x]$:

$$x_i = a_x + ih_x, \quad i = 0, 1, \cdots, n_x, \quad h_x = \frac{b_x - a_x}{n_x}.$$

Let $u_i = u(x_i)$. It follows from the Taylor's series expansion that

$$-u_{xx}(x_i) \;=\; \frac{-u_{i-1} + 2u_i - u_{i+1}}{h_x^2} + \frac{u_{xxxx}(x_i)}{12} h_x^2 + \cdots .$$

Thus the central second-order finite difference (FD) scheme for u_{xx} at x_i reads

$$-u_{xx}(x_i) \approx \frac{-u_{i-1} + 2u_i - u_{i+1}}{h_x^2}. \tag{2.5}$$

Apply the FD scheme for (2.4.a) to have

$$-u_{i-1} + (2 + h_x^2 c)u_i - u_{i+1} = h_x^2 f_i, \;\; i = 0, 1, \cdots, n_x. \tag{2.6}$$

However, we will meet ghost grid values at the end points. For example, at the point $a_x = x_0$, the formula becomes

$$-u_{-1} + (2 + h_x^2 c)u_0 - u_1 = h_x^2 f_0. \tag{2.7}$$

Here the value u_{-1} is not defined and we call it a **ghost grid value**.

Now, let's replace the value by using the boundary condition (2.4.b). The central FD scheme for u_x at x_0 can be formulated as

$$u_x(x_0) \approx \frac{u_1 - u_{-1}}{2h_x}, \;\; \text{Trunc.Err} = -\frac{u_{xxx}(x_0)}{6} h_x^2 + \cdots . \tag{2.8}$$

Thus the equation (2.4.b), $-u_x + \beta u = g$, can be approximated (at x_0)

$$u_{-1} + 2h_x \beta u_0 - u_1 = 2h_x g_0. \tag{2.9}$$

Hence it follows from (2.7) and (2.9) that

$$(2 + h_x^2 c + 2h_x \beta)u_0 - 2u_1 = h_x^2 f_0 + 2h_x g_0. \tag{2.10}$$

The same can be considered for the algebraic equation at the point x_n.

The problem (2.4) is reduced to finding the solution u satisfying

$$A\mathbf{u} = \mathbf{b}, \tag{2.11}$$

where $A \in \mathbb{R}^{(n_x+1)\times(n_x+1)}$,

$$A = \begin{bmatrix} 2+h_x^2c+2h_x\beta & -2 & & & \\ -1 & 2+h_x^2c & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2+h_x^2c & -1 \\ & & & -2 & 2+h_x^2c+2h_x\beta \end{bmatrix},$$

and

$$\mathbf{b} = \begin{bmatrix} h_x^2 f_0 \\ h_x^2 f_1 \\ \vdots \\ h_x^2 f_{n_x-1} \\ h_x^2 f_{n_x} \end{bmatrix} + \begin{bmatrix} 2h_x g_0 \\ 0 \\ \vdots \\ 0 \\ 2h_x g_{n_x} \end{bmatrix}.$$

Such a technique of removing ghost grid values is called **outer bordering**.

Dirichlet Boundary Condition: When the boundary values of the DE are known (Dirichlet boundary condition), the algebraic system does not have to include rows corresponding to the nodal points. However, it is more reusable if the algebraic system incorporates rows for all nodal points.

For example, consider

$$\begin{array}{lrcll} \text{(a)} & -u_{xx} + cu & = & f, & x \in (a_x, b_x), \\ \text{(b)} & -u_x + \beta u & = & g, & x = a_x, \\ \text{(c)} & u & = & u_d, & x = b_x. \end{array} \tag{2.12}$$

Then, the corresponding algebraic system can be formulated as

$$A'\,\mathbf{u} = \mathbf{b}', \tag{2.13}$$

where $A' \in \mathbb{R}^{(n_x+1)\times(n_x+1)}$,

$$A' = \begin{bmatrix} 2 + h_x^2 c + 2h_x\beta & -2 & & & \\ -1 & 2 + h_x^2 c & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 + h_x^2 c & -1 \\ & & & 0 & 1 \end{bmatrix},$$

and

$$\mathbf{b}' = \begin{bmatrix} h_x^2 f_0 \\ h_x^2 f_1 \\ \vdots \\ h_x^2 f_{n_x-1} \\ u_d \end{bmatrix} + \begin{bmatrix} 2h_x g_0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}.$$

Note: The system of linear equations (2.13) can be reformulated involving an $n_x \times n_x$ matrix

2.2. Triangular Systems

Definition 2.4.

1. *A matrix* $L = (\ell_{ij}) \in \mathbb{R}^{n\times n}$ *is* **lower triangular** *if*

$$\ell_{ij} = 0 \text{ whenever } i < j.$$

2. *A matrix* $U = (u_{ij}) \in \mathbb{R}^{n\times n}$ *is* **upper triangular** *if*

$$u_{ij} = 0 \text{ whenever } i > j.$$

Theorem 2.5. *Let G be a triangular matrix. Then G is nonsingular if and only if $g_{ii} \neq 0$ for $i = 1, \cdots, n$.*

2.2.1. Lower-triangular systems

Consider the $n \times n$ system

$$L\,y = b \tag{2.14}$$

where L is a nonsingular, lower-triangular matrix ($\ell_{ii} \neq 0$). It is easy to see how to solve this system if we write it in detail:

$$\begin{array}{lcl}
\ell_{11}\,y_1 & = & b_1 \\
\ell_{21}\,y_1 + \ell_{22}\,y_2 & = & b_2 \\
\ell_{31}\,y_1 + \ell_{32}\,y_2 + \ell_{33}\,y_3 & = & b_3 \\
\quad\vdots & & \vdots \\
\ell_{n1}\,y_1 + \ell_{n2}\,y_2 + \ell_{n3}\,y_3 + \cdots + \ell_{nn}\,y_n & = & b_n
\end{array} \tag{2.15}$$

The first equation involves only the unknown y_1, which can be found as

$$y_1 = b_1/\ell_{11}.$$

With y_1 just obtained, we can determine y_2 from the second equation:

$$y_2 = (b_2 - \ell_{21}\,y_1)/\ell_{22}.$$

Now with y_2 known, we can solve the third equation for y_3, and so on. In general, once we have $y_1,\, y_2,\, \cdots,\, y_{i-1}$, we can solve for y_i using the ith equation:

$$\begin{aligned}
y_i &= (b_i - \ell_{i1}\,y_1 - \ell_{i2}\,y_2 - \cdots - \ell_{i,i-1}\,y_{i-1})/\ell_{ii} \\
&= \frac{1}{\ell_{ii}}\Big(b_i - \sum_{j=1}^{i-1} \ell_{ij}\,y_j\Big)
\end{aligned} \tag{2.16}$$

Algorithm 2.6. (Forward Substitution)

```
for i=1:n
   for j=1:i-1
      b(i) = b(i)-L(i,j)*b(j)
   end                                        (2.17)
   if L(i,i)==0, set error flag, exit
   b(i) = b(i)/L(i,i)
end
```

The result is y.

Computational complexity: For each i, the forward substitution requires $2(i-1)+1$ flops. Thus the total number of flops becomes

$$\sum_{i=1}^{n}\{2(i-1)+1\} = \sum_{i=1}^{n}\{2i-1\} = n(n+1)-n = n^2. \tag{2.18}$$

2.2.2. Upper-triangular systems

Consider the system

$$U\,x = y \tag{2.19}$$

where $U = (u_{ij}) \in \mathbb{R}^{n\times n}$ is nonsingular, upper-triangular. Writing it out in detail, we get

$$\begin{array}{rcl} u_{11}\,x_1 + u_{12}\,x_2 + \cdots + u_{1,n-1}\,x_{n-1} + u_{1,n}\,x_n &=& y_1 \\ u_{22}\,x_2 + \cdots + u_{2,n-1}\,x_{n-1} + u_{2,n}\,x_n &=& y_2 \\ \vdots &=& \vdots \\ u_{n-1,n-1}\,x_{n-1} + u_{n-1,n}\,x_n &=& y_{n-1} \\ u_{n,n}\,x_n &=& y_n \end{array} \tag{2.20}$$

It is clear that we should solve the system from bottom to top.

$\boxed{\textbf{Algorithm}}$ **2.7. (Back Substitution)** *A Matlab code:*

```
for i=n:-1:1
   if(U(i,i)==0), error('U: singular!'); end
   x(i)=b(i)/U(i,i);                                   (2.21)
   b(1:i-1)=b(1:i-1)-U(1:i-1,i)*x(i);
end
```

Computational complexity: $n^2 + \mathcal{O}(n)$ flops.

2.3. Gauss Elimination

— a very basic algorithm for solving $A\mathbf{x} = \mathbf{b}$

The algorithms developed here produce (in the absence of rounding errors) the unique solution of $A\mathbf{x} = \mathbf{b}$ whenever $A \in \mathbb{R}^{n\times n}$ is nonsingular.

Our strategy: Transform the system $A\mathbf{x} = \mathbf{b}$ to a equivalent system $U\mathbf{x} = \mathbf{y}$, where U is upper-triangular.

It is convenient to represent $A\mathbf{x} = \mathbf{b}$ by an augmented matrix $[A|\mathbf{b}]$; each equation in $A\mathbf{x} = \mathbf{b}$ corresponds to a row of the augmented matrix.

Transformation of the system: By means of three **elementary row operations** applied on the augmented matrix.

Definition 2.8. Elementary row operations.

$$\begin{array}{ll} \textit{Replacement:} & R_i \leftarrow R_i + \alpha R_j \;\; (i \neq j) \\ \textit{Interchange:} & R_i \leftrightarrow R_j \\ \textit{Scaling:} & R_i \leftarrow \beta R_i \;\; (\beta \neq 0) \end{array} \tag{2.22}$$

Proposition 2.9.

1. *If* $[\hat{A}\,|\,\hat{\mathrm{b}}]$ *is obtained from* $[A\,|\,\mathrm{b}]$ *by elementary row operations (EROs), then systems* $[A\,|\,\mathrm{b}]$ *and* $[\hat{A}\,|\,\hat{\mathrm{b}}]$ *represent the same solution.*
2. *Suppose* $\hat{A}$ *is obtained from* A *by EROs. Then* $\hat{A}$ *is nonsingular if and only if* A *is.*
3. *Each ERO corresponds to left-multiple of an* **elementary matrix**.
4. *Each elementary matrix is nonsingular.*
5. *The elementary matrices corresponding to "Replacement" and "Scaling" operations are lower triangular.*

2.3.1. Gauss elimination without row interchanges

Let $E_1, E_2, \cdots, E_p$ be the p elementary matrices which transform A to an upper-triangular matrix U, that is,

$$E_p E_{p-1} \cdots E_2 E_1 \, A = U \tag{2.23}$$

Assume: No “Interchange" is necessary to apply. Then

- $E_p E_{p-1} \cdots E_2 E_1$ is lower-triangular and nonsingular
- $L \equiv (E_p E_{p-1} \cdots E_2 E_1)^{-1}$ is lower-triangular.
- In this case,

$$A = (E_p E_{p-1} \cdots E_2 E_1)^{-1} U = LU. \tag{2.24}$$

Example 2.10. *Let* $A = \begin{bmatrix} 2 & 1 & 1 \\ 2 & 2 & -1 \\ 4 & -1 & 6 \end{bmatrix}$ *and* $\mathbf{b} = \begin{bmatrix} 9 \\ 9 \\ 16 \end{bmatrix}$.

1. *Perform EROs to obtain an upper-triangular system.*
2. *Identity elementary matrices for the EROs.*
3. *What are the inverses of the elementary matrices?*
4. *Express* A *as* LU*.*

$\boxed{\textbf{Theorem}}$ **2.11.** (**LU Decomposition Theorem**) *The following are equivalent.*

1. *All leading principal submatrices of A are nonsingular. (The jth leading principal submatrix is $A(1:j, 1:j)$.)*
2. *There exists a unique unit lower triangular L and nonsingular upper-triangular U such that $A = LU$.*

Proof. **(2)** $\Rightarrow$ **(1)**: $A = LU$ may also be written

$$\begin{aligned}\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} &= \begin{bmatrix} L_{11} & \mathbf{0} \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ \mathbf{0} & U_{22} \end{bmatrix} \\ &= \begin{bmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{bmatrix} \end{aligned} \tag{2.25}$$

where A_{11} is a $j \times j$ leading principal submatrix. Thus

$$\det(A_{11}) = \det(L_{11}U_{11}) = 1 \cdot \det(U_{11}) = \prod_{k=1}^{j} (U_{11})_{kk} \neq 0.$$

Here we have used the assumption that U is nonsingular and so is U_{11}.

(1) $\Rightarrow$ **(2)**: It can be proved by induction on n. □

2.3.2. Solving linear systems by LU factorization

$$A = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ * & 1 & 0 & 0 \\ * & * & 1 & 0 \\ * & * & * & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} \blacksquare & * & * & * & * \\ 0 & \blacksquare & * & * & * \\ 0 & 0 & 0 & \blacksquare & * \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{U}$$

Let $A \in \mathbb{R}^{n\times n}$ be nonsingular. If $A = LU$, where L is a **unit lower triangular** matrix and U is an **upper triangular** matrix, then

$$A\mathbf{x} = \mathbf{b} \iff (LU)\mathbf{x} = L(U\mathbf{x}) = \mathbf{b} \iff \boxed{\left\{\begin{array}{l} L\mathbf{y} = \mathbf{b} \\ U\mathbf{x} = \mathbf{y} \end{array}\right.}$$

In the following couple of examples, LU is given.

Example 2.12. *Let* $A = \begin{bmatrix} 1 & 4 & -2 \\ 2 & 5 & -3 \\ -3 & -18 & 16 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} -12 \\ -14 \\ 64 \end{bmatrix}$.

$$A = LU \triangleq \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & -2 \\ 0 & -3 & 1 \\ 0 & 0 & 8 \end{bmatrix}$$

Use the LU factorization of A to solve $A\mathbf{x} = \mathbf{b}$.

Solution. Since

$$A\mathbf{x} = \mathbf{b} \Longleftrightarrow (LU)\mathbf{x} = L(U\mathbf{x}) = \mathbf{b} \Longleftrightarrow \boxed{\left\{ \begin{array}{l} L\mathbf{y} = \mathbf{b} \\ U\mathbf{x} = \mathbf{y} \end{array} \right.}$$

there are two steps:

(1) Solve $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$;
(2) Solve $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$.

(1) Solve $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$ by row reduction

$$[L \vdots \mathbf{b}] = \left[\begin{array}{ccccc} 1 & 0 & 0 & \vdots & -12 \\ 2 & 1 & 0 & \vdots & -14 \\ -3 & 2 & 1 & \vdots & 64 \end{array}\right] \to \left[\begin{array}{ccccc} 1 & 0 & 0 & \vdots & -12 \\ 0 & 1 & 0 & \vdots & 10 \\ 0 & 0 & 1 & \vdots & 8 \end{array}\right]$$

So $\mathbf{y} = \begin{bmatrix} -12 \\ 10 \\ 8 \end{bmatrix}$

(2) Solve $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$ by row reduction

$$[U \vdots \mathbf{y}] = \left[\begin{array}{ccccc} 1 & 4 & -2 & \vdots & -12 \\ 0 & -3 & 1 & \vdots & 10 \\ 0 & 0 & 8 & \vdots & 8 \end{array}\right] \to \left[\begin{array}{ccccc} 1 & 0 & 0 & \vdots & 2 \\ 0 & 1 & 0 & \vdots & -3 \\ 0 & 0 & 1 & \vdots & 1 \end{array}\right]$$

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

$\boxed{\textbf{Example}}$ **2.13.** *Let*

$$A = \begin{bmatrix} 5 & 4 & -2 & -3 \\ 15 & 13 & 2 & -10 \\ -5 & -1 & 28 & 3 \\ 10 & 10 & 8 & -8 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -10 \\ -29 \\ 30 \\ -22 \end{bmatrix}.$$

$$A = LU \triangleq \begin{bmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ -1 & 3 & 1 & 0 \\ 2 & 2 & -2 & 1 \end{bmatrix} \begin{bmatrix} 5 & 4 & -2 & -3 \\ 0 & 1 & 8 & -1 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 6 \end{bmatrix}$$

Use the LU factorization of A to solve $A\mathbf{x} = \mathbf{b}$.

Solution.

(1) Solve $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$

$$[L \vdots \mathbf{b}] = \left[\begin{array}{cccc:c} 1 & 0 & 0 & 0 & -10 \\ 3 & 1 & 0 & 0 & -29 \\ -1 & 3 & 1 & 0 & 30 \\ 2 & 2 & -2 & 1 & -22 \end{array}\right] \to \left[\begin{array}{cccc:c} 1 & 0 & 0 & 0 & -10 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 17 \\ 0 & 0 & 0 & 1 & 30 \end{array}\right]$$

So $\mathbf{y} = \begin{bmatrix} -10 \\ 1 \\ 17 \\ 30 \end{bmatrix}$

(2) Solve $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$

$$[U \vdots \mathbf{y}] = \left[\begin{array}{cccc:c} 5 & 4 & -2 & -3 & -10 \\ 0 & 1 & 8 & -1 & 1 \\ 0 & 0 & 2 & 3 & 17 \\ 0 & 0 & 0 & 6 & 30 \end{array}\right] \to \left[\begin{array}{cccc:c} 1 & 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 & -2 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 5 \end{array}\right]$$

So

$$\mathbf{x} = \begin{bmatrix} 3 \\ -2 \\ 1 \\ 5 \end{bmatrix}$$

LU Factorization Algorithm

- A is reduced to REF $\iff E_pE_{p-1}\cdots E_2E_1A = U$.
- $A = (E_pE_{p-1}\cdots E_2E_1)^{-1}U = LU$.
- $L = (E_pE_{p-1}\cdots E_2E_1)^{-1}$

How can we get L by row reduction?

$$E_pE_{p-1}\cdots E_2E_1L = I \iff L \xrightarrow{\text{row replacement}} I$$

$$E_pE_{p-1}\cdots E_2E_1A = U \iff A \xrightarrow{\text{row replacement}} U$$

The row operation sequences are the same!

Algorithm for an LU Factorization:

$$\boxed{\begin{array}{l} A \xrightarrow{\text{row replacement}} U \text{ (REF)} \\ L \xrightarrow{\text{row replacement}} I \end{array}}$$

Remark 2.14.

- *To get L, theoretically we reduce I to L ($I \to L$) by using the row operations used in $A \to U$ in* **BOTH** *reverse order and reverse operation* **(different from finding A^{-1})**.
- *If the size of $A \in \mathbb{R}^{m\times n}$ and $A = LU$, then L must be an $m \times m$* **unit lower triangular** *matrix, and U is the REF of A by using* **only row replacement** *operations. $A \sim U$. The size of U is $m \times n$.*
- *To reduce the round-off errors, practical implementations also use* **partial pivoting**, *where* **interchange operations** *may be used.*
- *In practice, to save storage, we can store L and U to the array of A.*

$$\underbrace{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ * & 1 & 0 & 0 \\ * & * & 1 & 0 \\ * & * & * & 1 \end{bmatrix}}_{\mathbf{L}} \underbrace{\begin{bmatrix} \blacksquare & * & * & * & * \\ 0 & \blacksquare & * & * & * \\ 0 & 0 & 0 & \blacksquare & * \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{U}}$$

Example 2.15. *Find the LU factorization of* $A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix}$.

Solution.

$$A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix} \xrightarrow[R_3 \leftarrow R_3 + \mathbf{2}R_1]{R_2 \leftarrow R_2 - \mathbf{3}R_1} \begin{bmatrix} 3 & -1 & 1 \\ 0 & 4 & -1 \\ 0 & 3 & -3 \end{bmatrix}$$

$$\xrightarrow{R_3 \leftarrow R_3 - \mathbf{\frac{3}{4}}R_2} \begin{bmatrix} 3 & -1 & 1 \\ 0 & 4 & -1 \\ 0 & 0 & -\frac{9}{4} \end{bmatrix} = U$$

$$\begin{aligned} A \to U: \quad & R_2 \leftarrow R_2 - \mathbf{3}R_1 \Longrightarrow R_3 \leftarrow R_3 + \mathbf{2}R_1 \Longrightarrow R_3 \leftarrow R_3 - \mathbf{\tfrac{3}{4}}R_2 \\ & E_3E_2E_1A = U \Longrightarrow A = (E_3E_2E_1)^{-1}U \\ & L = (E_3E_2E_1)^{-1}I = E_1^{-1}E_2^{-1}E_3^{-1}I \\ & E_1^{-1}E_2^{-1}E_3^{-1}I = L \Longrightarrow E_3E_2E_1L = I \\ I \to L: \quad & R_3 \leftarrow R_3 + \mathbf{\tfrac{3}{4}}R_2 \Longrightarrow R_3 \leftarrow R_3 - \mathbf{2}R_1 \Longrightarrow R_2 \leftarrow R_2 + \mathbf{3}R_1 \end{aligned}$$

Each step happens “simultaneously"

$$\begin{aligned} A \to U: \quad & R_2 \leftarrow R_2 - \mathbf{3}R_1 \Longrightarrow R_3 \leftarrow R_3 + \mathbf{2}R_1 \Longrightarrow R_3 \leftarrow R_3 - \mathbf{\tfrac{3}{4}}R_2 \\ & A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix} \to \begin{bmatrix} 3 & -1 & 1 \\ 0 & 4 & -1 \\ 0 & 0 & -\frac{9}{4} \end{bmatrix} = U \\ I \to L: \quad & R_3 \leftarrow R_3 + \mathbf{\tfrac{3}{4}}R_2 \Longrightarrow R_3 \leftarrow R_3 - \mathbf{2}R_1 \Longrightarrow R_2 \leftarrow R_2 + \mathbf{3}R_1 \end{aligned}$$

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \xrightarrow{R_3 \leftarrow R_3 + \mathbf{\frac{3}{4}}R_2} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{3}{4} & 1 \end{bmatrix}$$

$$\xrightarrow[R_3 \leftarrow R_3 - \mathbf{2}R_1]{R_2 \leftarrow R_2 + \mathbf{3}R_1} \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ -2 & \frac{3}{4} & 1 \end{bmatrix} = L$$

Example 2.16. *Find the LU factorization of* $A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix}$.

Solution. (Practical Implementation):

$$A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix} \xrightarrow[R_3 \leftarrow R_3 + 2R_1]{R_2 \leftarrow R_2 - 3R_1} \begin{bmatrix} \mathbf{3} & -1 & \mathbf{1} \\ \hline \mathbf{3} & \mathbf{4} & -1 \\ \mathbf{-2} & \mathbf{3} & -3 \end{bmatrix}$$

$$\xrightarrow{R_3 \leftarrow R_3 - \frac{3}{4} R_2} \begin{bmatrix} \mathbf{3} & -1 & \mathbf{1} \\ \hline \mathbf{3} & \mathbf{4} & -1 \\ \mathbf{-2} & \mathbf{\frac{3}{4}} & -\frac{9}{4} \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \mathbf{3} & 1 & 0 \\ -2 & \mathbf{\frac{3}{4}} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 3 & -1 & 1 \\ 0 & 4 & -1 \\ 0 & 0 & -\frac{9}{4} \end{bmatrix}.$$

Note: it is easy to verify that $A = LU$.

Example **2.17.** *Find the LU factorization of*

$$A = \begin{bmatrix} 2 & -1 \\ 6 & 5 \\ -10 & 3 \\ 12 & -2 \end{bmatrix}.$$

Solution.

$$\begin{bmatrix} 2 & -1 \\ 6 & 5 \\ -10 & 3 \\ 12 & -2 \end{bmatrix} \xrightarrow[\substack{R_3 \leftarrow R_3 + 5R_1 \\ R_4 \leftarrow R_4 - 6R_1}]{R_2 \leftarrow R_2 - 3R_1} \left[\begin{array}{c|c} \mathbf{2} & -1 \\ \hline \mathbf{3} & 8 \\ -5 & -2 \\ \mathbf{6} & 4 \end{array}\right] \xrightarrow[R_4 \leftarrow R_4 - \frac{1}{2}R_2]{R_3 \leftarrow R_3 + \frac{1}{4}R_2} \begin{bmatrix} \mathbf{2} & -1 \\ \mathbf{3} & 8 \\ -5 & -\frac{1}{4} \\ \mathbf{6} & \frac{1}{2} \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \mathbf{3} & 1 & 0 & 0 \\ \mathbf{-5} & -\frac{1}{4} & 1 & 0 \\ \mathbf{6} & \frac{1}{2} & 0 & 1 \end{bmatrix}_{4\times 4}, \quad U = \begin{bmatrix} 2 & -1 \\ 0 & 8 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}_{4\times 2}.$$

Note: U has the same size as A, that is, its size is 4×2. L is a square matrix and is a unit lower triangular matrix.

Numerical Notes: For an $n \times n$ dense matrix A (with most entries nonzero) with n moderately large.

- Computing an LU factorization of A takes about $2n^3/3$ flops[†] ($\sim$ row reducing $[A\ \mathrm{b}]$), while finding A^{-1} requires about $2n^3$ flops.
- Solving $L\mathrm{y} = \mathrm{b}$ and $U\mathrm{x} = \mathrm{y}$ requires about $2n^2$ flops, because any $n \times n$ triangular system can be solved in about n^2 flops.
- Multiplying b by A^{-1} also requires about $2n^2$ flops, but the result may not as accurate as that obtained from L and U (due to round-off errors in computing A^{-1} & $A^{-1}\mathrm{b}$).
- If A is sparse (with mostly zero entries), then L and U may be sparse, too. On the other hand, A^{-1} is likely to be dense. In this case, a solution of $A\mathrm{x} = \mathrm{b}$ with LU factorization is much faster than using A^{-1}.

[†] A **flop** is $+$, $-$, $\times$ or $\div$.

Further Comments: LU Algorithms with row interchange operations

1. To reduce the round-off errors.

$$\underbrace{\begin{bmatrix} 3 & -1 & 1 \\ \boxed{9} & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix}}_{A} \to \begin{bmatrix} \boxed{9} & 1 & 2 \\ 3 & -1 & 1 \\ -6 & 5 & -5 \end{bmatrix} \to \cdots \to \underbrace{\begin{bmatrix} \boxed{9} & 1 & 2 \\ 0 & \boxed{\frac{17}{3}} & -\frac{11}{3} \\ 0 & 0 & \boxed{-\frac{9}{17}} \end{bmatrix}}_{U}$$

$$P*A = L*U, \quad P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, L = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{2}{3} & 1 & 0 \\ \frac{1}{3} & -\frac{4}{17} & 1 \end{bmatrix}$$

For partial pivoting, a pivot in a pivot column is chosen from the entries with largest magnitude.

2. **Pivoting is needed if the pivot position entry is 0**

$$\begin{bmatrix} 0 & 2 & 5 \\ 1 & 3 & 0 \\ 6 & 2 & -7 \end{bmatrix} \xrightarrow{R_1 \leftrightarrow R_3} \begin{bmatrix} \boxed{6} & 2 & -7 \\ 1 & 3 & 0 \\ 0 & 2 & 5 \end{bmatrix}$$

MATLAB: $[L, U] = lu(A)$ stores an upper triangular matrix in U and a “psychologically lower triangular matrix" (i.e. a product of lower triangular and permutation matrices) in L, so that $A = L*U$. A can be rectangular. $[L, U, P] = lu(A)$ returns unit lower triangular matrix L, upper triangular matrix U, and permutation matrix P so that $P*A = L*U$.

2.3.3. Gauss elimination with pivoting

Definition 2.18. *A* **permutation matrix** *is a matrix that has exactly one* 1 *in each row ans in each column, all other entries being zero.*

Example 2.19. *Show that if* P *is permutation matrix, then* $P^T P = PP^T = I$. *Thus* P *is nonsingular and*

$$P^{-1} = P^T.$$

Proof. (Self study)

Lemma 2.20. *Let* P *and* Q *be* $n \times n$ **permutation matrices** *and* $A \in \mathbb{R}^{n\times n}$. *Then*

1. PA — A *with its rows permuted*
   AP — A *with its columns permuted.*
2. $\det(P) = \pm 1$.
3. PQ *is also a permutation matrix.*

Example 2.21. *Let $A \in \mathbb{R}^{n\times n}$, and let $\widehat{A}$ be a matrix obtained from scrambling the rows. Show that there is a unique permutation matrix $P \in \mathbb{R}^{n\times n}$ such that $\widehat{A} = PA$.*

Hint: Consider the row indices in the scrambled matrix $\widehat{A}$, say $\{k_1,\, k_2,\, \cdots,\, k_n\}$. (This means that for example, the first row of $\widehat{A}$ is the same as the k_1-th row of A.) Use the index set to define a permutation matrix P.

Proof. (Self study)

$\boxed{\textbf{Theorem}}$ **2.22.** *Gauss elimination with partial pivoting, applied to $A \in \mathbb{R}^{n\times n}$, produces a unit lower-triangular matrix L with $|\ell_{ij}| \le 1$, an upper-triangular matrix U, and a permutation matrix P such that*

$$\widehat{A} = PA = LU$$

or, equivalently,

$$A = P^T LU \tag{2.26}$$

Note: If A is singular, then so is U.

$\boxed{\textbf{Algorithm}}$ **2.23.** *Solving $A\mathbf{x} = \mathbf{b}$ using GE.*

1. *Factorize A into $A = P^T LU$, where*
   - P = *permutation matrix,*
   - L = *unit lower triangular matrix (i.e., with ones on the diagonal),*
   - U = *nonsingular upper-triangular matrix.*
2. *Solve $P^T LU\mathbf{x} = \mathbf{b}$*
   - *(a)* $LU\mathbf{x} = P\mathbf{b}$ *(permuting* $\mathbf{b}$*)*
   - *(b)* $U\mathbf{x} = L^{-1}(P\mathbf{b})$ *(forward substitution)*
   - *(c)* $\mathbf{x} = U^{-1}(L^{-1}P\mathbf{b})$ *(back substitution)*

In practice:

$$\left.\begin{array}{rl} A\mathbf{x} = \mathbf{b} \iff & P^T(LU)\mathbf{x} = \mathbf{b} \\ \iff & L(U\mathbf{x}) = P\mathbf{b} \end{array}\right\} \iff \boxed{\left\{\begin{array}{l} L\mathbf{y} = P\mathbf{b} \\ U\mathbf{x} = \mathbf{y} \end{array}\right.}$$

Theorem 2.24. *If A is nonsingular, then there exist permutations P_1 and P_2, a unit lower triangular matrix L, and a nonsingular upper-triangular matrix U such that*

$$P_1AP_2 = LU.$$

Only one of P_1 and P_2 is necessary.

Remark 2.25. *P_1A reorders the rows of A, AP_2 reorders the columns, and P_1AP_2 reorders both. Consider*

$$\begin{aligned} P_1'AP_2' &= \begin{bmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0} \\ L_{21} & I \end{bmatrix} \begin{bmatrix} u_{11} & U_{12} \\ \mathbf{0} & \tilde{A}_{22} \end{bmatrix} \\ &= \begin{bmatrix} u_{11} & U_{12} \\ L_{21}u_{11} & L_{21}U_{12} + \tilde{A}_{22} \end{bmatrix} \end{aligned} \tag{2.27}$$

*We can choose $P_2' = I$ and P_1' so that a_{11} is the largest entry in absolute value in its column, which implies $L_{21} = \frac{A_{21}}{a_{11}}$ has entries bounded by 1 in modulus. More generally, at step k of Gaussian elimination, where we are computing the kth column of L, we reorder the rows so that the largest entry in the column is on the pivot. This is called "***Gaussian elimination with partial pivoting***", or* **GEPP** *for short. GEPP guarantees that all entries of L are bounded by one in modulus.*

Remark 2.26. *We can choose P_1 and P_2 so that a_{11} in (2.27) is the largest entry in modulus in the whole matrix. More generally, at step k of Gaussian elimination, we reorder the rows and columns so that the largest entry in the matrix is on the pivot. This is called "***Gaussian elimination with complete pivoting***", or* **GECP** *for short.*

Algorithm 2.27. *Factorization with pivoting*

```
for $i = 1$ to $n - 1$
   apply permutations so $a_{ii} \neq 0$
   /* compute column $i$ of $L$ ($L_{21}$ in (2.27))*/
   for $j = i + 1$ to $n$
      $l_{ji} = a_{ji}/a_{ii}$
   end for
   /* compute row $i$ of $U$ ($U_{12}$ in (2.27) */
   for $j = i$ to $n$
      $u_{ij} = a_{ij}$
   end for
   /* update $A_{22}$ (to get $\tilde{A}_{22} = A_{22} - L_{21}U_{12}$ in (2.27) */
   for $j = i + 1$ to $n$
      for $k = i + 1$ to $n$
         $a_{jk} = a_{jk} - l_{ji} * u_{ik}$
      end for
   end for
end for
```

Algorithm 2.28. *LU factorization with pivoting, overwriting L and U on A:*

```
for $i = 1$ to $n-1$
    apply permutations
    for $j = i+1$ to $n$
        $a_{ji} = a_{ji}/a_{ii}$
    end for
    for $j = i+1$ to $n$
        for $k = i+1$ to $n$
            $a_{jk} = a_{jk} - a_{ji}/a_{ik}$
        end for
    end for
end for
```

Computational Cost: The flop count of LU is done by replacing loops by summations over the same range, and inner loops by their flop counts:

$$\begin{aligned} \sum_{i=1}^{n-1}\Big(\sum_{j=i+1}^{n} 1 + \sum_{j=i+1}^{n}\sum_{k=i+1}^{n} 2\Big) &= \sum_{i=1}^{n-1}\left[(n-i)+2(n-i)^2\right] \\ &= \frac{2n^3}{3} + \mathcal{O}(n^2) \end{aligned} \tag{2.28}$$

The forward and back substitutions with L and U to complete the solution of $Ax = b$ cost $\mathcal{O}(n^2)$.

Thus overall solving $Ax = b$ with Gaussian elimination costs $\boldsymbol{\frac{2n^3}{3} + \mathcal{O}(n^2)}$ flops.

Example 2.29. *Find the* LU *factorization of* $A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix}$

Solution. (Without pivoting)

$$A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix} \xrightarrow[R_3 \leftarrow R_3 + 2R_1]{R_2 \leftarrow R_2 - 3R_1} \begin{bmatrix} \mathbf{3} & -1 & \mathbf{1} \\ \mathbf{3} & \mathbf{4} & -1 \\ \mathbf{-2} & \mathbf{3} & -3 \end{bmatrix}$$

$$\xrightarrow{R_3 \leftarrow R_3 - \frac{3}{4}R_2} \begin{bmatrix} \mathbf{3} & -1 & \mathbf{1} \\ \mathbf{3} & \mathbf{4} & -1 \\ \mathbf{-2} & \mathbf{\frac{3}{4}} & -\frac{9}{4} \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \mathbf{3} & 1 & 0 \\ -2 & \mathbf{\frac{3}{4}} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 3 & -1 & 1 \\ 0 & 4 & -1 \\ 0 & 0 & -\frac{9}{4} \end{bmatrix}.$$

(With partial pivoting)

$$A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix} \xrightarrow{\mathbf{R_1 \leftrightarrow R_2}} \begin{bmatrix} \boxed{9} & 1 & 2 \\ 3 & -1 & 1 \\ -6 & 5 & -5 \end{bmatrix}$$

$$\xrightarrow[R_3 \leftarrow R_3 + \frac{2}{3}R_1]{R_2 \leftarrow R_2 - \frac{1}{3}R_1} \begin{bmatrix} \boxed{9} & 1 & 2 \\ \mathbf{\frac{1}{3}} & -\frac{4}{3} & \frac{1}{3} \\ -\mathbf{\frac{2}{3}} & \frac{17}{3} & -\frac{11}{3} \end{bmatrix} \xrightarrow{\mathbf{R_2 \leftrightarrow R_3}} \begin{bmatrix} \boxed{9} & 1 & 2 \\ -\mathbf{\frac{2}{3}} & \boxed{\frac{17}{3}} & -\frac{11}{3} \\ \mathbf{\frac{1}{3}} & -\frac{4}{3} & \frac{1}{3} \end{bmatrix}$$

$$\xrightarrow{R_3 \leftarrow R_3 + \frac{4}{17}R_2} \begin{bmatrix} \boxed{9} & 1 & 2 \\ -\mathbf{\frac{2}{3}} & \boxed{\frac{17}{3}} & -\frac{11}{3} \\ \mathbf{\frac{1}{3}} & -\mathbf{\frac{4}{17}} & -\frac{9}{17} \end{bmatrix}, \quad I \xrightarrow{\mathbf{R_1 \leftrightarrow R_2}} E \xrightarrow{\mathbf{R_2 \leftrightarrow R_3}} P$$

$PA = LU$

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 & 0 \\ -\mathbf{\frac{2}{3}} & 1 & 0 \\ \mathbf{\frac{1}{3}} & -\mathbf{\frac{4}{17}} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} \boxed{9} & 1 & 2 \\ 0 & \boxed{\frac{17}{3}} & -\frac{11}{3} \\ 0 & 0 & \boxed{-\frac{9}{17}} \end{bmatrix}$$

2.3.4. Calculating A^{-1}

The program to solve $Ax = b$ can be used to calculate the inverse of a matrix. Letting $X = A^{-1}$, we have $AX = I$. This equation can be written in partitioned form:

$$A[x_1\, x_2\, \cdots\, x_n] = [e_1\, e_2\, \cdots\, e_n] \tag{2.29}$$

where $x_1,\, x_2,\, \cdots,\, x_n$ and $e_1,\, e_2,\, \cdots,\, e_n$ are columns of X and I, respectively. Thus $AX = I$ is equivalent to the n equations

$$Ax_i = e_i, \quad i = 1, 2, \cdots, n. \tag{2.30}$$

Solving these n systems by Gauss elimination with partial pivoting, we obtain A^{-1}.

Computational complexity

A naive flop count:

LU-factorization of A:	$\frac{2}{3}n^3 + \mathcal{O}(n^2)$
Solve for n equations in (2.30):	$n \cdot 2n^2 = 2n^3$
Total cost:	$\frac{8}{3}n^3 + \mathcal{O}(n^2)$

A modification: The forward-substitution phase requires the solution of

$$Ly_i = e_i, \quad i = 1, 2, \cdots, n. \tag{2.31}$$

Some operations can be saved by exploiting the leading zeros in e_i. (For each i, the portion of L to be accessed is triangular.) With these savings, one can conclude that A^{-1} can be computed in $2n^3 + \mathcal{O}(n^2)$ flops; see Homework 4.

2.4. Special Linear Systems

2.4.1. Symmetric positive definite (SPD) matrices

Definition 2.30. *A real matrix is* **symmetric positive definite** *(s.p.d.) if $A = A^T$ and $\mathbf{x}^T A\mathbf{x} > 0$, $\forall \mathbf{x} \neq 0$.*

Proposition 2.31. *Let $A \in \mathbb{R}^{n\times n}$ be a real matrix.*

1. *A is s.p.d. if and only if $A = A^T$ and all its eigenvalues are positive.*
2. *If A is s.p.d and H is any principal submatrix of A ($H = A(j:k, j:k)$ for some $j \le k$), then H is s.p.d.*
3. *If X is nonsingular, then A is s.p.d. if and only if X^TAX is s.p.d.*
4. *If A is s.p.d., then all $a_{ii} > 0$, and $\max_{ij} |a_{ij}| = \max_i a_{ii} > 0$.*
5. *A is s.p.d. if and only if there is a unique lower triangular nonsingular matrix L, with positive diagonal entries, such that $A = LL^T$.*

The decomposition $A = LL^T$ is called the **Cholesky factorization** *of A, and L is called the Cholesky factor of A.*

Algorithm 2.32. *Cholesky algorithm:*

```
for $j = 1$ to $n$
    $l_{jj} = \Big(a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2\Big)^{\frac{1}{2}}$
    for $i = j+1$ to $n$
        $l_{ij} = \Big(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}\Big)/l_{jj}$
    end for
end for
```

Derivation: See Homework 2.

Remark 2.33. *The Cholesky factorization is mainly used for the numerical solution of linear systems $Ax = b$.*

- *For symmetric linear systems, the Cholesky decomposition (or its LDL^T variant) is the method of choice, for superior efficiency and numerical stability.*
- *Compared with the LU-decomposition, it is roughly twice as efficient ($\mathcal{O}(n^3/3)$ flops).*

Algorithm 2.34.

(Cholesky: Gaxpy Version) [10, p.144]

```
for $j = 1 : n$
    if $j > 1$
        $A(j : n, j) = A(j : n, j)$
            $-A(j : n, 1 : j-1)A(j, 1 : j-1)^T$
    end
    $A(j : n, j) = A(j : n, j)/\sqrt{A(j, j)}$
end
```

(Cholesky: Outer Product Version)

```
for $k = 1 : n$
    $A(k, k) = \sqrt{A(k, k)}$
    $A(k+1 : n, k) = A(k+1 : n, k)/A(k, k)$
    for $j = k+1 : n$
        $A(j : n, j) = A(j : n, j) - A(j : n, k)A(j, k)$
    end
end
```

- Total cost of the Cholesky algorithms: $\mathcal{O}(n^3/3)$ flops.
- ***L*** overwrites the lower triangle of A in the above algorithms.

Stability of Cholesky Process

- $$A = LL^T \Longrightarrow l_{ij}^2 \le \sum_{k=1}^{i} l_{ik}^2 = a_{ii}, \quad ||L||_2^2 = ||A||_2$$
- If $\hat{\mathrm{x}}$ is the computed solution to $A\mathrm{x} = \mathrm{b}$, obtained via any of Cholesky procedures, then $\hat{\mathrm{x}}$ solves the perturbed system $(A+E)\hat{\mathrm{x}} = \mathrm{b}$
- **Example:** [10] If Cholesky algorithm is applied to the s.p.d. matrix

  $$A = \begin{bmatrix} 100 & 15 & .01 \\ 15 & 2.26 & .01 \\ .01 & .01 & 1.00 \end{bmatrix}$$

  For round arithmetic, we get

  $$\hat{l}_{11} = 10, \quad \hat{l}_{21} = 1.5, \quad \hat{l}_{31} = .001,$$

  and

  $$\hat{l}_{22} = (a_{22} - \hat{l}_{21}{}^2)^{1/2} = .00$$

  The algorithm then breaks down trying to compute l_{32}.

  By the way: The Cholesky factor of A is

  $$L = \begin{bmatrix} 10.0000 & 0 & 0 \\ 1.5000 & 0.1000 & 0 \\ 0.0010 & 0.0850 & 0.9964 \end{bmatrix}$$

Symmetric Positive Semidefinite Matrices

- A matrix is said to be **symmetric positive semidefinite** (SPS) if $A = A^T$ and $\mathbf{x}^T A\mathbf{x} \ge 0 \quad$ for all $\mathbf{x} \in \mathbb{R}^n$.
- **Cholesky Algorithm** for SPS matrices

```
for $k = 1 : n$
    if $A(k,k) > 0$
        $A(k,k) = \sqrt{A(k,k)}$
        $A(k+1:n,k) = A(k+1:n,k)/A(k,k)$
        for $j = k+1 : n$
            $A(j:n,j) = A(j:n,j) - A(j:n,k)A(j,k)$
        end
    end
end
```

- It breaks down if $A(k,k) \le 0$.

2.4.2. LDL^T and Cholesky factorizations

- LDM^T **Factorization:** If all the leading principal submatrices of $A \in \mathbb{R}^{n\times n}$ are nonsingular, then there exist unique unit lower triangular matrices L and M, and a unique diagonal matrix $D = \mathrm{diag}(d_1, \cdots, d_n)$, such that $A = LDM^T$.
- **Symmetric** LDM^T **Factorization:** If A is a nonsingular symmetric matrix and has the factorization $A = LDM^T$, where L and M are unit lower triangular matrices and D is a diagonal matrix, then $L = M$.
- LDL^T **Algorithm:** If $A \in \mathbb{R}^{n\times n}$ is symmetric and has an LU factorization, then this algorithm computes a unit lower triangular matrix L and a diagonal matrix $D = \mathrm{diag}(d_1, \cdots, d_n)$, so $A = LDL^T$. The entry a_{ij} is overwritten with l_{ij} if $i > j$ and with d_i if $i = j$.
- **Total cost of** LDL^T: $\mathcal{O}(n^3/3)$ flops, about half the number of flops involved in GE.

Remark 2.35. *Let $A = L_0 L_0^T$ be the Cholesky factorization. Then the Cholesky factor L_0 can be decomposed as*

$$L_0 = L D_0, \tag{2.32}$$

where L is the unit Cholesky factor and D_0 is diagonal:

$$L(:,j) = L_0(:,j)/L_0(j,j) \ \textit{ and } \ D_0 = \mathrm{diag}(L_0)$$

Then, for the LDL^T variant, we set $D = D_0 D_0^T = D_0{}^2$.

LDL^T Factorization

To implement the LDL^T algorithm, define $\mathbf{v}(1:j)$ by the first j-th components of $DL^T\mathbf{e}_j$ (j-th column of DL^T or j-th row of LD). Then

$$\mathbf{v}(1:j) = \begin{bmatrix} \mathbf{d}(1)L(j,1) \\ \mathbf{d}(2)L(j,2)\vdots \\ \mathbf{d}(j-1)L(j,j-1) \\ \mathbf{d}(j) \end{bmatrix}$$

Note that $\mathbf{v} = A(j,j) - L(j,1:j-1)\mathbf{v}(1:j-1)$ can be derived from the j-th equation in

$$L(1:j,1:j)\mathbf{v} = A(1:j,j).$$

$$\underbrace{\begin{bmatrix} a_{11} & a_{21} & a_{31} & a_{41} \\ a_{21} & a_{22} & a_{32} & a_{42} \\ a_{31} & a_{32} & a_{33} & a_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ l_{41} & l_{42} & l_{43} & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} d_1 & d_1 l_{21} & d_1 l_{31} & d_1 l_{41} \\ 0 & d_2 & d_2 l_{32} & d_2 l_{42} \\ 0 & 0 & d_3 & d_3 l_{43} \\ 0 & 0 & 0 & d_4 \end{bmatrix}}_{DL^T}$$

Algorithm 2.36. **(LDL^T Algorithm)**

```
for $j = 1 : n$
    %% Compute $v(1 : j)$
    for $i = 1 : j - 1$
        $v(i) = A(j, i)A(i, i)$
    end
    $v(j) = A(j, j) - A(j, 1 : j - 1)v(1 : j - 1);$
    %% Store $d(j)$ and compute $L(j + 1 : n, j)$
    $A(j, j) = v(j)$
    $A(j + 1 : n, j) = (A(j + 1 : n, j)$
                $-A(j + 1 : n, 1 : j - 1)v(1 : j - 1))/v(j)$
end
```

Example 2.37.

$$A = \begin{bmatrix} 2 & 6 & 8 \\ 6 & 23 & 34 \\ 8 & 34 & 56 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \mathbf{3} & 1 & 0 \\ \mathbf{4} & \mathbf{2} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{2} & 0 & 0 \\ 0 & \mathbf{5} & 0 \\ 0 & 0 & \mathbf{4} \end{bmatrix} \begin{bmatrix} 1 & \mathbf{3} & \mathbf{4} \\ 0 & 1 & \mathbf{2} \\ 0 & 0 & 1 \end{bmatrix}$$

So, when LDL^T decomposition is applied, A can be overwritten as

$$A = \begin{bmatrix} \mathbf{2} & 6 & 8 \\ \mathbf{3} & \mathbf{5} & 34 \\ \mathbf{4} & \mathbf{2} & \mathbf{4} \end{bmatrix}$$

Recall the theorem: If $A \in \mathbb{R}^{n\times n}$ is s.p.d., then there exists a unique lower triangular matrix $L \in \mathbb{R}^{n\times n}$ with positive diagonal entries such that $A = LL^T$.

The Cholesky factorization can be easily transformed to the LDL^T form; see Remark 2.35 on page 63.

Example 2.38. (From Cholesky to LDL^T)

$$\begin{aligned}
\begin{bmatrix} 2 & -2 \\ -2 & 5 \end{bmatrix} &= \begin{bmatrix} \sqrt{2} & 0 \\ -\sqrt{2} & \sqrt{3} \end{bmatrix} \begin{bmatrix} \sqrt{2} & -\sqrt{2} \\ 0 & \sqrt{3} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{2} & 0 \\ 0 & \sqrt{3} \end{bmatrix} \begin{bmatrix} \sqrt{2} & 0 \\ 0 & \sqrt{3} \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}
\end{aligned}$$

Matlab:

- “chol": Cholesky factorization
- “ldl": LDL^T factorization

2.4.3. M-matrices and Stieltjes matrices

Definition 2.39.

1. *A matrix $A = [a_{ij}] \in \mathbb{R}^{n\times n}$ with $a_{ij} \le 0$ for all $i \ne j$ is an* **M-matrix** *if A is nonsingular and $A^{-1} \ge O$.*
2. *A matrix $A = [a_{ij}] \in \mathbb{R}^{n\times n}$ with $a_{ij} \le 0$ for all $i \ne j$ is a* **Stieltjes matrix** *if A is symmetric and positive definite.*

Remark 2.40.

- *A Stieltjes matrix is an M-matrix.*
- *In the simulation of PDEs, many applications involve M-matrices.*
- *Consider an algebraic system: $A\mathrm{x} = \mathrm{b}$ with $A^{-1} \ge 0$. Then the solution*

$$\mathbf{x} = A^{-1}\mathbf{b}$$

must be nonnetative for all nonnegative sources b.

We will deal with these matrices in detail, when we study iterative methods for linear algebraic systems.

Exercises for Chapter 2

2.1. Consider the finite difference method on uniform meshes to solve

$$\begin{array}{ll}\text{(a)} & -u_{xx} + u = (\pi^2+1)\cos(\pi x), \;\; x \in (0,1), \\ \text{(b)} & u(0) = 1 \;\text{ and }\; u_x(1) = 0.\end{array} \tag{2.33}$$

(a) Implement a function to construct algebraic systems in the full matrix form, for general $n_x \ge 1$.
(b) Use a direct method (e.g., $A\backslash b$) to find approximate solutions for $n_x = 25,\ 50,\ 100$.
(c) The actual solution for (2.33) is $u(x) = \cos(\pi x)$. Measure the maximum errors for the approximate solutions.
(d) Now, save the coefficient matrices exploring the Matlab built-in functions `sparse` and/or `spdiags`; see, for example, `matlabtutorials.com/howto/spdiags`.
(e) Then, solve the linear systems and compare the elapsed times with those of the full-matrix solve. (You may check the elapsed time using `tic ... toc`.

2.2. Derive the details of the Cholesky algorithm 2.32 on page 59.
Hint: You may first set

$$L = \begin{bmatrix} \ell_{11} & 0 & 0 & \cdots & 0 \\ \ell_{21} & \ell_{22} & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ \ell_{n1} & \ell_{n2} & & \cdots & \ell_{nn} \end{bmatrix} \tag{2.34}$$

and then try to compare entries of LL^T with those of A.

2.3. Let $L = [\ell_{ij}]$ and $M = [m_{ij}]$ be lower-triangular matrices.

(a) Prove that LM is lower triangular.
(b) Prove that the entries of the main diagonal of LM are

$$\ell_{11}m_{11},\ \ell_{22}m_{22},\ \cdots,\ \ell_{nn}m_{nn}$$

Thus the product of two unit lower-triangular matrices is unit lower triangular.

2.4. Consider the savings explained with equation (2.31) for the computation of A^{-1}. Taking those savings into account, show that A^{-1} can be computed in $2n^3 + \mathcal{O}(n^2)$ flops.

2.5. Use LU decomposition obtained with partial pivoting to solve the system $A\mathrm{x} = \mathrm{b}$, where

$$A = \begin{bmatrix} 1 & -2 & -1 & 3 \\ 1 & -2 & 0 & 1 \\ -3 & -2 & 1 & 7 \\ 0 & -2 & 8 & 5 \end{bmatrix} \;\text{ and }\; \mathrm{b} = \begin{bmatrix} -12 \\ -5 \\ -14 \\ -7 \end{bmatrix}.$$

2.6. Let A be a nonsingular symmetric matrix and have the factorization $A = LDM^T$, where L and M are unit lower triangular matrices and D is a diagonal matrix. Show that $L = M$.

Hint: You may use the uniqueness argument, the first claim in Section 2.4.2.

CHAPTER **3**

The Least-Squares Problem

In this chapter, we study the least-squares problem, which arises frequently in scientific and engineering computations. After describing the problem, we will develop tools to solve the problem:

- normal equation
- reflectors and rotators
- the Gram-Schmidt orthonormalization process
- the QR decomposition
- the singular value decomposition (SVD)

Contents of Chapter 3

3.1. The Discrete Least-Squares Problem

Definition 3.1. *Let $A \in \mathbb{R}^{m\times n}$ and $\mathrm{b} \in \mathbb{R}^m$. The* **linear least-squares problem** *is to find $\mathrm{x} \in \mathbb{R}^n$ which minimizes*

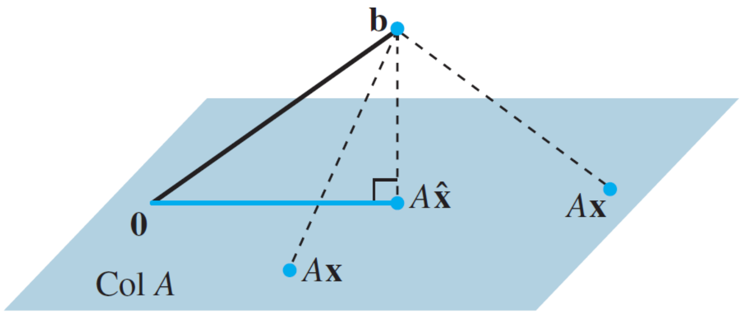
$$||A\mathrm{x} - \mathrm{b}||_2 \text{ or, equivalently, } ||A\mathrm{x} - \mathrm{b}||_2^2$$

Remark 3.2.

- *The error $||\mathbf{r}||_2 \equiv ||A\mathrm{x} - \mathrm{b}||_2$ is the distance between* b *and the vector*

$$A\mathbf{x} = x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + \cdots + x_n\mathbf{a}_n \tag{3.1}$$

- *If $\hat{\mathrm{x}}$ is the LS solution, then $A\hat{\mathrm{x}} = Proj_{Col(A)}\mathrm{b}$:*



Methods Solving LS problems	1. normal equation	• It is the fastest but lea when the condition nu
	2. QR decomp.	• QR decomposition is t up to twice as much as
	3. SVD	• SVD is of most use on i.e., when A is not of fu more expensive.
	4. transform to a linear system	• The last method lets and improve the soluti conditioned.

- All methods but the 3rd can be adapted to work with sparse matrices.
- We will assume $rank(A) = n$ in method 1 and 2.

3.1.1. Normal equations

- **Normal Equation:**
$$A^TA\mathbf{x} = A^T\mathbf{b} \tag{3.2}$$

- **Derivation:** Goal: Minimize
$$||A\mathbf{x}-\mathbf{b}||_2^2 = (A\mathbf{x}-\mathbf{b})^T(A\mathbf{x}-\mathbf{b})$$

$$\begin{aligned} 0 &= \lim_{\mathbf{e}\to\mathbf{0}} \frac{||A(\mathbf{x}+\mathbf{e})-\mathbf{b}||_2^2 - ||A\mathbf{x}-\mathbf{b}||_2^2}{||\mathbf{e}||_2} \\ &= \lim_{\mathbf{e}\to\mathbf{0}} \frac{2\mathbf{e}^T(A^TA\mathbf{x}-A^T\mathbf{b}) + \mathbf{e}^TA^TA\mathbf{e}}{||\mathbf{e}||_2} \end{aligned}$$

$$\text{2nd Term:} \left|\frac{\mathbf{e}^TA^TA\mathbf{e}}{||\mathbf{e}||_2^2}\right| = \frac{||A\mathbf{e}||_2^2}{||\mathbf{e}||_2} \le \frac{||A||_2^2\cdot||\mathbf{e}||_2^2}{||\mathbf{e}||_2} = ||A||_2^2\cdot||\mathbf{e}||_2$$
$$\implies \frac{\mathbf{e}^TA^TA\mathbf{e}}{||\mathbf{e}||_2^2} \to 0, \quad \text{as } \mathbf{e}\to\mathbf{0}.$$

Thus, if we look for x where the gradient of $||A\mathrm{x}-\mathbf{b}||_2^2$ vanishes, then $A^TA\mathrm{x} = A^T\mathbf{b}$.

Remark 3.3.

- *Why* $\mathbf{x}$ *is the minimizer of* $||A\mathbf{x} - \mathbf{b}||_2^2 \overset{\Delta}{=} f(\mathbf{x})$*?*

$$\begin{aligned} \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} &= \nabla_{\mathbf{x}} f(\mathbf{x}) = \frac{\partial \left((A\mathbf{x}-\mathbf{b})^T (A\mathbf{x}-\mathbf{b}) \right)}{\partial \mathbf{x}} \\ &= \frac{\partial \left(\mathbf{x}^T A^T A \mathbf{x} - 2\mathbf{x}^T A^T \mathbf{b} \right)}{\partial \mathbf{x}} = 2A^T A\mathbf{x} - 2A^T \mathbf{b} \\ H_f(\mathbf{x}) &= \frac{\partial}{\partial \mathbf{x}} \left(\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right) = 2A^T A \quad \textit{positive definite} \\ & \quad \textit{(under the assumption that } rank(A) = n\textit{)} \end{aligned}$$

So the function f *is strictly convex, and any critical point is a global minimum.*[1] □

- *For* $A \in \mathbb{R}^{m\times n}$*, if* $rank(A) = n$*, then* $A^T A$ *is SPD; indeed,*
  - $A^T A$ *is symmetric since* $(A^T A)^T = A^T (A^T)^T = A^T A$*.*
  - *Since* $rank(A) = n$*,* $m \ge n$*, and the columns of* A *are linearly independent.*

    So $A\mathbf{x} = \mathbf{0}$ *has only the trivial solution.*

    Hence, for any $\mathbf{x} \in \mathbb{R}^n$*, if* $\mathbf{x} \neq \mathbf{0}$*, then* $A\mathbf{x} \neq \mathbf{0}$*.*
  - *For any* $\mathbf{0} \neq \mathbf{x} \in \mathbb{R}^n$*,* $\mathbf{x}^T A^T A\mathbf{x} = (A\mathbf{x})^T (A\mathbf{x}) = ||A\mathbf{x}||_2^2 > 0$*.*

  So the normal equation $A^T A\mathbf{x} = A^T\mathbf{b}$ *has a unique solution* $\mathbf{x} = (A^T A)^{-1} A^T \mathbf{b}$*.*

[1] $H_f(\mathbf{x})$ is called **Hessian matrix** of f.

- *If let* $\mathrm{x}' = \mathrm{x} + \mathrm{e}$*, where* x *is the minimizer, one can easily verify the following*
$$||A\mathbf{x}' - \mathbf{b}||_2^2 = ||A\mathbf{x} - \mathbf{b}||_2^2 + ||A\mathrm{e}||_2^2 \geq ||A\mathbf{x} - \mathbf{b}||_2^2 \tag{3.3}$$
(See Homework 1.) Note that since A *is of full-rank,* $A\mathrm{e} = 0$ *if and only if* $\mathrm{e} = 0$*; which again proves the uniqueness of the minimizer* x*.*

- *Numerically, normal equation can be solved by the Cholesky factorization. Total cost:*
$$n^2m + \frac{1}{3}n^2 + O(n^2) \text{ flops}.$$
Since $m \geq n$*, the term* n^2m *dominates the cost.*

Example 3.4. *Solve the following least-squares problem*

$$\begin{cases} x_1 + 3x_2 - 2x_3 = 5 \\ 3x_1 + x_3 = 4 \\ 2x_1 x_2 + x_3 = 3 \\ 2x_1 x_2 + x_3 = 2 \\ x_1 - 2x_2 + 3x_3 = 1 \end{cases}$$

Solution. Let the coefficient matrix be denoted by A and the RHS by b. Then

$$A = \begin{bmatrix} 1 & 3 & -2 \\ 3 & 0 & 1 \\ 2 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & -2 & 3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 5 \\ 4 \\ 3 \\ 2 \\ 1 \end{bmatrix}$$

$$\Rightarrow A^TA = \begin{bmatrix} 19 & 5 & 8 \\ 5 & 15 & -10 \\ 8 & -10 & 16 \end{bmatrix}, \quad A^T\mathbf{b} = \begin{bmatrix} 28 \\ 18 \\ 2 \end{bmatrix}$$

Solving the normal equation $A^TA\mathrm{x} = A^T\mathbf{b}$ gives

$$\mathbf{x} = \left[\frac{7}{5}, \frac{3}{5}, -\frac{1}{5}\right]^T.$$

3.2. LS Solution by QR Decomposition

3.2.1. Gram-Schmidt process

Given a basis $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_p\}$ for a nonzero subspace $W \subseteq \mathbb{R}^n$, define

$$\begin{aligned}
\mathbf{v}_1 &= \mathbf{x}_1 \\
\mathbf{v}_2 &= \mathbf{x}_2 - \frac{\mathbf{x}_2 \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1}\mathbf{v}_1 \\
\mathbf{v}_3 &= \mathbf{x}_3 - \frac{\mathbf{x}_3 \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1}\mathbf{v}_1 - \frac{\mathbf{x}_3 \cdot \mathbf{v}_2}{\mathbf{v}_2 \cdot \mathbf{v}_2}\mathbf{v}_2 \\
&\vdots \\
\mathbf{v}_p &= \mathbf{x}_p - \frac{\mathbf{x}_p \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1}\mathbf{v}_1 - \frac{\mathbf{x}_p \cdot \mathbf{v}_2}{\mathbf{v}_2 \cdot \mathbf{v}_2}\mathbf{v}_2 - \cdots - \frac{\mathbf{x}_p \cdot \mathbf{v}_{p-1}}{\mathbf{v}_{p-1} \cdot \mathbf{v}_{p-1}}\mathbf{v}_{p-1}
\end{aligned}$$

Then $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_p\}$ is an **orthogonal basis** for W.

In addition,

$$span\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_k\} = span\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k\}, \quad 1 \le k \le p$$

Example 3.5. *Let* $\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \\ -1 \\ 1 \end{bmatrix}$, $\mathbf{x}_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$, $\mathbf{x}_3 = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 1 \end{bmatrix}$, *and* $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ *be a basis for a subspace* W *of* $\mathbb{R}^4$. *Construct an orthogonal basis for* W.

Solution. Let $\mathbf{v}_1 = \mathbf{x}_1$. Then $W_1 = Span\{\mathbf{x}_1\} = Span\{\mathbf{v}_1\}$.

$$\mathbf{v}_2 = \mathbf{x}_2 - \frac{\mathbf{x}_2 \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1}\mathbf{v}_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} - \frac{2}{3}\begin{bmatrix} 1 \\ 0 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{3} \\ 1 \\ \frac{2}{3} \\ \frac{1}{3} \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix},$$

$$\mathbf{v}_2 \Leftarrow 3\mathbf{v}_2 = \begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix} \quad \text{(for convenience)}$$

Then,

$$\mathbf{v}_3 = \mathbf{x}_3 - \frac{\mathbf{x}_3 \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1}\mathbf{v}_1 - \frac{\mathbf{x}_3 \cdot \mathbf{v}_2}{\mathbf{v}_2 \cdot \mathbf{v}_2}\mathbf{v}_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \frac{-1}{3}\begin{bmatrix} 1 \\ 0 \\ -1 \\ 1 \end{bmatrix} - \frac{8}{15}\begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix} = \frac{1}{5}\begin{bmatrix} -1 \\ -3 \\ 3 \\ 4 \end{bmatrix},$$

$$\mathbf{v}_3 \Leftarrow 5\mathbf{v}_3 = \begin{bmatrix} -1 \\ -3 \\ 3 \\ 4 \end{bmatrix}$$

Then $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ is an orthogonal basis for W.

Note: Replacing $\mathbf{v}_2$ with $3\mathbf{v}_2$, $\mathbf{v}_3$ with $5\mathbf{v}_3$ does not break orthogonality.

Definition **3.6.** *An* **orthonormal basis** *is an orthogonal basis* $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_p\}$ *with* $||\mathbf{v}_i|| = 1$*, for all* $i = 1, 2 \cdots, p$*. That is* $\mathbf{v}_i \cdot \mathbf{v}_j = \delta_{ij}$*.*

Example **3.7.** *Let* $\mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$*,* $\mathbf{v}_2 = \begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix}$*,* $W = Span\{\mathbf{v}_1, \mathbf{v}_2\}$*. Construct an orthonormal basis for* W*.*

Solution. It is easy to see that $\mathbf{v}_1 \cdot \mathbf{v}_2 = 0$. So $\{\mathbf{v}_1, \mathbf{v}_2\}$ is an orthogonal basis for W. Rescale the basis vectors to form a orthonormal basis $\{\mathbf{u}_1, \mathbf{u}_2\}$:

$$\begin{aligned} \mathbf{u}_1 &= \frac{1}{||\mathbf{v}_1||}\mathbf{v}_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} \\ \mathbf{u}_2 &= \frac{1}{||\mathbf{v}_2||}\mathbf{v}_2 = \frac{1}{3\sqrt{3}} \begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \end{bmatrix} \end{aligned}$$

3.2.2. QR decomposition, by Gram-Schmidt process

$\boxed{\textbf{Theorem}}$ **3.8.** *(QR* **Decomposition***) Let $A \in \mathbb{R}^{m\times n}$ with $m \geq n$. Suppose A has full column rank ($rank(A) = n$). Then there exist a unique $m \times n$* **orthogonal** *matrix Q ($Q^TQ = I_n$) and a* **unique** *$n \times n$* **upper triangular** *matrix R with* **positive** *diagonals $r_{ii} > 0$ such that $A = QR$.*

Proof. **(Sketch)** Let $A = [\mathbf{a}_1 \ \cdots \ \mathbf{a}_n]$.

1. Construct an orthonormal basis $\{\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_n\}$ for $W = Col(A)$ by Gram-Schmidt process, starting from $\{\mathbf{a}_1, \ \cdots, \ \mathbf{a}_n\}$

2. $Q = [\mathbf{q}_1 \ \mathbf{q}_2 \ \cdots \mathbf{q}_n]$.

   $$\begin{array}{l} Span\{\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k\} = Span\{\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_k\}, \quad 1 \leq k \leq n \\ \mathbf{a}_k = r_{1k}\mathbf{q}_1 + r_{2k}\mathbf{q}_2 + \cdots + r_{kk}\mathbf{q}_k + 0\cdot\mathbf{q}_{k+1} + \cdots + 0\cdot\mathbf{q}_n \\ \textbf{Note}\text{: If } r_{kk} < 0\text{, multiply both } r_{kk} \text{ and } \mathbf{q}_k \text{ by } -1. \end{array}$$

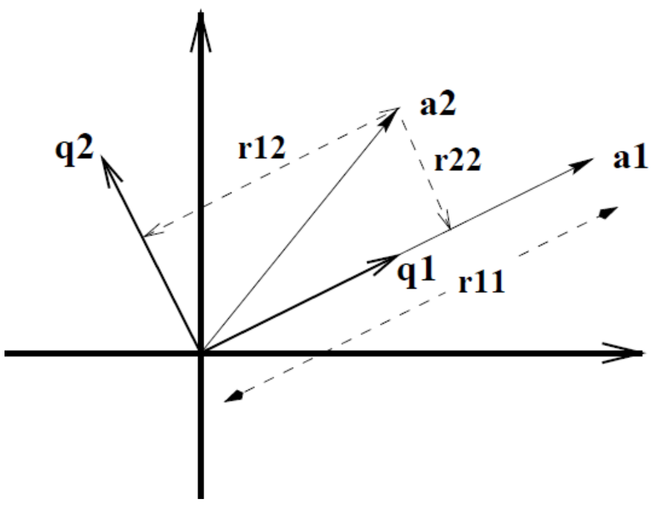
3. Let $\mathbf{r}_k = [r_{1k} \ r_{2k} \ \cdots \ r_{kk} \ 0 \ \cdots \ 0]^T$. Then $\mathbf{a}_k = Q\mathbf{r}_k$ for $k = 1, 2, \cdots, n$. Let $R = [\mathbf{r}_1 \ \mathbf{r}_2 \ \cdots \ \mathbf{r}_n]$.

4. $A = [\mathbf{a}_1 \ \cdots \ \mathbf{a}_n] = [Q\mathbf{r}_1 \ Q\mathbf{r}_2 \ \cdots \ Q\mathbf{r}_n] = QR$.

□

Example 3.9. *Let A be a 2×2 matrix:*

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = [\ \mathbf{a}_1 \ \ \mathbf{a}_2\ ]$$



The GS method:

$$\begin{aligned} r_{11} &= ||\mathbf{a}_1||_2 \\ \mathbf{q}_1 &= \mathbf{a}_1/r_{11} \\ r_{12} &= \mathbf{q}_1 \cdot \mathbf{a}_2 \\ \mathbf{q}_2 &= \mathbf{a}_2 - r_{12}\mathbf{q}_1 \\ r_{22} &= ||\mathbf{q}_2||_2 \\ \mathbf{q}_2 &= \mathbf{q}_2/r_{22} \end{aligned}$$

QR ***Decomposition for a*** 2×2 ***matrix.***

$\boxed{\textbf{Algorithm}}$ **3.10. (QR Decomposition).** *Let*

$$A = [\mathbf{a}_1\ \cdots\ \mathbf{a}_n].$$

In Gram-Schmidt, $\mathbf{q}_i \leftarrow \mathbf{v}_i/||\mathbf{v}_i||$, *then*

$$\begin{aligned}
\mathbf{a}_1 &= (\mathbf{q}_1 \cdot \mathbf{a}_1)\mathbf{q}_1 \\
\mathbf{a}_2 &= (\mathbf{q}_1 \cdot \mathbf{a}_2)\mathbf{q}_1 + (\mathbf{q}_2 \cdot \mathbf{a}_2)\mathbf{q}_2 \\
\mathbf{a}_3 &= (\mathbf{q}_1 \cdot \mathbf{a}_3)\mathbf{q}_1 + (\mathbf{q}_2 \cdot \mathbf{a}_3)\mathbf{q}_2 + (\mathbf{q}_3 \cdot \mathbf{a}_3)\mathbf{q}_3 \\
&\vdots \\
\mathbf{a}_n &= \sum_{j=1}^{n} (\mathbf{q}_j \cdot \mathbf{a}_n)\mathbf{q}_j
\end{aligned}$$

Thus,

$$A = [\mathbf{a}_1\ \mathbf{a}_2\ \cdots\ \mathbf{a}_n] = QR,$$

where

$$\begin{aligned}
Q &= [\mathbf{q}_1\ \mathbf{q}_2\ \cdots\ \mathbf{q}_n] \\
R &= \begin{bmatrix}
\mathbf{q}_1 \cdot \mathbf{a}_1 & \mathbf{q}_1 \cdot \mathbf{a}_2 & \mathbf{q}_1 \cdot \mathbf{a}_3 & \cdots & \mathbf{q}_1 \cdot \mathbf{a}_n \\
0 & \mathbf{q}_2 \cdot \mathbf{a}_2 & \mathbf{q}_2 \cdot \mathbf{a}_3 & \cdots & \mathbf{q}_2 \cdot \mathbf{a}_n \\
0 & 0 & \mathbf{q}_3 \cdot \mathbf{a}_3 & \cdots & \mathbf{q}_3 \cdot \mathbf{a}_n \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & \mathbf{q}_n \cdot \mathbf{a}_n
\end{bmatrix}
\end{aligned}$$

Algorithm 3.11. Classical Gram-Schmidt (CGS).

```
$R(1,1) = ||A(:,1)||_2$
$Q(:,1) = A(:,1)/R(1,1)$
for $k = 2 : n$
    $R(1:k-1,k) = Q(1:m,1:k-1)^T A(1:m,k)$
    $z = A(1:m,k) - Q(1:m,1:k-1)R(1:k-1,k)$
    $R(k,k) = ||z||_2$
    $Q(1:m,k) = z/R(k,k)$
end
```

- **For $A \in \mathbb{R}^{m\times n}$, if $rank(A) = n$,**

$$\mathbf{q}_k = \left(\mathbf{a}_k - \sum_{i=1}^{k-1} r_{ik}\mathbf{q}_i\right)/r_{kk}, \quad \mathbf{z}_k = \mathbf{a}_k - \sum_{i=1}^{k-1} r_{ik}\mathbf{q}_i,$$

where $r_{ik} = \mathbf{q}_i^T \mathbf{a}_k$, $i = 1 : k-1$ and
$\mathbf{z}_k \in Span\{\mathbf{q}_1, \cdots, \mathbf{q}_{k-1}\}^\perp$.

$\boxed{\textbf{Algorithm}}$ **3.12. Modified Gram-Schmidt (MGS)**

```
for $k = 1 : n$
        $R(k,k) = ||A(1:m,k)||_2$
        $Q(1:m,k) = A(1:m,k)/R(k,k)$
        for $j = k+1 : n$
                $R(k,j) = Q(1:m,k)^T A(1:m,j)$
                $A(1:m,j) = A(1:m,j) - Q(1:m,k)R(k,j)$
        end
end
```

- **Let $A^{(k)} \in \mathbb{R}^{m\times(n-k+1)}$. Then it is defined by**

$$A - \sum_{i=1}^{k-1} \mathbf{q}_i \mathbf{r}_i^T = \sum_{i=k}^{n} \mathbf{q}_i \mathbf{r}_i^T [\mathbf{0} \;\; A^{(k)}], \qquad A^{(k)} = \begin{array}{c} \left[\begin{array}{cc} \mathbf{z} & B \end{array} \right] \\ \begin{array}{cc} 1 & n-k \end{array} \end{array}$$

Then $r_{kk} = ||\mathbf{z}||_2$, $\mathbf{q}_k = \mathbf{z}/r_{kk}$ and $(r_{k,k+1} \;\; \cdots \;\; r_{kn}) = \mathbf{q}_k^T B$. Then compute the outer product $A^{(k+1)} = B - \mathbf{q}_k (r_{k,k+1} \;\; \cdots \;\; r_{kn})$. ...

$\boxed{\textbf{Algorithm}}$ **3.13.** *The classical Gram-Schmidt (CGS) and modified Gram-Schmidt (MGS) Algorithms for factoring $A = QR$:*

```
for i=1 to n   /* compute ith columns of Q and R */
  $q_i = a_i$
  for $j = 1$ to $i-1$ /*subtract components in $q_j$ direction from $a_i$*/
      $\begin{cases} r_{ji} = q_j^T a_i & \text{CGS} \\ r_{ji} = q_j^T q_i & \text{MGS} \end{cases}$
      $q_i = q_i - r_{ji} q_j$
  end for
  $r_{ii} = ||q_i||_2$
  if $r_{ii} = 0$   /* $a_i$ is linearly dependent on $a_1, \cdots, a_{i-1}$ */
      quit
  else if
      $q_i = q_i / r_{ii}$
end for
```

Note: The two formulas for r_{ij} in the above algorithm are mathematically equivalent.

Example 3.14. *Use the CGS process to find a QR decomposition of*

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ -1 & 0 & 2 \\ 1 & 1 & 1 \end{bmatrix}.$$

Solution. Let $A \equiv [\mathbf{x}_1\ \mathbf{x}_2\ \mathbf{x}_3]$. Then, it follows from Example 3.5 on page 77 that $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$, where

$$\mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \\ -1 \\ 1 \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix}, \quad \mathbf{v}_3 = \begin{bmatrix} -1 \\ -3 \\ 3 \\ 4 \end{bmatrix},$$

is an orthogonal basis of $Col(A)$, the column space of A.
Normalize the basis vectors, we get

$$\mathbf{q}_1 = \frac{\mathbf{v}_1}{||\mathbf{v}_1||} = \begin{bmatrix} \frac{1}{\sqrt{3}} \\ 0 \\ -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \end{bmatrix}, \mathbf{q}_2 = \frac{\mathbf{v}_2}{||\mathbf{v}_2||} = \begin{bmatrix} \frac{1}{\sqrt{15}} \\ \frac{3}{\sqrt{15}} \\ \frac{2}{\sqrt{15}} \\ \frac{1}{\sqrt{15}} \end{bmatrix}, \mathbf{q}_3 = \frac{\mathbf{v}_3}{||\mathbf{v}_3||} = \begin{bmatrix} -\frac{1}{\sqrt{35}} \\ -\frac{3}{\sqrt{35}} \\ \frac{3}{\sqrt{35}} \\ \frac{4}{\sqrt{35}} \end{bmatrix}.$$

Then,

$$\begin{aligned} r_{11} = \mathbf{q}_1 \cdot \mathbf{x}_1 = \sqrt{3}, \quad & r_{12} = \mathbf{q}_1 \cdot \mathbf{x}_2 = \tfrac{2}{\sqrt{3}}, \quad && r_{13} = \mathbf{q}_1 \cdot \mathbf{x}_3 = -\tfrac{1}{\sqrt{3}} \\ & r_{22} = \mathbf{q}_2 \cdot \mathbf{x}_2 = \tfrac{5}{\sqrt{15}}, && r_{23} = \mathbf{q}_2 \cdot \mathbf{x}_3 = \tfrac{8}{\sqrt{15}}, \\ & && r_{33} = \mathbf{q}_3 \cdot \mathbf{x}_3 = \tfrac{7}{\sqrt{35}} \end{aligned}$$

Thus,

$$Q = \begin{bmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{15}} & -\frac{1}{\sqrt{35}} \\ 0 & \frac{3}{\sqrt{15}} & -\frac{3}{\sqrt{35}} \\ -\frac{1}{\sqrt{3}} & \frac{2}{\sqrt{15}} & \frac{3}{\sqrt{35}} \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{15}} & \frac{4}{\sqrt{35}} \end{bmatrix}, \quad R = \begin{bmatrix} \sqrt{3} & \frac{2}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \\ 0 & \frac{5}{\sqrt{15}} & \frac{8}{\sqrt{15}} \\ 0 & 0 & \frac{7}{\sqrt{35}} \end{bmatrix}. \tag{3.4}$$

It is easy to check that $A = QR$.

Example 3.15. *Find a QR decomposition of* $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$ *by MGS.*

Solution. Let $A \equiv [\mathbf{a}_1\ \mathbf{a}_2\ \mathbf{a}_3]$.

$\boldsymbol{k} = \mathbf{1}$:

$$r_{11} = ||\mathbf{a}_1||_2 = \sqrt{2}, \quad \mathbf{a}_1 \leftarrow \mathbf{a}_1/r_{11} = \frac{\sqrt{2}}{2}\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\left.\begin{array}{l} r_{12} = \mathbf{a}_1^T\mathbf{a}_2 = \frac{\sqrt{2}}{2}, \\ \\ r_{13} = \mathbf{a}_1^T\mathbf{a}_3 = 0, \end{array}\right\} \quad \mathbf{a}_2 \leftarrow \mathbf{a}_2 - r_{12}\mathbf{a}_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} - \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 \\ -1 \\ 2 \\ 0 \end{bmatrix}$$

$$\mathbf{a}_3 \leftarrow \mathbf{a}_3 - r_{13}\mathbf{a}_1 = \mathbf{a}_3 - \mathbf{0} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

$\boldsymbol{k} = \mathbf{2}$:

$$r_{22} = ||\mathbf{a}_2||_2 = \frac{\sqrt{6}}{2}, \quad \mathbf{a}_2 \leftarrow \mathbf{a}_2/r_{22} = \frac{\sqrt{6}}{6}\begin{bmatrix} 1 \\ -1 \\ 2 \\ 0 \end{bmatrix}$$

$$r_{23} = \mathbf{a}_2^T\mathbf{a}_3 = 0, \quad \mathbf{a}_3 \leftarrow \mathbf{a}_3 - r_{23}\mathbf{a}_2 = \mathbf{a}_3 - \mathbf{0} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

$\boldsymbol{k} = \mathbf{3}$:

$$r_{33} = ||\mathbf{a}_3||_2 = 2, \quad \mathbf{a}_3 \leftarrow \mathbf{a}_3/r_{33} = \frac{1}{2}\begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

So we get the QR decomposition result:

$$A = QR = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{6}}{6} & \frac{1}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} & -\frac{1}{2} \\ 0 & \frac{2\sqrt{6}}{6} & -\frac{1}{2} \\ 0 & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} \sqrt{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & \frac{\sqrt{6}}{2} & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Note: In the above process, we use matrix A to store Q, and R is stored in another array.

Remark 3.16.

- **CGS** *(classical Gram-Schmidt) method has very poor numerical properties in that there is typically a severe loss of orthogonality among the computed* q_i*.*
- **MGS** *(Modified Grad-Schmidt) method yields a much better computational result by a* $\underline{\textit{rearrangement}}$ *of the calculation.*
- **CGS** *is numerically unstable in floating point arithmetic when the columns of* A *are nearly linearly dependent.* **MGS** *is more stable but may still result in* Q *being far from orthogonal when* A *is ill-conditioned.*
- *Both CGS and MGS require* $\mathcal{O}(2mn^2)$ *flops to compute a QR factorization of* $A \in \mathbb{R}^{m\times n}$*.*

3.2.3. Application to LS solution

- Let $A \in \mathbb{R}^{m\times n}$ with $m \ge n$. If $m > n$, we can choose $m-n$ more orthonormal vectors $\tilde{Q}$ so that $[Q, \tilde{Q}]$ is a square orthogonal matrix. So

$$\begin{aligned}
||A\mathbf{x}-\mathbf{b}||_2^2 &= ||[Q,\tilde{Q}]^T(A\mathbf{x}-\mathbf{b})||_2^2 = \left|\left| \begin{bmatrix} Q^T \\ \tilde{Q}^T \end{bmatrix} (QR\mathbf{x}-\mathbf{b}) \right|\right|_2^2 \\
&= \left|\left| \begin{bmatrix} I^{n\times n} \\ O^{(m-n)\times n} \end{bmatrix} R\mathbf{x} - \begin{bmatrix} Q^T\mathbf{b} \\ \tilde{Q}^T\mathbf{b} \end{bmatrix} \right|\right|_2^2 \\
&= \left|\left| \begin{bmatrix} R\mathbf{x}-Q^T\mathbf{b} \\ -\tilde{Q}^T\mathbf{b} \end{bmatrix} \right|\right|_2^2 \\
&= \left[(R\mathbf{x}-Q^T\mathbf{b})^T \;\; -(\tilde{Q}^T\mathbf{b})^T\right] \begin{bmatrix} R\mathbf{x}-Q^T\mathbf{b} \\ -\tilde{Q}^T\mathbf{b} \end{bmatrix} \\
&= ||R\mathbf{x}-Q^T\mathbf{b}||_2^2 + ||\tilde{Q}^T\mathbf{b}||_2^2 \ge ||\tilde{Q}^T\mathbf{b}||_2^2
\end{aligned}$$

- Since $rank(A) = rank(R) = n$, R is nonsingular.
  So $\mathbf{x} = R^{-1}Q^T\mathbf{b}$ is the solution for $R\mathbf{x} = Q^T\mathbf{b}$, and

$$\min_{\mathbf{x}} ||A\mathbf{x}-\mathbf{b}||_2^2 = ||\tilde{Q}^T\mathbf{b}||_2^2$$

- **Second Derivation:**

$$\begin{aligned} A\mathbf{x}-\mathbf{b} &= QR\mathbf{x}-\mathbf{b} = QR\mathbf{x} - (QQ^T + I - QQ^T)\mathbf{b} \\ &= Q(R\mathbf{x}-Q^T\mathbf{b}) - (I-QQ^T)\mathbf{b}. \end{aligned}$$

  Since

$$\begin{aligned} &\big(Q(R\mathbf{x}-Q^T\mathbf{b})\big)^T \big((I-QQ^T)\mathbf{b}\big) \\ &\quad = (R\mathbf{x}-Q^T\mathbf{b})^T Q^T (I-QQ^T)\mathbf{b} = (R\mathbf{x}-Q^T\mathbf{b})^T [0]\mathbf{b} = \mathbf{0}, \end{aligned}$$

  the vectors $Q(R\mathbf{x}-Q^T\mathbf{b})$ and $(I-QQ^T)\mathbf{b}$ are orthogonal. So by Pythagorean theorem,

$$\begin{aligned} ||A\mathbf{x}-\mathbf{b}||_2^2 &= ||Q(R\mathbf{x}-Q^T\mathbf{b})||_2^2 + ||(I-QQ^T)\mathbf{b}||_2^2 \\ &= ||R\mathbf{x}-Q^T\mathbf{b}||_2^2 + ||(I-QQ^T)\mathbf{b}||_2^2 \end{aligned}$$

- **Third Derivation:**

$$\begin{aligned} \mathbf{x} &= (A^TA)^{-1}A^T\mathbf{b} \\ &= (R^TQ^TQR)^{-1}R^TQ^T\mathbf{b} = (R^TR)^{-1}R^TQ^T\mathbf{b} \\ &= R^{-1}R^{-T}R^TQ^T\mathbf{b} = R^{-1}Q^T\mathbf{b} \end{aligned}$$

Example 3.17. *Solve the LS problem* $\min_{\mathbf{x}} ||A\mathbf{x}-\mathbf{b}||_2^2$ *when*

$$A = \begin{bmatrix} 2 & -1 \\ 0 & 1.e-6 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ 2.e-6 \\ 2 \end{bmatrix}.$$

Solution. **(1. Normal equation):** The normal equation $A^TA\mathbf{x} = A^T\mathbf{b}$ reads

$$\begin{bmatrix} 4 & -2 \\ -2 & 1+10^{-12} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 2.e-12 \end{bmatrix}$$

Solution: $x_1 = 1$, $x_2 = 2$.

(Round-off issue): The arrays A^TA and $A^T\mathbf{b}$, when rounded to **10 digits**, read

$$A^TA \doteq \begin{bmatrix} 4 & -2 \\ -2 & 1 \end{bmatrix}, \quad A^T\mathbf{b} \doteq \begin{bmatrix} 0 \\ 2.e-12 \end{bmatrix}$$

No solution (singular matrix)

(2. *QR* decomposition): Factor $A = QR$ as

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad R = \begin{bmatrix} 2 & -1 \\ 0 & 1.e-6 \end{bmatrix}.$$

Solution of $R\mathbf{x} = Q^T\mathbf{b} = \begin{bmatrix} 0 \\ 2.e-6 \end{bmatrix}$: $x_1 = 1$, $x_2 = 2$.

Conclusion: QR is better in numerical stability.

3.3. Orthogonal Matrices

Approaches for orthogonal decomposition:

1. Classical Gram-Schmidt (CGS) and
   Modified Gram-Schmidt (MGS) process (§ 3.2.2)
2. Householder reflection (reflector)
3. Givens rotation (rotator)

This section studies Householder reflection and Givens rotation in detail, in the use for QR decomposition.

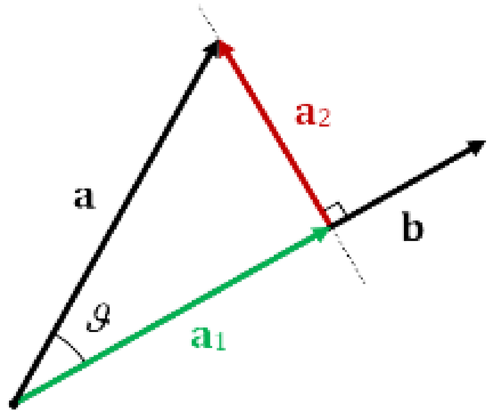
3.3.1. Householder reflection

The Householder reflection is also called “reflector". In order to introduce the reflector, let’s begin with the projection.

Definition 3.18. *Let* a, b $\in \mathbb{R}^n$*. The* **projection of** a **onto** {b} *is the vector defined by*

$$proj_{\mathrm{b}}\mathrm{a} = \frac{\mathrm{a}\cdot\mathrm{b}}{\mathrm{b}\cdot\mathrm{b}}\,\mathrm{b} \tag{3.5}$$

Pictorial interpretation:



- $\mathrm{proj}_{\mathrm{b}}\mathrm{a} = \mathrm{a}_1$
- $\|\mathbf{a}_1\| = \|\mathbf{a}\| \cos\theta$
- $\mathrm{a} = \mathrm{a}_1 + \mathrm{a}_2$ is called an **orthogonal decomposition** of a.

Householder reflection (or, Householder transformation, Elementary reflector) is a transformation that takes a vector and reflects it about some plane or hyperplane. Let $\mathbf{u}$ be a unit vector which is orthogonal to the hyperplane. Then, the **reflection of a point** x **about this hyperplane** is

$$\mathbf{x} - 2\,\mathrm{proj}_{\mathbf{u}}\mathbf{x} = \mathbf{x} - 2(\mathbf{x}\cdot\mathbf{u})\mathbf{u}. \tag{3.6}$$

Note that

$$\mathbf{x} - 2(\mathbf{x}\cdot\mathbf{u})\mathbf{u} = \mathbf{x} - 2(\mathbf{u}\mathbf{u}^T)\mathbf{x} = (I - 2\mathbf{u}\mathbf{u}^T)\mathbf{x}.$$

Definition 3.19. *Let* $\mathbf{u} \in \mathbb{R}^n$ *with* $\|\mathbf{u}\| = 1$*. Then the matrix* $P = I - 2\mathbf{u}\mathbf{u}^T$ *is called a* **Householder matrix** *(or, Householder reflector).*

Proposition 3.20. *Let* $P = I - 2\mathbf{u}\mathbf{u}^T$ *with* $\|\mathbf{u}\| = 1$*.*

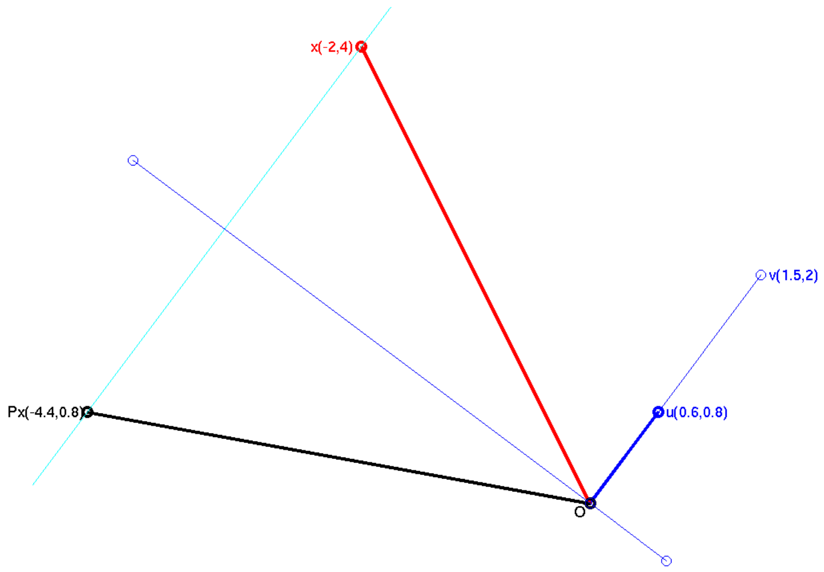
1. $P\mathbf{u} = -\mathbf{u}$
2. $P\mathbf{v} = \mathbf{v}$, *if* $\mathbf{u} \cdot \mathbf{v} = 0$
3. $P = P^T$ *(*P *is symmetric)*
4. $P^T = P^{-1}$ *(*P *is orthogonal)*
5. $P^{-1} = P$ *(*P *is an involution)*
6. $PP^T = P^2 = I$*.*

Indeed,

$$PP^T = (I - 2\mathbf{u}\mathbf{u}^T)(I - 2\mathbf{u}\mathbf{u}^T) = I - 4\mathbf{u}\mathbf{u}^T + 4\mathbf{u}\mathbf{u}^T\mathbf{u}\mathbf{u}^T = I$$

Example 3.21. *Let* $\mathbf{u} = \begin{bmatrix} .6 \\ .8 \end{bmatrix}$, $P = I - 2\mathbf{u}\mathbf{u}^T$, $\mathbf{x} = \begin{bmatrix} -2 \\ 4 \end{bmatrix}$. *Then* $P\mathbf{x}$ *is* **reflection** *of* $\mathbf{x}$ *about the line through the origin* $\perp$ $\mathbf{u}$.

Solution. $P\mathbf{x} = \begin{bmatrix} .28 & -.96 \\ -.96 & -.28 \end{bmatrix} \begin{bmatrix} -2 \\ 4 \end{bmatrix} = \begin{bmatrix} -4.4 \\ .8 \end{bmatrix}$ One can easily see that $||P\mathbf{x}||_2 = ||\mathbf{x}||_2 = \sqrt{20}$.



Claim 3.22. *For a nonzero vector* $\mathbf{v} \in \mathbb{R}^n$, *define*

$$P = I - \beta\,\mathbf{v}\mathbf{v}^T, \quad \beta = \frac{2}{||\mathbf{v}||_2^2}.$$

- *Then* P *is a reflector, which satisfies all the properties in Proposition 3.20.*
- *Any nonzero constant multiple of* $\mathbf{v}$ *will generate the same reflector.*

$\boxed{\textbf{Theorem}}$ **3.23.** *Let* $\mathbf{x}, \mathbf{y}$ *be two vectors in* $\mathbb{R}^n$ *with* $\mathbf{x} \neq \mathbf{y}$ *but* $||\mathbf{x}||_2 = ||\mathbf{y}||_2$. *Then there exists a unique reflector* P *such that* $P\mathbf{x} = \mathbf{y}$.

Proof. **(Existence)** Let $\mathbf{v} = \mathbf{x} - \mathbf{y}$ and $P = I - \beta\, \mathbf{v}\mathbf{v}^T$, where $\beta = 2/\|\mathbf{v}\|_2^2$. Note that

$$\mathbf{x} = \frac{1}{2}(\mathbf{x} - \mathbf{y}) + \frac{1}{2}(\mathbf{x} + \mathbf{y}). \tag{3.7}$$

Then

$$P(\mathbf{x} - \mathbf{y}) = -(\mathbf{x} - \mathbf{y}). \tag{3.8}$$

Since $(\mathbf{x} - \mathbf{y}) \cdot (\mathbf{x} + \mathbf{y}) = ||\mathbf{x}||_2^2 - ||\mathbf{y}||_2^2 = 0$, by Proposition 3.20, we have

$$P(\mathbf{x} + \mathbf{y}) = (\mathbf{x} + \mathbf{y}). \tag{3.9}$$

It follows from (3.7), (3.8), and (3.9) that $P\mathbf{x} = \mathbf{y}$.

(Uniqueness) It will be a good exercise to prove the uniqueness of the reflector; see Homework 2. $\square$

Summary: Let $\mathbf{x} \neq \pm\mathbf{y}$ but $||\mathbf{x}||_2 = ||\mathbf{y}||_2$. Then the reflector $P : \mathbf{x} \mapsto \pm\mathbf{y}$ can be formulated by

1. $\mathbf{v} = \mathbf{x} - (\pm\mathbf{y}) = \mathbf{x} \mp \mathbf{y}$.
2. $P = I - \beta\, \mathbf{v}\mathbf{v}^T$, where $\beta = 2/\|\mathbf{v}\|_2^2$.

Example 3.24. *Let* $\mathbf{x} = \begin{bmatrix} -2 \\ 1 \\ 2 \end{bmatrix}$*,* $\mathbf{y} = \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix}$*. Find the Householder matrix* P *so* $P\mathbf{x} = \mathbf{y}$*.*

Solution. Let

$$\begin{aligned}
\mathbf{v} &= \mathbf{x} - \mathbf{y} = \begin{bmatrix} -2 \\ 1 \\ 2 \end{bmatrix} - \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}, \quad ||\mathbf{v}|| = \sqrt{6} \\
P &= I - \frac{2}{6}\mathbf{v}\mathbf{v}^T \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \frac{2}{6} \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 2 \end{bmatrix} \\
&= \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{2}{3} \\ -\frac{1}{3} & \frac{2}{3} & -\frac{2}{3} \\ -\frac{2}{3} & -\frac{2}{3} & -\frac{1}{3} \end{bmatrix}
\end{aligned}$$

Now, it is a simple matter to check $P\mathbf{x} = \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix} = \mathbf{y}$.

$\boxed{\textbf{Claim}}$ **3.25.** *For an arbitrary vector* $\mathbf{x}$*, suppose we want to find a Householder reflection* P *in order to zero out all but the first entry of* $\mathbf{x}$*, i.e.,*

$$P\mathbf{x} = [c \;\; 0 \;\; \cdots \;\; 0]^T = c \cdot \mathbf{e}_1.$$

Then, the reflector can be defined as follows.

1. *Set* $c = \pm\|\mathbf{x}\|_2 \quad (||\mathbf{x}||_2 = ||P\mathbf{x}||_2 = |c|)$
2. *Set* $\mathbf{v} = \mathbf{x} - c \cdot \mathbf{e}_1$ *such that* $\mathbf{v} \neq 0$*.*
3. *Get* $P = I - \beta\,\mathbf{v}\mathbf{v}^T$*, where* $\beta = 2/\|\mathbf{v}\|_2^2$*.*

Remarks:

- In the first step of Claim 3.25, we often set

$$c = -\textbf{sign}(x_1)\|\mathbf{x}\|_2 \tag{3.10}$$

  so that

$$\mathbf{v} = \mathbf{x} - c \cdot \mathbf{e}_1 = \begin{bmatrix} x_1 + \textbf{sign}(x_1)\|\mathbf{x}\|_2 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \text{ with } \mathbf{u} = \frac{\mathbf{v}}{||\mathbf{v}||_2}, \tag{3.11}$$

  which may avoid **cancellation** ($\mathbf{v} = 0$).
  Definition: We write (3.11) as $\boxed{\mathbf{u} = \text{House}(\mathbf{x})}$.

- Using the arguments in the proof of Theorem 3.23, one can check

$$P\mathbf{x} = \begin{bmatrix} c \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} -\textbf{sign}(x_1)\|\mathbf{x}\|_2 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Example 3.26. *Let* $\mathrm{x} = \begin{bmatrix} -2 \\ 1 \\ 2 \end{bmatrix}$. *Find the Householder reflector so that all but the first entry of the reflection of* x *are zero.*

Solution. $||\mathrm{x}||_2 = \sqrt{(-2)^2 + 1^2 + 2^2} = \sqrt{9} = 3$. $x_1 = -2$.

$$\begin{aligned} \mathbf{v} &= \begin{bmatrix} x_1 + \textbf{sign}(x_1)||\mathbf{x}||_2 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -2-3 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -5 \\ 1 \\ 3 \end{bmatrix}, \\ &\Longrightarrow ||\mathbf{v}|| = \sqrt{30} \\ P &= I - \beta \mathbf{v}\mathbf{v}^T \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \frac{2}{30} \begin{bmatrix} -5 \\ 1 \\ 2 \end{bmatrix} \begin{bmatrix} -5 & 1 & 2 \end{bmatrix} \\ &= \begin{bmatrix} -\frac{2}{3} & \frac{1}{3} & \frac{2}{3} \\ \frac{1}{3} & \frac{14}{15} & -\frac{2}{15} \\ \frac{2}{3} & -\frac{2}{15} & \frac{11}{15} \end{bmatrix} \end{aligned}$$

Thus,

$$P\mathrm{x} = \begin{bmatrix} -\textbf{sign}(x_1)||\mathbf{x}||_2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}$$

Question: What if we set $\mathbf{v} = \begin{bmatrix} x_1 - ||\mathbf{x}||_2 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, for $x_1 > 0$?

Answer. To avoid cancellation, instead of computing $x_1 - ||\mathbf{x}||_2$ directly, we may use

$$x_1 - ||\mathbf{x}||_2 = \frac{x_1^2 - ||\mathbf{x}||_2^2}{x_1 + ||\mathbf{x}||_2} = \frac{-(x_2^2 + x_3^2 + \cdots + x_n^2)}{x_1 + ||\mathbf{x}||_2} \equiv \frac{-\sigma}{x_1 + \mu}$$

where $\sigma = x_2^2 + x_3^2 + \cdots + x_n^2$ and $\mu = ||\mathbf{x}||_2$.
Then,

$$||\mathbf{v}||_2^2 = \sigma + \left(\frac{-\sigma}{x_1 + \mu}\right)^2$$

and

$$P = I - \beta \mathbf{v}\mathbf{v}^T, \;\; \beta = \frac{2}{||\mathbf{v}||_2^2}; \quad P\mathbf{x} = ||\mathbf{x}||_2\,\mathbf{e}_1.$$

□

$\boxed{\textbf{Algorithm}}$ **3.27. (house(x))** *Given* $\mathbf{x} \in \mathbb{R}^n$, *the function* `house(x)` *computes* $\mathbf{v} \in \mathbb{R}^n$ *with* $\mathbf{v}(1) = 1$ *and* $\beta \in \mathbb{R}$, *such that*

$$P = I_n - \beta \mathbf{v}\mathbf{v}^T, \quad P\mathbf{x} = ||\mathbf{x}||_2\, \mathbf{e}_1,$$

where P *is a reflector. Total cost of* `house(x)`: $3n$ *flops.*

```
function [$\mathbf{v}, \beta$] = house($\mathbf{x}$)
  $n$ = length($\mathbf{x}$);
  $\sigma = \mathbf{x}(2:n)^T\mathbf{x}(2:n)$;     ($\sim 2n$ flops)
  $\mathbf{v} = [1;\ \mathbf{x}(2:n)]^T$;
  if  $\sigma = 0$;
      $\beta = 0$;
  else
      $\mu = \sqrt{\mathbf{x}(1)^2 + \sigma}$;  // $= ||\mathbf{x}||_2$
      if $\mathbf{x}(1) \le 0$
          $\mathbf{v}(1) = \mathbf{x}(1) - \mu$;
      else
          $\mathbf{v}(1) = -\sigma/(\mathbf{x}(1) + \mu)$;
      end
      $\beta = 2\mathbf{v}(1)^2/(\sigma + \mathbf{v}(1)^2)$;
      $\mathbf{v} = \mathbf{v}/\mathbf{v}(1)$;        ($\sim n$ flops)
  end
```

The feature of $\mathbf{v}(1) = 1$ *is important in the computation of* PA *when* m *is small, and in the computation of* AP *when* n *is small.*

Application of reflectors to a matrix $A \in \mathbb{R}^{m\times n}$

- If $P = I - \beta \mathbf{v}\mathbf{v}^T \in \mathbb{R}^{m\times m}$, then

$$PA = (I - \beta \mathbf{v}\mathbf{v}^T)A = A - \mathbf{v}\mathbf{w}^T \tag{3.12}$$

  where $\mathbf{w} = \beta A^T \mathbf{v}$.

- If $P = I - \beta \mathbf{v}\mathbf{v}^T \in \mathbb{R}^{n\times m}$, then

$$AP = A(I - \beta \mathbf{v}\mathbf{v}^T) = A - \mathbf{w}\mathbf{v}^T \tag{3.13}$$

  where $\mathbf{w} = \beta A \mathbf{v}^T$.

- An m-by-n Householder update cost: $4mn$ **flops**
  = {a matrix-vector multiplication} + {an outer product update}.

- Householder updates **never** require the explicit formation of the Householder matrix.

3.3.2. QR decomposition by reflectors

Example 3.28. *Compute the QR decomposition of a 5×3 matrix A using Householder Reflection.*

Solution.

$$\underbrace{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}}_{A} \xrightarrow{P_1} \underbrace{\left[\begin{array}{c|cc} * & * & * \\ \hline 0 & * & * \\ 0 & * & * \\ 0 & * & * \\ 0 & * & * \end{array}\right]}_{A_1=P_1A} \xrightarrow{P_2} \underbrace{\left[\begin{array}{cc|c} * & * & * \\ 0 & * & * \\ \hline 0 & 0 & * \\ 0 & 0 & * \\ 0 & 0 & * \end{array}\right]}_{A_2=P_2P_1A} \xrightarrow{P_3} \underbrace{\left[\begin{array}{ccc} * & * & * \\ 0 & * & * \\ 0 & 0 & * \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}\right]}_{A_3=P_3P_2P_1A}$$

where

$$P_2 = \left[\begin{array}{c|c} 1 & 0 \\ \hline 0 & P_2' \end{array}\right], \quad P_3 = \left[\begin{array}{c|c} I_2 & 0 \\ \hline 0 & P_3' \end{array}\right],$$

P_1, P_2', P_3' are Householder matrices.

Let $\tilde{R} \equiv A_3$. Then

$$A = P_1^T P_2^T P_3^T \tilde{R} = QR,$$

where

$$Q = P_1^T P_2^T P_3^T = P_1 P_2 P_3,$$

and R is the first 3 rows of A_3.

Algorithm 3.29. (**QR Factorization Using Reflectors**)

```
for  $i = 1$ to min$(m-1, n)$
    $\mathbf{u}_i$ = House$(A(i:m, i))$
    $P'_i = I - 2\mathbf{u}_i\mathbf{u}_i^T$
    $A(i:m, i:n) = P'_i A(i:m, i:n)$                (†)
end
```

The computation in (†):

$$(I - 2\mathbf{u}_i\mathbf{u}_i^T)A(i:m, i:n) = A(i:m, i:n) - 2\mathbf{u}_i\left(\mathbf{u}_i^T A(i:m, i:n)\right) \qquad (‡)$$

costs much less than the matrix-matrix multiplication.

Computational complexity: In Algorithm 3.29, the computations are mainly in (†) or (‡).

$2(m-i)(n-i)$	for	$\mathbf{u}_i^T A(i:m, i:n)$
$(m-i)(n-i)$	for	$\mathbf{u}_i(*)$
$(m-i)(n-i)$	for	subtractions: $A(i:m, i:n) - 2\mathbf{u}_i(*)$

Thus

$$\begin{aligned} \text{total} &= \sum_{i=1}^{n} 4(m-i)(n-i) = 4\sum_{i=1}^{n}\left(mn - i(m+n) + i^2\right) \\ &= 4mn^2 - 4(m+n)\frac{n(n+1)}{2} + 4\cdot\frac{n(n+1)(2n+1)}{6} \\ &\sim 2mn^2 - \frac{2n^3}{3} \end{aligned}$$

$\boxed{\textbf{Algorithm}}$ **3.30. (Householder QR) [10]** *Given $A \in \mathbb{R}^{m\times n}$ with $m \geq n$, the following algorithm finds Householder matrices $H_1, \cdots, H_n$ such that if $Q = H_1 \cdots H_n$, then $Q^T A = R$ is upper triangular.*

The upper triangular part of A is overwritten by the upper triangular part of R and components $j+1 : m$ of the j-th Householder vector are stored in $A(j+1:m,\ j)$, $j < m$.

```
for $j = 1 : n$
    $[\mathbf{v}, \beta] =$ house$(A(j:m,\ j))$
    $A(j:m,\ j:n) = (I_{m-j+1} - \beta \mathbf{v}\mathbf{v}^T) A(j:m,\ j:n)$
    if $j < m$
        $A(j+1:m,\ j) = \mathbf{v}(2:m-j+1)$
    end
end
```

How A is overwritten in QR [10, p.225]

$$\mathbf{v}^{(j)} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \mathbf{1} \\ v_{j+1}^{(j)} \\ \vdots \\ v_m^{(j)} \end{bmatrix} \begin{matrix} \\ \\ \\ \leftarrow j \\ \\ \\ \\ \end{matrix} \; ; \quad A = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} & r_{15} & r_{16} \\ v_2^{(1)} & r_{22} & r_{23} & r_{24} & r_{25} & r_{26} \\ v_3^{(1)} & v_3^{(2)} & r_{33} & r_{34} & r_{35} & r_{36} \\ v_4^{(1)} & v_4^{(2)} & v_4^{(3)} & r_{44} & r_{45} & r_{46} \\ v_5^{(1)} & v_5^{(2)} & v_5^{(3)} & v_5^{(4)} & r_{55} & r_{56} \\ v_6^{(1)} & v_6^{(2)} & v_6^{(3)} & v_6^{(4)} & v_6^{(5)} & r_{66} \end{bmatrix}$$

Upon completion, the upper triangular part of A is overwritten by R, and the other part by Householder vectors.

- If it quires the computation of

  $$Q = (H_n \cdots H_1)^T = H_1 \cdots H_n,$$

  then it can be accumulated using (3.12), or (3.13). This accumulation requires $4(m^2n - mn^2 + n^3/3)$ flops.

Example 3.31. *Let* $A = \begin{bmatrix} 0.70000 & 0.70711 \\ 0.70001 & 0.70711 \end{bmatrix}$. *Compute the factor* Q *by Householder reflection and MGS, and test orthogonality of* Q. *Round the intermediate results to 5 significant digits.*

Solution. **(MGS)** The classical and modified Gram-Schmidt algorithms are identical in the 2×2 case.

$$r_{11} = .98996, \ \mathbf{q}_1 = \mathbf{a}_1/r_{11} = \begin{bmatrix} .70000/.98996 \\ .70001/.98996 \end{bmatrix} = \begin{bmatrix} .70710 \\ .70711 \end{bmatrix}$$

$$r_{12} = \mathbf{q}_1^T \mathbf{a}_2 = .70710 * .70711 + .70711 * .70711 = 1.0000$$

$$\mathbf{q}_2 = \mathbf{a}_1 - r_{12}\mathbf{q}_1 = \begin{bmatrix} .70711 \\ .70711 \end{bmatrix} - \begin{bmatrix} .70710 \\ .70711 \end{bmatrix} = \begin{bmatrix} .00001 \\ .00000 \end{bmatrix},$$

$$Q = \begin{bmatrix} .70710 & 1.0000 \\ .70711 & .0000 \end{bmatrix}, \quad ||Q^TQ - I_2|| \approx 0.70710$$

So the Q matrix is not close to any orthogonal matrix!

(Householder reflection) Solution by Householder.

$$\mathbf{a}_1 = \begin{bmatrix} .70000 \\ .70001 \end{bmatrix}, \ \mathbf{a}_2 = \begin{bmatrix} .70711 \\ .70711 \end{bmatrix}, \ ||\mathbf{a}_1||_2 = r_{11} \approx .98996.$$

$$\mathbf{v} = \begin{bmatrix} .70000 + .98996. \\ .70001 \end{bmatrix} = \begin{bmatrix} 1.6900 \\ .70001 \end{bmatrix},$$

$$P_1 = I_2 - 2\mathbf{v}\mathbf{v}^T/||\mathbf{v}||_2^2, \quad P_1\mathbf{a}_2 = \mathbf{a}_2 - \frac{2\mathbf{v}^T\mathbf{a}_2}{||\mathbf{v}||_2^2}\mathbf{v}$$

$$\Rightarrow Q = P_1^T = \begin{bmatrix} -.70710 & -.70711 \\ -.70711 & .70710 \end{bmatrix}$$

$$||Q^TQ - I_2|| = 5.0379e-6$$

So Householder reflection is more stable than MGS.

3.3.3. Solving LS problems by Householder reflectors

- To solve the LS problem $\min ||A\mathbf{x}-\mathbf{b}||_2^2$ using $A = QR$, we need to compute

  $$Q^T\mathbf{b} \;\; \text{and} \;\; R\mathbf{x} = Q^T\mathbf{b}.$$

  Since $A \to P_n P_{n-1} \cdots P_1 A = R$, $Q = (P_n P_{n-1} \cdots P_1)^T$,

  $$Q^T\mathbf{b} = P_n P_{n-1} \cdots P_1 \mathbf{b}.$$

  So we can apply **exactly** the same steps that were applied to A in the Householder QR algorithm:

- **(Computing Q^Tb)**

```
for $i = 1$ to $n$
    $\gamma = -2 \cdot \mathbf{u}_i^T \mathbf{b}(i:m)$
    $\mathbf{b}(i:m) = \mathbf{b}(i:m) + \gamma \mathbf{u}_i$
end for
```

- **(Solving $R\mathbf{x} = Q^T\mathbf{b}$)**

```
$\tilde{R} \equiv [R \quad Q^T\mathbf{b}]$
for $k = n : -1 : 1$
    $\tilde{R}(k,\ n+1) = \tilde{R}(k,\ n+1)/\tilde{R}(k,k);$
    if $k \neq 1$
        $\tilde{R}(1:k-1,\ n+1) = \tilde{R}(1:k-1,\ n+1) - \tilde{R}(1:k-1,k)\tilde{R}(k,n+1);$
    end if
end for
```

(Solution x overwrites the $(n+1)$-th column of $\tilde{R}$.)

- Total cost in solving the LS problem:

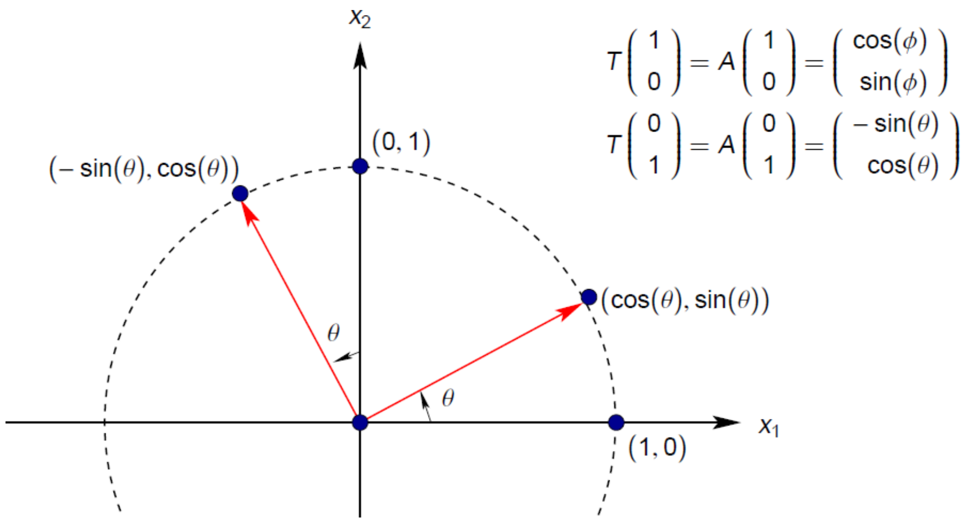
$$\mathbf{2mn^2 - \frac{2n^3}{3}}$$

3.3.4. Givens rotation

Definition 3.32. A Givens rotation

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

rotates any vector $\mathbf{x} \in \mathbb{R}^2$ *counter-clockwise by* θ*:*



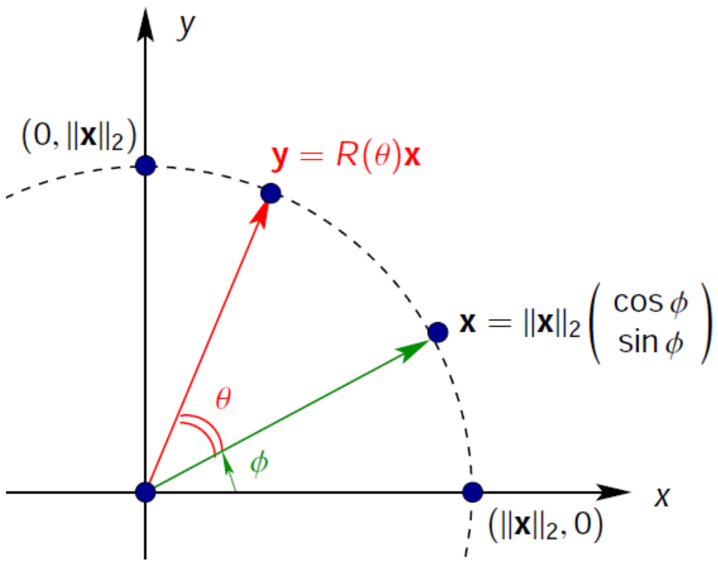
x_2
x_1
$T\begin{pmatrix}1\\0\end{pmatrix} = A\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}\cos(\phi)\\\sin(\phi)\end{pmatrix}$
$T\begin{pmatrix}0\\1\end{pmatrix} = A\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}-\sin(\theta)\\\cos(\theta)\end{pmatrix}$
$(0,1)$
$(-\sin(\theta), \cos(\theta))$
$(\cos(\theta), \sin(\theta))$
θ
θ
$(1,0)$

Linear Transformation: $T : \mathbb{R}^2 \to \mathbb{R}^2$, *with standard matrix* $A = R(\theta)$.

Givens Rotation:

Any vector $\mathbf{x} \in \mathbb{R}^2$, we can always write

$$\mathbf{x} \;=\; ||\mathbf{x}||_2 \begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix}$$


y
$(0, \|\mathbf{x}\|_2)$
$\mathbf{y} = R(\theta)\mathbf{x}$
$\mathbf{x} = \|\mathbf{x}\|_2 \begin{pmatrix} \cos\phi \\ \sin\phi \end{pmatrix}$
θ
ϕ
x
$(\|\mathbf{x}\|_2, 0)$

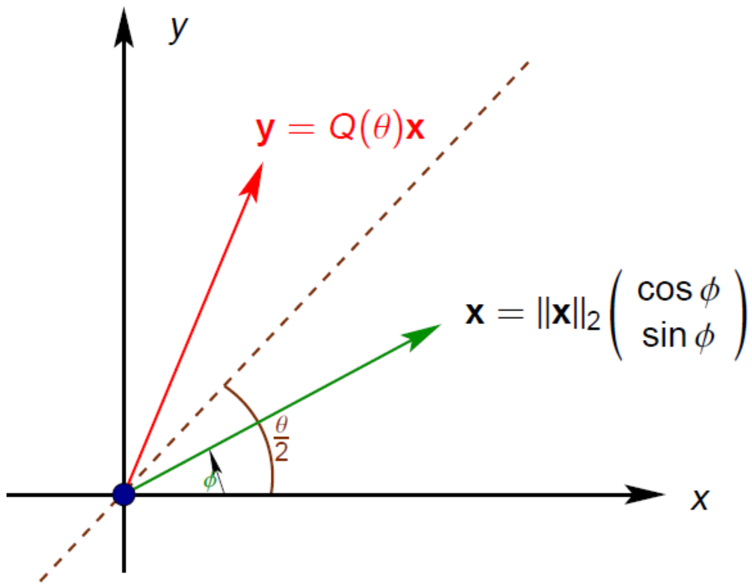
- Thus,

$$\begin{aligned} \mathbf{y} \;&=\; R(\theta)\mathbf{x} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} ||\mathbf{x}||_2 \begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix} \\ &=\; ||\mathbf{x}||_2 \begin{bmatrix} \cos\theta\cos\phi - \sin\theta\sin\phi \\ \sin\theta\cos\phi + \cos\theta\sin\phi \end{bmatrix} = ||\mathbf{x}||_2 \begin{bmatrix} \cos(\theta+\phi) \\ \sin(\theta+\phi) \end{bmatrix} \end{aligned}$$

- $R(\theta)$ is an **orthogonal** matrix, i.e., $R(\theta)^T R(\theta) = I$

- $Q(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \sin(\theta) & -\cos(\theta) \end{bmatrix}$ is a **reflection** matrix. That is, $\mathbf{y} = Q(\theta)\mathbf{x}$ is a reflection of x across the line $\frac{\theta}{2}$.

$$\begin{aligned} \mathbf{y} =& Q(\theta)\mathbf{x} \\ =& ||\mathbf{x}||_2 \begin{bmatrix} \cos(\theta-\phi) \\ \sin(\theta-\phi) \end{bmatrix} \end{aligned}$$



y
$\mathbf{y} = Q(\theta)\mathbf{x}$
$\mathbf{x} = ||\mathbf{x}||_2 \begin{pmatrix} \cos\phi \\ \sin\phi \end{pmatrix}$
$\frac{\theta}{2}$
ϕ
x

- Let

$$R(i,j,\theta) = \left[\begin{array}{ccccccccc} 1 &&&&&&&& \\ & 1 &&&&&&& \\ && \ddots &&&&&& \\ &&& \cos\theta && -\sin\theta &&& \\ &&&& \ddots &&&& \\ &&& \sin\theta && \cos\theta &&& \\ &&&&&& \ddots && \\ &&&&&&& 1 & \\ &&&&&&&& 1 \end{array}\right] \begin{array}{l} \\ \\ \\ \leftarrow \boldsymbol{i} \\ \\ \leftarrow \boldsymbol{j} \\ \\ \\ \\ \end{array}$$
$$\begin{array}{ccccccccc} & & & \uparrow & & \uparrow &&& \\ &&& \boldsymbol{i} && \boldsymbol{j} &&& \end{array}$$

Then, for any vector $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} = R(i,j,\theta)\mathbf{x}$ is a rotation of x counter-clockwise by θ. The matrix $R(i,j,\theta)$ is sometimes called a **plane rotator**.

- For any vector $\mathrm{x} \in \mathbb{R}^2$, $\mathrm{y} = R(i, j, \theta)\mathrm{x}$ rotates x counter-clockwise by θ.

$$y_k = \begin{cases} cx_i - sx_j, & k = i, \\ sx_i + cx_j, & k = j, \\ x_k, & k \neq i, j \end{cases}$$

  $R(i, j, \theta)\mathrm{x}$ just effects two rows (i and j rows) of x. In order to transform y_j to zero, we let

$$c = \cos\theta = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}, \qquad s = \sin\theta = \frac{-x_j}{\sqrt{x_i^2 + x_j^2}},$$

  Then

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_i \\ x_j \end{bmatrix} = \begin{bmatrix} \sqrt{x_i^2 + x_j^2} \\ 0 \end{bmatrix}$$

$\boxed{\textbf{Algorithm}}$ **3.33. (Givens Rotation Algorithm)** *Given scalars a and b, the following function computes $c = \cos\theta$ and $s = \sin\theta$.*

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

where $r = \sqrt{a^2 + b^2}$.

```
function [c, s] = givens(a, b)
if $b = 0$
    $c = 1; s = 0;$
else
    if $|b| > |a|$
        $\tau = -\frac{a}{b}; s = \frac{1}{\sqrt{1+\tau^2}}; c = s\tau;$
    else
        $\tau = -\frac{b}{a}; c = \frac{1}{\sqrt{1+\tau^2}}; s = c\tau;$
    end
end
```

Note: *It does not compute θ and it does not involve inverse trig. functions.* **Total cost:** *5 flops and a single square root.*

Example 3.34. *Compute* $R(\mathbf{2},\mathbf{4},\theta)\mathbf{x}$*, where* $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$.

Solution. Since

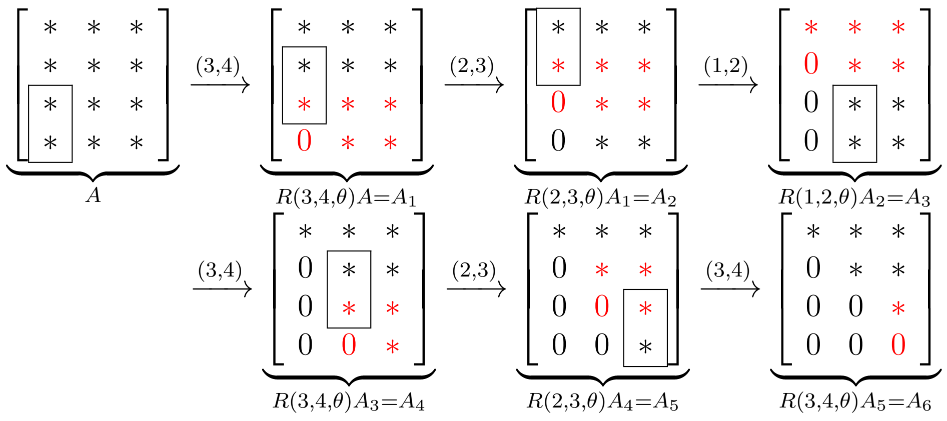
$$\cos\theta = \frac{2}{\sqrt{2^2+4^2}} = \frac{1}{\sqrt{5}} \quad \textbf{and} \quad \sin\theta = \frac{-4}{\sqrt{2^2+4^2}} = \frac{-2}{\sqrt{5}},$$

we have

$$R(\mathbf{2},\mathbf{4},\theta)\mathbf{x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{5}} & 0 & \frac{2}{\sqrt{5}} \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{2}{\sqrt{5}} & 0 & \frac{1}{\sqrt{5}} \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ \sqrt{20} \\ 3 \\ 0 \end{bmatrix}.$$

3.3.5. QR factorization by Givens rotation

Graphical Interpretation



- **Note**: $R(2,3,\theta)$'s and $R(3,4,\theta)$'s in different steps are different since they depend on different x_i's and x_j's.

$\boxed{\text{Algorithm}}$ 3.35. (Givens rotation: Row operations)

$$A([i,k],:) = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} A([i,k],:)$$

This requires just $6n$ flops:

```
for $j = 1:n$
    $\tau_1 = A(i,j);$
    $\tau_2 = A(k,j);$
    $A(i,j) = c\tau_1 - s\tau_2;$
    $A(k,j) = s\tau_1 + c\tau_2;$
end
```

- Recall that
  $A \leftarrow R(i,k,\theta)A$ effects i-th and k-th rows only.

$\boxed{\textbf{Algorithm}}$ **3.36. (Givens QR Algorithm)**
Given** $A \in \mathbb{R}^{m\times n}$ **with** $m \geq n$**, the following algorithm overwrites** A **with** $R(= Q^TA)$**, where** R **is the upper triangular and** Q **is orthogonal.

```
for  $j = 1 : n$                                          $\leftarrow n$ steps
  for  $i = m : -1 : j+1$                                 $\leftarrow m-j$ steps
    $[c, s] = \texttt{givens}(A(i-1, j), A(i, j))$         $\leftarrow 6$ flops
    $A(i-1:i, j:n) = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} A(i-1:i, j:n)$   $\leftarrow 6(n-j)$ flops
  end
end
```

Computational complexity:

$$\begin{aligned}\sum_{j=1}^{n}(m-j)[6(n-j)+6] &= 6\sum_{j=1}^{n}\left[mn-(m+n)j+j^2\right] \\ &= 6\left[mn^2-(m+n)\sum_{j=1}^{n}j+\sum_{j=1}^{n}j^2\right] \\ &= 3mn^2-n^3-3mn+n = \mathcal{O}(\boldsymbol{3mn^2-n^3}).\end{aligned}$$

Note: For the notation $R_{i,j} = R(i, j, \theta)$,

$$\underbrace{R_{n,n+1}\cdots R_{m-1,m}}_{\text{last column}}\cdots\underbrace{R_{2,3}\cdots R_{m-1,m}}_{\text{2nd column}}\underbrace{R_{1,2}\cdots R_{m-1,m}}_{\text{1st column}}A = R$$

Thus,

$$Q = (R_{n,n+1}\cdots R_{m-1,m}\cdots R_{2,3}\cdots R_{m-1,m}R_{1,2}\cdots R_{m-1,m})^T$$

3.3.6. Error propagation

Lemma 3.37. *Let P be an exact Householder (or Givens) transformation, and $\tilde{P}$ be its floating point approximation. Then*

$$\textit{float}(\tilde{P}A) = P(A+E),\ \ ||E||_2 = \mathcal{O}(\epsilon)\cdot||A||_2$$
$$\textit{float}(A\tilde{P}) = (A+F)P,\ \ ||F||_2 = \mathcal{O}(\epsilon)\cdot||A||_2$$

In words, this says that applying a single orthogonal matrix is backward stable.

Proof. Apply the usual formula

$$\text{float}(a \circ b) = (a \circ b)(1+\epsilon)$$

to the formulas for computing and applying $\tilde{P}$. □

Definition 3.38. *Let alg(x) be a computational algorithm of $f(x)$, including effects of round-off. The algorithm alg(x) is said to be* **backward stable** *if for all x there is a "small" δx such that alg$(x) = f(x+\delta x)$, where δx is called the* **backward error**.

When our employed algorithm is backward stable, we say *informally* that we get the exact answer $(f(x+\delta x))$ for a slightly wrong problem $(x+\delta x)$.

$\boxed{\textbf{Theorem}}$ **3.39.** ***Consider applying a sequence of orthogonal transformations to A_0. Then the computed product is an exact orthogonal transformation of $A_0 + \delta A$, where $||\delta A||_2 = O(\epsilon)||A||_2$. In other words, the entire computation is backward stable:***

$$\textit{\textbf{float}}(\tilde{P}_j \tilde{P}_{j-1} \cdots \tilde{P}_1 A_0 \tilde{Q}_1 \tilde{Q}_2 \cdots \tilde{Q}_j) = P_j \cdots P_1 (A_0 + E) Q_1 \cdots Q_j$$

with $||E||_2 = j \cdot O(\epsilon) \cdot ||A||_2$. Here, as in Lemma 3.37, $\tilde{P}_i$ and $\tilde{Q}_i$ are floating orthogonal matrices and P_i and Q_i are exact orthogonal matrices.

Why Orthogonal Matrices?

- float$(\tilde{X}A) = XA + E = X(A + X^{-1}E) \equiv X(A + F)$, where $||E||_2 \leq O(\epsilon)||X||_2 \cdot ||A||_2$
- $||F||_2 = ||X^{-1}E||_2 \leq ||X^{-1}||_2 \cdot ||E||_2 \leq O(\epsilon) \cdot \kappa_2(X) \cdot ||A||_2$.
- The error $||E||_2$ is magnified by the condition number $\kappa_2(X) \geq 1$.
- In a larger product $\tilde{X}_k \cdots \tilde{X}_1 A \tilde{Y}_1 \cdots \tilde{Y}_k$, the error would be magnified by

$$\boldsymbol{\Pi_i \kappa_2(X_i) \cdot \kappa_2(Y_i)}.$$

- This factor is minimized if and only if all X_i and Y_i are **orthogonal** (or scalar multiples of **orthogonal** matrices); the factor is one.

Exercises for Chapter 3

3.1. Verify (3.3) on page 74. (You may have to use the normal equation (3.2).)

3.2. Prove the uniqueness part of Theorem 3.23. **Hint**: Let P, Q be reflectors such that $P\mathrm{x} = \mathrm{y}$ and $Q\mathrm{x} = \mathrm{y}$. Then, use properties in Proposition 3.20 to conclude $P = Q$.

3.3. Consider the matrix in Example 3.14. Implement codes to find its QR decomposition by using

(a) the modified Gram-Schmidt process,
(b) the Householder reflection, and
(c) the Givens rotation.

Then, compare your results with the decomposition in (3.4).

3.4. For a given data set

x	1	2	3	4	5
y	0.8	2.1	3.3	4.1	4.7

we would like to find the best-fitting line $y = a_0 + a_1 x$.

(a) Construct the algebraic system of the form $A\mathrm{v} = \mathrm{y}$, where $A \in \mathbb{R}^{5\times 2}$, $\mathrm{v} = [a_0, a_1]^T$, and y is a column vector including y-values in the data set.

(b) Solve the least-squares problem by

i. Normal equation
ii. QR factorization operated by the classical Gram-Schmidt process.

(c) Find the corresponding errors.

CHAPTER **4**

Singular Value Decomposition (SVD)

The QR decomposition is a fine tool for the solution of least-squares problems when the matrix is known to have full rank. However, if the matrix does not have full rank, or if the rank is not known, a more powerful tool is required. Such more powerful tools (for rank-deficient cases) are

- The QR decomposition with column pivoting
- Singular value decomposition (SVD)

SVD may be the most important matrix decomposition of all, for both theoretical and computational purposes.

Contents of Chapter 4

4.1. Introduction to the SVD

Rank-deficient case: We begin with a discussion on rank-deficient algebraic problems.

Definition **4.1.** *Let $A \in \mathbb{R}^{m\times n}$, $m \ge n$. The matrix A is called* **rank-deficient** *if rank$(A) = k < n$.*

Theorem **4.2.** *Let $A \in \mathbb{R}^{m\times n}$, $m \ge n$, with rank$(A) = k > 0$. Then there exist matrices $\widehat{A}$, Q, and R such that*

$$\widehat{A} = Q\,R, \tag{4.1}$$

where $\widehat{A}$ is obtained from A by permuting its columns, $Q \in \mathbb{R}^{m\times m}$ is orthogonal, and

$$R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{m\times n},$$

$R_{11} \in \mathbb{R}^{k\times k}$ is nonsingular and upper triangular.

Proposition **4.3.** **(Rank-deficient LS problems)** *Let $A \in \mathbb{R}^{m\times n}$, $m \ge n$, with rank$(A) = k < n$. Then there exists an $(n-k)$-dimensional set of vectors* x *that minimize*

$$||A\mathrm{x} - \mathbf{b}||_2.$$

Proof. Let $A\mathrm{z} = \mathbf{0}$. Then if x minimize $||A\mathrm{x}-\mathbf{b}||_2$, so does x+z. $\dim(\mathbf{Null}(A)) = n - \text{rank}(A) = n - k$ by the Rank Theorem. □

Singular Value Decomposition (SVD)

Theorem 4.4. (SVD Theorem) *Let $A \in \mathbb{R}^{m\times n}$ with $m \geq n$. Then we can write*

$$A = U\,\Sigma\,V^T, \tag{4.2}$$

where $U \in \mathbb{R}^{m\times n}$ and satisfies $U^TU = I$, $V \in \mathbb{R}^{n\times n}$ and satisfies $V^TV = I$, and $\Sigma =$ diag$(\sigma_1, \sigma_2, \cdots, \sigma_n)$, where

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0.$$

Remark 4.5. *The matrices are illustrated pictorially as*

$$\begin{bmatrix} \\ \quad A \quad \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \quad U \quad \\ \\ \end{bmatrix} \begin{bmatrix} \quad \Sigma \quad \end{bmatrix} \begin{bmatrix} \quad V^T \quad \end{bmatrix}$$

U is $m \times n$ orthogonal (the **left singular vectors** *of A.)*
Σ is $n \times n$ diagonal (the **singular values** *of A.)*
V is $n \times n$ orthogonal (the **right singular vectors** *of A.)*

- *For some $k \leq n$, the singular values may satisfy*

$$\underbrace{\sigma_1 \;\geq\; \sigma_2 \;\geq\; \cdots \;\geq\; \sigma_k}_{\text{nonzero singular values}} > \sigma_{k+1} = \cdots = \sigma_n = 0.$$

  In this case, rank$(A) = k$.

- *If $m < n$, the* **SVD** *is defined by considering A^T.*

Proof. (of Theorem 4.4) Use induction on m and n: we assume that the SVD exists for $(m-1)\times(n-1)$ matrices, and prove it for $m\times n$. We assume $A\neq 0$; otherwise we can take $\Sigma=0$ and let U and V be arbitrary orthogonal matrices.

- The basic step occurs when $n=1$ ($m\geq n$). We let $A=U\Sigma V^T$ with $U=A/||A||_2$, $\Sigma=||A||_2$, $V=1$.
- For the induction step, choose $\mathbf{v}$ so that
$$||\mathbf{v}||_2=1 \text{ and } ||A||_2=||A\mathbf{v}||_2>0.$$
- Let $\mathbf{u}=\frac{A\mathbf{v}}{||A\mathbf{v}||_2}$, which is a unit vector. Choose $\tilde{U}$, $\tilde{V}$ such that
$$U=[\mathbf{u}\ \ \tilde{U}]\in\mathbb{R}^{m\times n} \text{ and } V=[\mathbf{v}\ \ \tilde{V}]\in\mathbb{R}^{n\times n}$$
are orthogonal.
- Now, we write
$$U^TAV = \begin{bmatrix}\mathbf{u}^T\\ \tilde{U}^T\end{bmatrix}\cdot A\cdot[\mathbf{v}\ \ \tilde{V}]=\begin{bmatrix}\mathbf{u}^TA\mathbf{v} & \mathbf{u}^TA\tilde{V}\\ \tilde{U}^TA\mathbf{v} & \tilde{U}^TA\tilde{V}\end{bmatrix}$$
Since
$$\begin{aligned}\mathbf{u}^TA\mathbf{v} &= \frac{(A\mathbf{v})^T(A\mathbf{v})}{||A\mathbf{v}||_2}=\frac{||A\mathbf{v}||_2^2}{||A\mathbf{v}||_2}=||A\mathbf{v}||_2=||A||_2\equiv\sigma,\\ \tilde{U}^TA\mathbf{v} &= \tilde{U}^T\mathbf{u}||A\mathbf{v}||_2=0,\end{aligned}$$
we have
$$U^TAV = \begin{bmatrix}\sigma & 0\\ 0 & U_1\Sigma_1V_1^T\end{bmatrix}=\begin{bmatrix}1&0\\0&U_1\end{bmatrix}\begin{bmatrix}\sigma&0\\0&\Sigma_1\end{bmatrix}\begin{bmatrix}1&0\\0&V_1\end{bmatrix}^T,$$
or equivalently
$$A=\left(U\begin{bmatrix}1&0\\0&U_1\end{bmatrix}\right)\begin{bmatrix}\sigma&0\\0&\Sigma_1\end{bmatrix}\left(V\begin{bmatrix}1&0\\0&V_1\end{bmatrix}\right)^T. \tag{4.3}$$

Equation (4.3) is our desired decomposition. □

4.1.1. Algebraic Interpretation of the SVD

Let the SVD of A be $A = U\,\Sigma\,V^T$, with

$$\begin{aligned} U &= [\mathbf{u}_1 \quad \mathbf{u}_2 \quad \cdots \quad \mathbf{u}_n], \\ \Sigma &= \text{diag}(\sigma_1, \sigma_2, \cdots, \sigma_n), \\ V &= [\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_n], \end{aligned}$$

and σ_k be the **smallest** positive singular value. Since

$$A = U\,\Sigma\,V^T \;\Leftrightarrow\; AV = U\Sigma V^T V = U\Sigma,$$

we have

$$\begin{aligned} AV &= A[\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_n] = [A\mathbf{v}_1 \quad A\mathbf{v}_2 \quad \cdots \quad A\mathbf{v}_n] \\ &= [\mathbf{u}_1 \quad \cdots \quad \mathbf{u}_k \quad \cdots \quad \mathbf{u}_n] \begin{bmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_k & & \\ & & & \ddots & \\ & & & & 0 \end{bmatrix} \\ &= [\sigma_1\mathbf{u}_1 \quad \cdots \quad \sigma_k\mathbf{u}_k \quad \mathbf{0} \quad \cdots \quad \mathbf{0}] \end{aligned}$$

Therefore,

$$A = U\,\Sigma\,V^T \Leftrightarrow \begin{cases} A\mathbf{v}_j = \sigma_j \mathbf{u}_j, & j = 1, 2, \cdots, k \\ A\mathbf{v}_j = \mathbf{0}, & j = k+1, \cdots, n \end{cases} \tag{4.4}$$

Similarly, starting from $A^T = V\,\Sigma\,U^T$,

$$A^T = V\,\Sigma\,U^T \Leftrightarrow \begin{cases} A^T\mathbf{u}_j = \sigma_j \mathbf{v}_j, & j = 1, 2, \cdots, k \\ A^T\mathbf{u}_j = \mathbf{0}, & j = k+1, \cdots, n \end{cases} \tag{4.5}$$

Summary 4.6. It follows from (4.4) and (4.5) that

- $\{\sigma_j^2\}$, $j = 1, 2, \cdots, k$, are positive eigenvalues of $A^T A$.

  $$A^T A\,\mathbf{v}_j = A^T(\sigma_j \mathbf{u}_j) = \sigma_j^2 \mathbf{v}_j, \;\; j = 1, 2, \cdots, k. \tag{4.6}$$

  So, the singular values play the role of eigenvalues.

- Equation (4.6) gives how to find the singular values and the right singular vectors, while (4.4) shows a way to compute the left singular vectors.

- **(Dyadic decomposition)** The matrix $A \in \mathbb{R}^{m\times n}$, with $\text{rank}(A) = k \le n$, can be expressed as

  $$A = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T. \tag{4.7}$$

  This property has been utilized for various approximations and applications, e.g., by dropping singular vectors corresponding to *small* singular values.

4.1.2. Geometric Interpretation of the SVD

The matrix A maps an **orthonormal basis**

$$\mathcal{B}_1 = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n\}$$

of $\mathbb{R}^n$ onto a new "scaled" **orthogonal basis**

$$\mathcal{B}_2 = \{\sigma_1\mathbf{u}_1, \sigma_2\mathbf{u}_2, \cdots, \sigma_k\mathbf{u}_k\}$$

for a subspace of $\mathbb{R}^m$.

$$\mathcal{B}_1 = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n\} \xrightarrow{A} \mathcal{B}_2 = \{\sigma_1\mathbf{u}_1, \sigma_2\mathbf{u}_2, \cdots, \sigma_k\mathbf{u}_k\} \tag{4.8}$$

Consider a unit sphere $\mathcal{S}^{n-1}$ in $\mathbb{R}^n$:

$$\mathcal{S}^{n-1} = \Big\{\mathbf{x}\Big| \sum_{j=1}^{n} x_j^2 = 1\Big\}.$$

Then, $\forall \mathbf{x} \in \mathcal{S}^{n-1}$,

$$\begin{aligned} \mathbf{x} &= x_1\mathbf{v}_1 + x_2\mathbf{v}_2 + \cdots + x_n\mathbf{v}_n \\ A\mathbf{x} &= \sigma_1 x_1\mathbf{u}_1 + \sigma_2 x_2\mathbf{u}_2 + \cdots + \sigma_k x_k\mathbf{u}_k \\ &= y_1\mathbf{u}_1 + y_2\mathbf{u}_2 + \cdots + y_k\mathbf{u}_k, \quad (y_j = \sigma_j x_j) \end{aligned} \tag{4.9}$$

So, we have

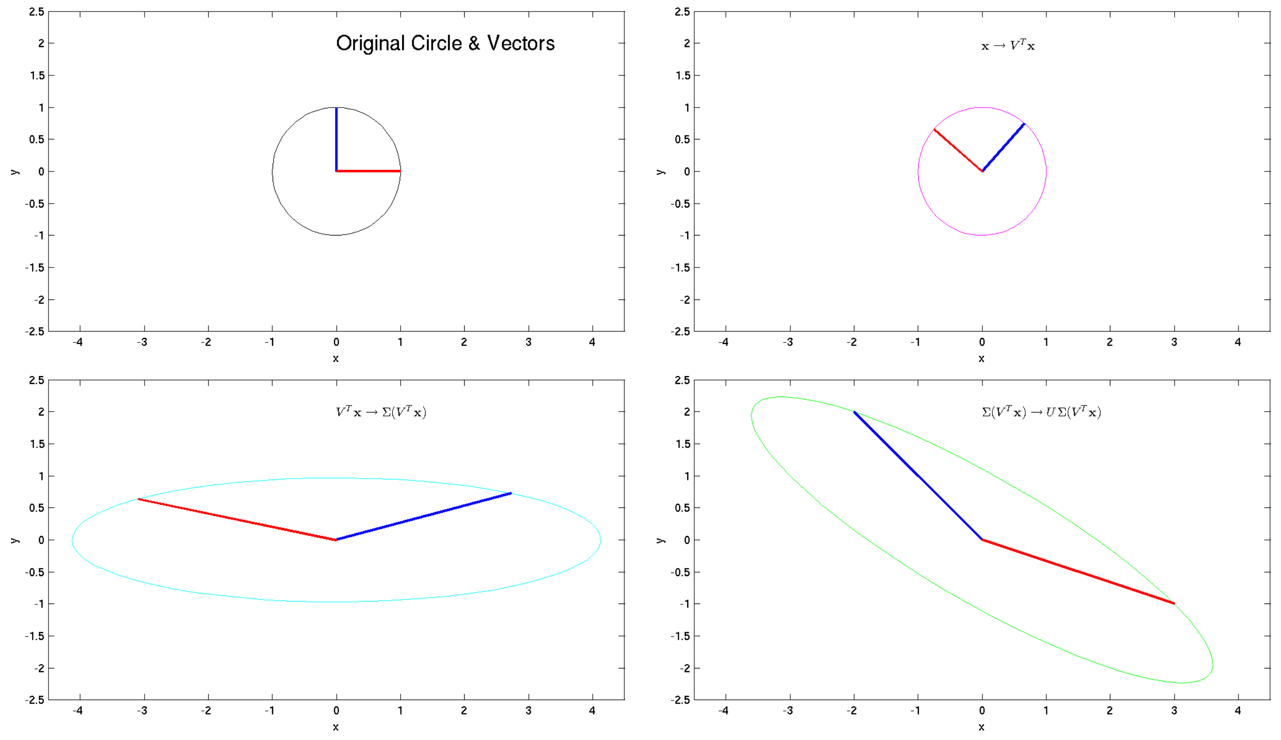
$$\begin{aligned} y_j = \sigma_j x_j \;&\Longleftrightarrow\; x_j = \frac{y_j}{\sigma_j} \\ \sum_{j=1}^{n} x_j^2 = 1 \text{ (sphere)} \;&\Longleftrightarrow\; \sum_{j=1}^{k} \frac{y_j^2}{\sigma_j^2} = \alpha \le 1 \text{ (ellipsoid)} \end{aligned} \tag{4.10}$$

Example 4.7. *We build the set $A \cdot \mathcal{S}^{n-1}$ by multiplying one factor of $A = U\Sigma V^T$ at a time. Assume for simplicity that $A \in \mathbb{R}^{2\times 2}$ and nonsingular. Let*

$$\begin{aligned} A &= \begin{bmatrix} 3 & -2 \\ -1 & 2 \end{bmatrix} = U\Sigma V^T \\ &= \begin{bmatrix} -0.8649 & 0.5019 \\ 0.5019 & 0.8649 \end{bmatrix} \begin{bmatrix} 4.1306 & 0 \\ 0 & 0.9684 \end{bmatrix} \begin{bmatrix} -0.7497 & 0.6618 \\ 0.6618 & 0.7497 \end{bmatrix} \end{aligned}$$

Then, for $\mathbf{x} \in \mathcal{S}^1$,

$$A\mathbf{x} = U\Sigma V^T\mathbf{x} = U\left(\Sigma(V^T\mathbf{x})\right)$$



In general,

- $V^T : \mathcal{S}^{n-1} \to \mathcal{S}^{n-1}$ (rotation in $\mathbb{R}^n$)
- $\Sigma : \mathbf{e}_j \mapsto \sigma_j\mathbf{e}_j$ (scaling from $\mathcal{S}^{n-1}$ to $\mathbb{R}^n$)
- $U : \mathbb{R}^n \to \mathbb{R}^m$ (rotation)

4.1.3. Properties of the SVD

$\boxed{\textbf{Theorem}}$ **4.8.** *Let* $A \in \mathbb{R}^{m\times n}$ *and* $p = \min(m,n)$. *Let* $A = U\Sigma V^T$ *be the SVD of* A, *with*

$$\sigma_1 \geq \cdots \geq \sigma_k > \sigma_{k+1} = \cdots = \sigma_p = 0.$$

Then,

$$\textit{1.}\ \begin{cases} \textit{\textbf{rank}}(A) &= k \\ \textit{\textbf{Null}}(A) &= \textit{\textbf{Span}}\{\mathbf{v}_{k+1}, \cdots, \mathbf{v}_n\} \\ \textit{\textbf{Range}}(A) &= \textit{\textbf{Span}}\{\mathbf{u}_1, \cdots, \mathbf{u}_k\} \\ A &= \displaystyle\sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \end{cases}$$

$$\textit{2.}\ \begin{cases} ||A||_2 &= \sigma_1 \quad \textbf{(See HW 1.)} \\ ||A||_F^2 &= \sigma_1^2 + \cdots + \sigma_k^2 \quad \textbf{(See HW 2.)} \\ \displaystyle\min_{\mathbf{x}\neq\mathbf{0}} \frac{||A\mathbf{x}||_2}{||\mathbf{x}||_2} &= \sigma_n \qquad (m \geq n) \\ \kappa_2(A) &= ||A||_2 \cdot ||A^{-1}||_2 = \dfrac{\sigma_1}{\sigma_n} \\ & \quad (\ \textit{\textbf{when}}\ \ m = n, \&\ \exists A^{-1}) \end{cases}$$

Theorem 4.9. *Let $A \in \mathbb{R}^{m\times n}$ with $m \ge n$. Then,*

$$\begin{aligned} &||A^T A||_2 = ||A||_2^2 \\ &\kappa_2(A^T A) = \kappa_2(A)^2. \quad \text{(when } A^{-1} \text{ exists)} \end{aligned} \tag{4.11}$$

Proof. Use the SVD of A to deduce the SVD of $A^T A$. □

Theorem 4.10. *Let $A \in \mathbb{R}^{m\times n}$, $m \ge n$, rank$(A) = n$, with singular values*

$$\sigma_1 \ge \sigma_2 \ge \cdots \sigma_n > 0.$$

Then

$$\begin{aligned} ||(A^T A)^{-1}||_2 &= \sigma_n^{-2}, \\ ||(A^T A)^{-1}A^T||_2 &= \sigma_n^{-1}, \\ ||A(A^T A)^{-1}||_2 &= \sigma_n^{-1}, \\ ||A(A^T A)^{-1}A^T||_2 &= 1. \end{aligned} \tag{4.12}$$

Definition 4.11. $(A^T A)^{-1}A^T$ *is called the* **pseudoinverse** *of A, while $A(A^T A)^{-1}$ is called the* **pseudoinverse** *of A^T. Let $A = U\Sigma V^T$ be the SVD of A. Then*

$$(A^T A)^{-1}A^T = V\Sigma^{-1}U^T \equiv A^+. \tag{4.13}$$

$\boxed{\textbf{Theorem}}$ **4.12.** *Let $A \in \mathbb{R}^{m\times n}$ with **rank**$(A) = r > 0$. Let $A = U\Sigma V^T$ be the SVD of A, with singular values*

$$\sigma_1 \geq \cdots \geq \sigma_r > 0.$$

*Define, for $k = 1, \cdots, r-1$, the **truncated SVD***

$$A_k = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T \quad \text{(sum of rank-1 matrices)}. \tag{4.14}$$

*Then, **rank**$(A_k) = k$ and*

$$\begin{aligned} ||A - A_k||_2 &= \min\{||A - B||_2 \,\big|\, \textbf{rank}(B) \leq k\} \\ &= \sigma_{k+1}, \\ ||A - A_k||_F^2 &= \min\{||A - B||_F^2 \,\big|\, \textbf{rank}(B) \leq k\} \\ &= \sigma_{k+1}^2 + \cdots + \sigma_r^2. \end{aligned} \tag{4.15}$$

That is, of all matrices of rank $\leq k$, A_k is closest to A.

Note: The truncated matrix A_k can be written as

$$A_k = U\Sigma_k V^T, \tag{4.16}$$

where $\Sigma_k = \text{diag}(\sigma_1, \cdots, \sigma_k, 0, \cdots, 0)$.

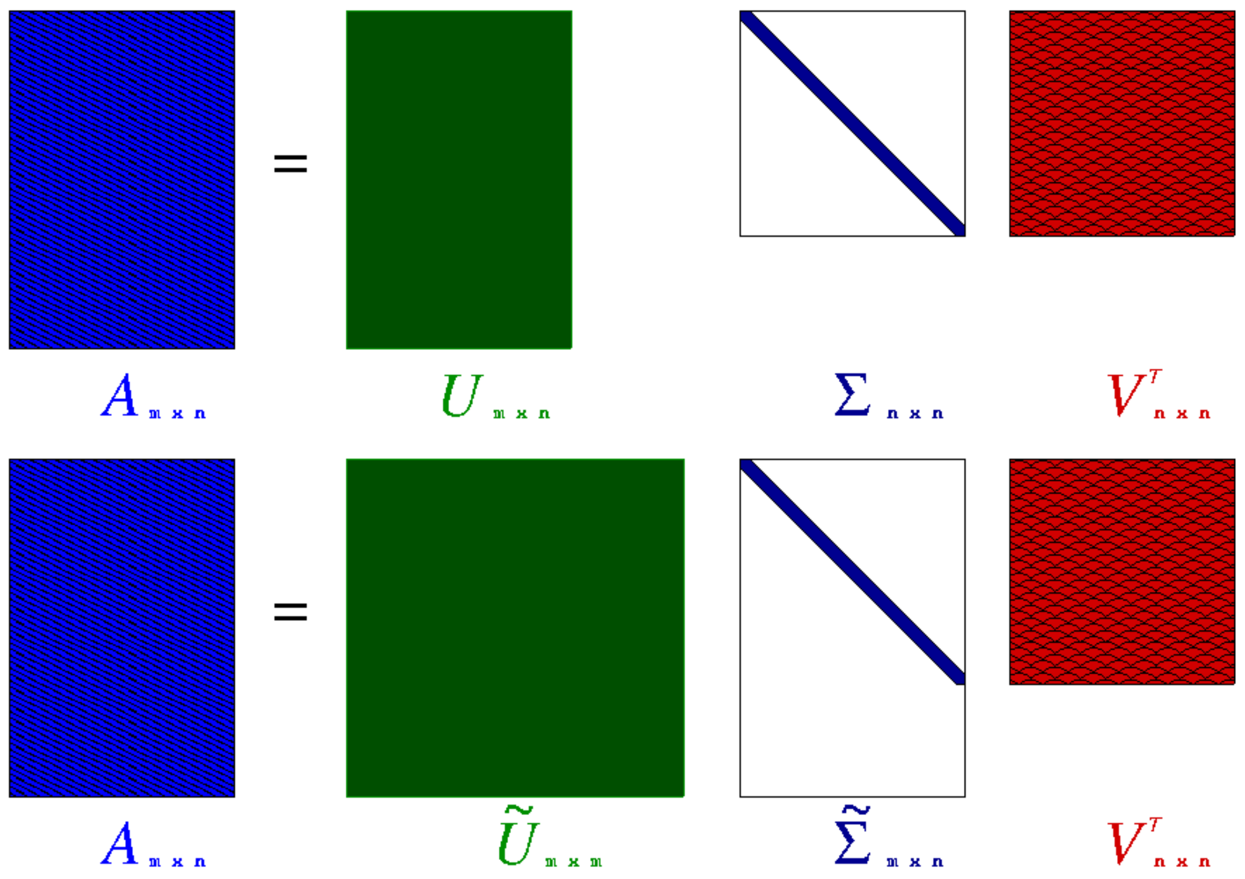
Corollary 4.13. *Suppose $A \in \mathbb{R}^{m\times n}$ has full rank. Thus rank$(A) = p$, where $p = \min(m, n)$. Let $\sigma_1 \geq \cdots \geq \sigma_p$ be the singular values of A. Let $B \in \mathbb{R}^{m\times n}$ satisfy*

$$||A - B||_2 < \sigma_p.$$

Then B also has full rank.

Full SVD

- For $A \in \mathbb{R}^{m\times n}$,
$$A = U\Sigma V^T \iff U^T AV = \Sigma,$$
where $U \in \mathbb{R}^{m\times n}$ and $\Sigma, V \in \mathbb{R}^{n\times n}$.
- Expand
$$\begin{aligned} U &\to \tilde{U} = [U \;\; U_2] \in \mathbb{R}^{m\times m}, \quad \text{(orthogonal)} \\ \Sigma &\to \tilde{\Sigma} = \begin{bmatrix} \Sigma \\ O \end{bmatrix} \in \mathbb{R}^{m\times n}, \end{aligned}$$
where O is an $(m-n)\times n$ zero matrix.
- Then,
$$\tilde{U}\tilde{\Sigma}V^T = [U \;\; U_2] \begin{bmatrix} \Sigma \\ O \end{bmatrix} V^T = U\Sigma V^T = A \tag{4.17}$$



Proposition 4.14. There are important relation between the SVD of a matrix A and the **Schur decompositions** of the **symmetric** matrices

$$A^T A,\ \ AA^T,\ \text{ and } \begin{bmatrix} O & A^T \\ A & O \end{bmatrix}.$$

Let $A \in \mathbb{R}^{m\times n}$, $m \geq n$. If $U^T AV = \text{diag}(\sigma_1, \cdots, \sigma_n)$ is the SVD of A, then

$$\begin{aligned} V^T(A^T A)V &= \text{diag}(\sigma_1^2, \cdots, \sigma_n^2) \in \mathbb{R}^{n\times n} \\ U^T(AA^T)U &= \text{diag}(\sigma_1^2, \cdots, \sigma_n^2, \underbrace{0 \cdots, 0}_{m-n}) \in \mathbb{R}^{m\times m} \end{aligned}$$

Separate U as

$$U = [\underbrace{U_1}_{n}\ \underbrace{U_2}_{m-n}],$$

and define

$$Q \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} V & V & O \\ U_1 & -U_1 & \sqrt{2}U_2 \end{bmatrix} \in \mathbb{R}^{(m+n)\times(m+n)}.$$

Then,

$$Q^T \begin{bmatrix} O & A^T \\ A & O \end{bmatrix} Q = \text{diag}(\sigma_1, \cdots, \sigma_n, -\sigma_1, \cdots, -\sigma_n, \underbrace{0 \cdots, 0}_{m-n}) \tag{4.18}$$

Theorem 4.15. *Let $A = U\Sigma V^T$ be the SVD of $A \in \mathbb{R}^{m\times n}$ ($m \geq n$).*

1. ($m = n$) *Suppose that A is* **symmetric**, *with eigenvalues λ_i and orthonormal e-vectors u_i. In other words,*

$$A = U\Lambda U^T$$

is an **eigendecomposition**, *with $\Lambda = diag(\lambda_1, \cdots, \lambda_n)$. Then an SVD of A is $A = U\Sigma V^T$, where*

$$\sigma_i = |\lambda_i| \text{ and } \mathbf{v}_i = sign(\lambda_i)\mathbf{u}_i, \text{ where } sign(0) = 1.$$

2. *The eigenvalues of the* **symmetric** *matrix A^TA are σ_i^2. The right singular vectors v_i are corresponding orthonormal eigenvectors.*
3. *The eigenvalues of the* **symmetric** *matrix AA^T are σ_i^2 and $m - n$ zeros. The left singular vectors $\mathbf{u}_i$ are corresponding orthonormal eigenvectors for the e-values σ_i^2. One can take any $m - n$ other orthogonal vectors as eigenvectors for the eigenvalue 0.*
4. *Let $H = \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}$. A is square and $A = U\Sigma V^T$ is the SVD of A. $\Sigma = diag(\sigma_1, \cdots, \sigma_n)$, $U = [\mathbf{u}_1\ \mathbf{u}_2\ \cdots\ \mathbf{u}_n]$, and $V = [\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_n]$. Then the $2n$ e-values of H are $\pm\sigma_i$, with e-vectors $\frac{1}{\sqrt{2}}\begin{bmatrix} \mathbf{v}_i \\ \pm\mathbf{u}_i \end{bmatrix}$.*
5. *If A has full rank, the solution of $\min_{\mathbf{x}} ||A\mathbf{x} - \mathbf{b}||_2$ is $\mathbf{x} = V\Sigma^{-1}U^T\mathbf{b}$.*

4.1.4. Computation of the SVD

For $A \in \mathbb{R}^{m\times n}$, the procedure is as follows.

1. Form A^TA (A^TA – **covariance matrix** of A).
2. Find the eigendecomposition of A^TA by orthogonalization process, i.e., $\Lambda = \text{diag}(\lambda_1, \cdots, \lambda_n)$,
$$A^TA = V\Lambda V^T,$$
where $V = [\mathbf{v}_1 \quad \cdots \quad \mathbf{v}_n]$ is orthogonal, i.e., $V^TV = I$.
3. Sort the eigenvalues according to their magnitude and let
$$\sigma_j = \sqrt{\lambda_j}, \;\; j = 1, 2, \cdots, n.$$
4. Form the U matrix as follows,
$$\mathbf{u}_j = \frac{1}{\sigma_j} A\mathbf{v}_j, \quad j = 1, 2, \cdots, r.$$
If necessary, pick up the remaining columns of U so it is orthogonal. (These additional columns must be in Null(AA^T).)
5. $A = U\Sigma V^T = [\mathbf{u}_1 \; \cdots \; \mathbf{u}_r \; \cdots \; \mathbf{u}_n]\, \text{diag}(\sigma_1, \cdots, \sigma_r, 0, \cdots, 0) \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_n^T \end{bmatrix}$

Lemma 4.16. *Let $A \in \mathbb{R}^{n\times n}$ be symmetric. Then (a) all the eigenvalues of A are real and (b) eigenvectors corresponding to distinct eigenvalues are orthogonal.*

Proof. See Homework 3. □

Example 4.17. *Find the SVD for* $A = \begin{bmatrix} 1 & 2 \\ -2 & 1 \\ 3 & 2 \end{bmatrix}$.

Solution.

1. $A^TA = \begin{bmatrix} 14 & 6 \\ 6 & 9 \end{bmatrix}$.

2. Solving $\det(A^TA - \lambda I) = 0$ gives the eigenvalues of A^TA

$$\lambda_1 = 18 \text{ and } \lambda_2 = 5,$$

of which corresponding eigenvectors are

$$\tilde{\mathbf{v}}_1 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}, \tilde{\mathbf{v}}_2 = \begin{bmatrix} -2 \\ 3 \end{bmatrix}. \Longrightarrow V = \begin{bmatrix} \frac{3}{\sqrt{13}} & -\frac{2}{\sqrt{13}} \\ \frac{2}{\sqrt{13}} & \frac{3}{\sqrt{13}} \end{bmatrix}$$

3. $\sigma_1 = \sqrt{\lambda_1} = \sqrt{18} = 3\sqrt{2}$, $\sigma_2 = \sqrt{\lambda_2} = \sqrt{5}$. So

$$\Sigma = \begin{bmatrix} \sqrt{18} & 0 \\ 0 & \sqrt{5} \end{bmatrix}$$

4. $\mathbf{u}_1 = \frac{1}{\sigma_1} A\mathbf{v}_1 = \frac{1}{\sqrt{18}} A \begin{bmatrix} \frac{3}{\sqrt{13}} \\ \frac{2}{\sqrt{13}} \end{bmatrix} = \frac{1}{\sqrt{18}} \frac{1}{\sqrt{13}} \begin{bmatrix} 7 \\ -4 \\ 13 \end{bmatrix} = \begin{bmatrix} \frac{7}{\sqrt{234}} \\ -\frac{4}{\sqrt{234}} \\ \frac{13}{\sqrt{234}} \end{bmatrix}$

   $\mathbf{u}_2 = \frac{1}{\sigma_2} A\mathbf{v}_2 = \frac{1}{\sqrt{5}} A \begin{bmatrix} \frac{-2}{\sqrt{13}} \\ \frac{3}{\sqrt{13}} \end{bmatrix} = \frac{1}{\sqrt{5}} \frac{1}{\sqrt{13}} \begin{bmatrix} 4 \\ 7 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{4}{\sqrt{65}} \\ \frac{7}{\sqrt{65}} \\ 0 \end{bmatrix}$.

5. $A = U\Sigma V^T = \begin{bmatrix} \frac{7}{\sqrt{234}} & \frac{4}{\sqrt{65}} \\ -\frac{4}{\sqrt{234}} & \frac{7}{\sqrt{65}} \\ \frac{13}{\sqrt{234}} & 0 \end{bmatrix} \begin{bmatrix} \sqrt{18} & 0 \\ 0 & \sqrt{5} \end{bmatrix} \begin{bmatrix} \frac{3}{\sqrt{13}} & \frac{2}{\sqrt{13}} \\ -\frac{2}{\sqrt{13}} & \frac{3}{\sqrt{13}} \end{bmatrix}$

Example 4.18. *Find the pseudoinverse of A,*

$$A^+ = (A^TA)^{-1}A^T = V\Sigma^{-1}U^T,$$

when $A = \begin{bmatrix} 1 & 2 \\ -2 & 1 \\ 3 & 2 \end{bmatrix}$.

Solution. From Example 4.17, we have

$$A \;=\; U\Sigma V^T = \begin{bmatrix} \frac{7}{\sqrt{234}} & \frac{4}{\sqrt{65}} \\ -\frac{4}{\sqrt{234}} & \frac{7}{\sqrt{65}} \\ \frac{13}{\sqrt{234}} & 0 \end{bmatrix} \begin{bmatrix} \sqrt{18} & 0 \\ 0 & \sqrt{5} \end{bmatrix} \begin{bmatrix} \frac{3}{\sqrt{13}} & \frac{2}{\sqrt{13}} \\ -\frac{2}{\sqrt{13}} & \frac{3}{\sqrt{13}} \end{bmatrix}$$

Thus,

$$\begin{aligned} A^+ \;&=\; V\Sigma^{-1}U^T = \begin{bmatrix} \frac{3}{\sqrt{13}} & -\frac{2}{\sqrt{13}} \\ \frac{2}{\sqrt{13}} & \frac{3}{\sqrt{13}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{18}} & 0 \\ 0 & \frac{1}{\sqrt{5}} \end{bmatrix} \begin{bmatrix} \frac{7}{\sqrt{234}} & -\frac{4}{\sqrt{234}} & \frac{13}{\sqrt{234}} \\ \frac{4}{\sqrt{65}} & \frac{7}{\sqrt{65}} & 0 \end{bmatrix} \\ &=\; \begin{bmatrix} -\frac{1}{30} & -\frac{4}{15} & \frac{1}{6} \\ \frac{11}{45} & \frac{13}{45} & \frac{1}{9} \end{bmatrix} \end{aligned}$$

Computer implementation [10]

$$\underset{A}{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}} \xrightarrow{\text{Phase 1}} \underset{B}{\begin{bmatrix} \times & \times & & \\ & \times & \times & \\ & & \times & \times \\ & & & \times \\ & & & \\ & & & \end{bmatrix}} \xrightarrow{\text{Phase 2}} \underset{\Sigma}{\begin{bmatrix} \times & & & \\ & \times & & \\ & & \times & \\ & & & \times \\ & & & \\ & & & \end{bmatrix}}.$$

- Algorithm is extremely stable.
- The SVD of $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$ is $A = U\Sigma V^T$:
  - Computation of U, V and Σ: $4m^2n + 8mn^2 + 9n^3$.
  - Computation of V, and Σ: $4mn^2 + 8n^3$.
- Matlab: **[U,S,V] = svd(A)**
- Mathematica:
  {U, S, V} = SingularValueDecomposition[A]
- Lapack: **dgelss**
- **Eigenvalue** problems.

4.1.5. Numerical rank

In the absence of roundoff errors and uncertainties in the data, the SVD reveals the rank of the matrix. Unfortunately the presence of errors makes rank determination problematic. For example, consider

$$A = \begin{bmatrix} 1/3 & 1/3 & 2/3 \\ 2/3 & 2/3 & 4/3 \\ 1/3 & 2/3 & 3/3 \\ 2/5 & 2/5 & 4/5 \\ 3/5 & 1/5 & 4/5 \end{bmatrix} \tag{4.19}$$

- Obviously A is of rank 2, as its third column is the sum of the first two.
- Matlab “svd" (with IEEE double precision) produces

$$\sigma_1 = 2.5987, \quad \sigma_2 = 0.3682, \quad \text{and } \sigma_3 = 8.6614 \times 10^{-17}.$$

- What is the rank of A, 2 or 3?
  What if σ_3 is in $\mathcal{O}(10^{-13})$?
- For this reason we must introduce a **threshold** T. Then we say that A has **numerical rank** k if A has k singular values larger than T, that is,

$$\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_k > T \ge \sigma_{k+1} \ge \cdots \tag{4.20}$$

Matlab

- Matlab has a “rank" command, which computes the numerical rank of the matrix with a default threshold

$$T = 2\max\{m,n\}\,\epsilon\,||A||_2 \tag{4.21}$$

  where ϵ is the unit roundoff error.

- In Matlab, the unit roundoff error can be found from the parameter “eps"

$$\textbf{eps} = 2^{-52} = 2.2204\times 10^{-16}.$$

- For the matrix A in (4.19),

$$T = 2\cdot 5\cdot \textbf{eps}\cdot 2.5987 = 5.7702\times 10^{-15}$$

  and therefore `rank(A)=2`.

See Homework 4 for an exercise with Matlab.

4.2. Applications of the SVD

- Solving (rank-deficient) LS problems
  (pseudoinverse $A^+ = V\Sigma^+U^T$, when $A = U\Sigma V^T$)
- Low-rank matrix approximation
  - Image/Data compression
  - Denoising
- Signal processing [1, 2, 4]
- Other applications
  - Principal component analysis
    (e.g., signal processing and pattern recognition)
  - Numerical weather prediction
  - Reduced order modeling [25]
  - Inverse problem theory [20]

4.2.1. Image compression

- $A \in \mathbb{R}^{m\times n}$ is a sum of rank-1 matrices:

$$V = [\mathbf{v}_1, \cdots, \mathbf{v}_n], \quad U = [\mathbf{u}_1, \cdots, \mathbf{u}_n],$$
$$A = U\Sigma V^T = \sum_{i=1}^{n} \sigma_i \mathbf{u}_i \mathbf{v}_i^T.$$

- The approximation

$$A_k = U\Sigma_k V^T = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

  is closest to A among matrices of rank$\leq k$, and

$$||A - A_k||_2 = \sigma_{k+1}.$$

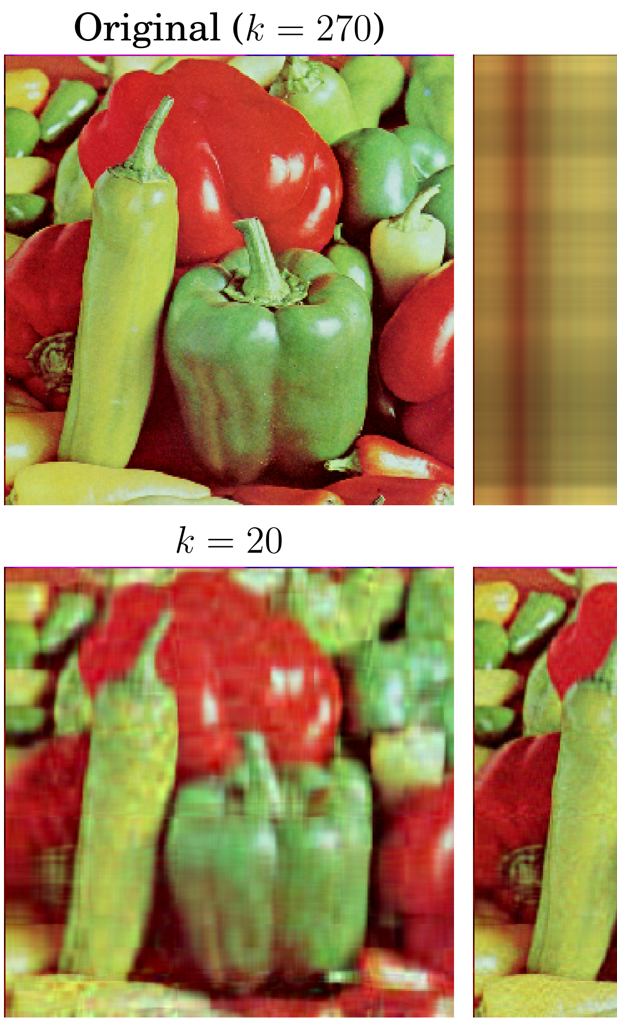
- It only takes $m \cdot k + n \cdot k = (m+n)\cdot k$ words to store $\mathbf{u}_1$ through $\mathbf{u}_k$, and $\sigma_1\mathbf{v}_1$ through $\sigma_k\mathbf{v}_k$, from which we can reconstruct A_k.
- We use A_k as our compressed images, stored using $(m+n)\cdot k$ words.

- Matlab code to demonstrate the SVD compression of images:

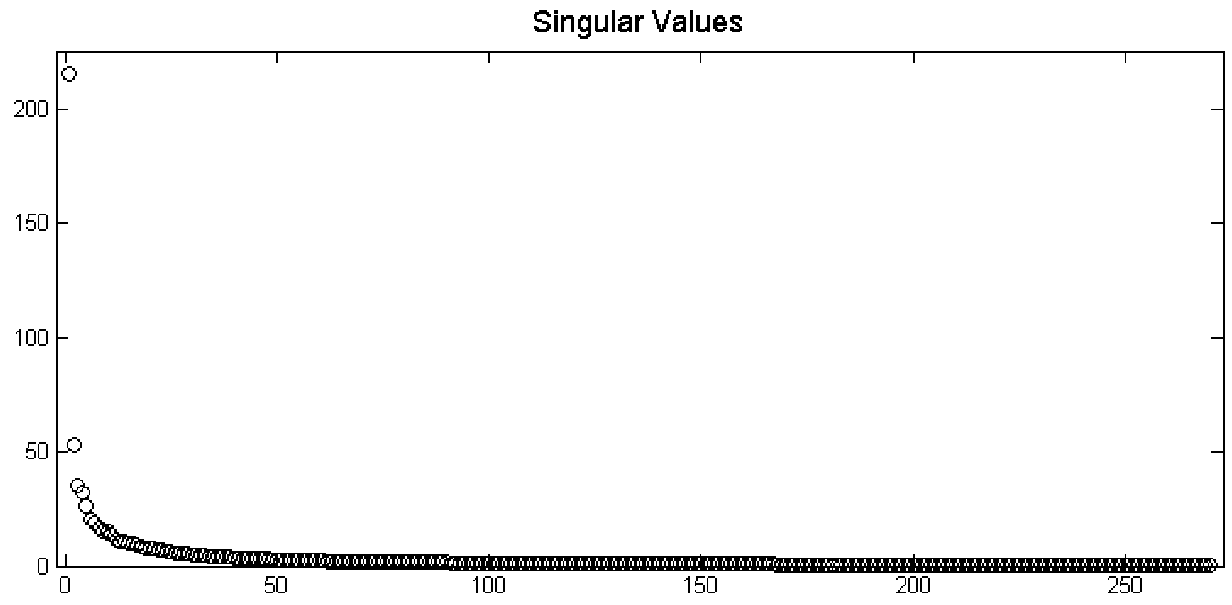
```
img = imread('Peppers.png'); [m,n,d]=size(img);
[U,S,V] = svd(reshape(im2double(img),m,[]));
%%---- select k <= p=min(m,n)
k = 20;
img_k = U(:,1:k)*S(1:k,1:k)*V(:,1:k)';
img_k = reshape(img_k,m,n,d);
figure, imshow(img_k)
```

The “Peppers" image is in $[270, 270, 3] \in \mathbb{R}^{270\times 810}$.

Image compression using k singular values



Peppers: Singular values



Peppers: Compression quality

$$\text{PSNR (dB)} = \begin{cases} 13.7 & \textbf{when } k = 1, \\ 20.4 & \textbf{when } k = 10, \\ 23.7 & \textbf{when } k = 20, \\ 29.0 & \textbf{when } k = 50, \\ 32.6 & \textbf{when } k = 100, \\ 37.5 & \textbf{when } k = 150, \end{cases}$$

where PSNR is "Peak Signal-to-Noise Ratio."

Peppers: Storage: It requires $(m+n)\cdot k$ words. For example, when $k = 50$,

$$(m+n)\cdot k = (270+810)\cdot 50 = \boxed{\mathbf{54{,}000}}, \qquad \textbf{(4.22)}$$

which is approximately **a quarter** the full storage space

$$270 \times 270 \times 3 = \boxed{\mathbf{218{,}700}}.$$

4.2.2. Rank-deficient least-squares problems

The LS problem: For the system of equations

$$A\mathbf{x} = \mathbf{b}, \tag{4.23}$$

where $A \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$, find the best-fitting solution $\widehat{\mathbf{x}}$ as a solution of

$$\min_{\mathbf{x}\in\mathbb{R}^n} ||A\mathbf{x} - \mathbf{b}||_2. \tag{4.24}$$

- If $m > n$, then the system (4.23) is overdetermined, and we cannot expect to find an exact solution. The solution can be obtained through the LS problem (4.24).
- The solution of the LS problem is sometimes not unique, for example, when
$$\textbf{rank}(A) = r < \min\{m, n\}.$$
In this case, we may consider the following additional problem:

  Of all the $\mathbf{x} \in \mathbb{R}^n$ *that minimizes* $||A\mathbf{x} - \mathbf{b}||_2$*, find the one for which* $||\mathbf{x}||_2$ *is minimized.*

  This problem always has a unique solution, the **minimum-norm solution**.

Proposition 4.19. (Rank-deficient LS problems, Proposition 4.3 on page 126). *Let $A \in \mathbb{R}^{m\times n}$, $m \ge n$, with rank$(A) = k < n$. Then there exists an $(n-k)$-dimensional set of vectors $\mathbf{x}$ that minimize*

$$||A\mathbf{x} - \mathbf{b}||_2.$$

Proposition 4.20. *Let $\sigma_{\min} = \sigma_{\min}(A)$, the smallest singular value of A. Assume $\sigma_{\min} > 0$. Then*

1. *If $\mathbf{x}$ minimize $||A\mathbf{x} - \mathbf{b}||_2$, then $||\mathbf{x}||_2 \ge |\mathbf{u}_n^T\mathbf{b}|/\sigma_{\min}$, where $\mathbf{u}_n$ is the last column of U in $A = U\Sigma V^T$.*
2. *Changing $\mathbf{b}$ to $\mathbf{b} + \delta\mathbf{b}$ can change $\mathbf{x}$ to $\mathbf{x} + \delta\mathbf{x}$, where $||\delta\mathbf{x}||_2$ is as large as $||\delta\mathbf{b}||_2/\sigma_{\min}$.*

In other words, if A is nearly rank deficient ($\sigma_{\min}$ is small), then the solution $\mathbf{x}$ is ill-conditioned and possibly very large.

Proof. Part 1:

$$\mathbf{x} = A^{+}\mathbf{b} = V\Sigma^{-1}U^T\mathbf{b}.$$

So

$$||\mathbf{x}||_2 = ||\Sigma^{-1}U^T\mathbf{b}||_2 \ge |\left(\Sigma^{-1}U^T\mathbf{b}\right)_n| = |\mathbf{u}_n^T\mathbf{b}|/\sigma_{\min}.$$

Part 2: choose $\delta\mathbf{b}$ parallel to $\mathbf{u}_n$. $\square$

Proposition 4.21. *When $A \in \mathbb{R}^{m\times n}$ is exactly singular, then*

$$\mathbf{x} = \textit{arg}\min_{\mathbf{x}\in\mathbb{R}^n} ||A\mathbf{x}-\mathbf{b}||_2$$

can be characterized as follows.
Let $A = U\Sigma V^T$ have rank $r < n$, and write the SVD of A as

$$A = [U_1\ \ U_2]\begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix}[V_1\ \ V_2] = U_1\Sigma_1V_1^T, \tag{4.25}$$

where Σ_1 is $r\times r$ and nonsingular and U_1 and V_1 have r columns. Let $\sigma = \sigma_{\min}(\Sigma_1)$, the smallest nonzero singular value of A. Then

1. *All solutions $\mathbf{x}$ can be written*

$$\mathbf{x} \ = \ V_1\Sigma_1^{-1}U_1^T\mathbf{b} + V_2\mathbf{z} \quad (= A^+\mathbf{b}), \tag{4.26}$$

where $\mathbf{z}$ is an arbitrary vector.

2. *The solution $\mathbf{x}$ has a minimal norm $||\mathbf{x}||_2$ precisely when $\mathbf{z} = \mathbf{0}$, in which case*

$$\mathbf{x} = V_1\Sigma_1^{-1}U_1^T\mathbf{b} \quad \textit{and} \quad ||\mathbf{x}||_2 \le ||\mathbf{b}||_2/\sigma. \tag{4.27}$$

3. *Changing $\mathbf{b}$ to $\mathbf{b}+\delta\mathbf{b}$ can change the minimal norm solution $\mathbf{x}$ by at most $||\delta\mathbf{b}||_2/\sigma$.*

In other words, the norm and condition number of the unique minimal norm solution $\mathbf{x}$ depend on the smallest nonzero singular value of A.

Solving LS problems by the SVD:
Let $A \in \mathbb{R}^{m\times n}$, $m \geq n$, have rank $r \leq n$. Let $A = U\Sigma V^T$ be the SVD of A.

- Because U is orthogonal,

$$||A\mathbf{x}-\mathbf{b}||_2 = ||U^T(A\mathbf{x}-\mathbf{b})||_2 = ||\Sigma(V^T\mathbf{x}) - U^T\mathbf{b}||_2.$$

- Letting $\mathbf{c} = U^T\mathbf{b}$ and $\mathbf{y} = V^T\mathbf{x}$, we have

$$||A\mathbf{x}-\mathbf{b}||_2^2 \;=\; ||\Sigma\mathbf{y}-\mathbf{c}||_2^2 = \sum_{j=1}^{r} |\sigma_j y_j - c_j|^2 + \sum_{j=r+1}^{m} |c_j|^2 \tag{4.28}$$

  It is clear that (4.28) is minimized **when and only when**

$$y_j = \frac{c_j}{\sigma_j}, \quad j = 1, \cdots, r. \tag{4.29}$$

- **(When $r < n$)**: It is clear to see that $||\mathbf{y}||_2$ is minimized when and only when

$$y_{r+1} = \cdots = y_n = 0.$$

- Since $\mathbf{x} = V\mathbf{y}$, we have $||\mathbf{x}||_2 = ||\mathbf{y}||_2$. Thus $||\mathbf{x}||_2$ is minimized when and only when $||\mathbf{y}||_2$ is. This proves that the LS problem has exactly one minimum-norm solution.

Solving LS problems by the SVD:
It is useful to repeat the development using partitioned matrices. Let $A \in \mathbb{R}^{m\times n}$, $m \geq n$, have rank $r \leq n$. Let $A = U\Sigma V^T$ be the SVD of A given as in (4.25):

$$A = [U_1 \;\; U_2] \begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix} [V_1 \;\; V_2] = U_1\Sigma_1 V_1^T, \tag{4.30}$$

where Σ_1 is $r \times r$ and nonsingular and U_1 and V_1 have r columns.

- Let, with the corresponding partitioning,

$$\mathbf{c} = U^T\mathbf{b} = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{bmatrix} \text{ and } \mathbf{y} = V^T\mathbf{x} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix}.$$

- Then,

$$\Sigma\mathbf{y} - \mathbf{c} = \begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} - \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{bmatrix} = \begin{bmatrix} \Sigma_1\mathbf{y}_1 - \mathbf{c}_1 \\ -\mathbf{c}_2 \end{bmatrix}.$$

- So

$$||A\mathbf{x} - \mathbf{b}||_2^2 \;=\; ||\Sigma\mathbf{y} - \mathbf{c}||_2^2 = ||\Sigma_1\mathbf{y}_1 - \mathbf{c}_1||_2^2 + ||\mathbf{c}_2||_2^2. \tag{4.31}$$

- Equation (4.31) is minimized when and only when

$$\mathbf{y}_1 = \Sigma_1^{-1}\mathbf{c}_1. \tag{4.32}$$

- For the minimum-norm solution, take $\mathbf{y}_2 = 0$.

Summary 4.22. (Solving LS problems by the SVD). *For solving the LS problem*

$$\min_{\mathbf{x}\in\mathbb{R}^n} ||A\mathbf{x}-\mathbf{b}||_2, \quad A\in\mathbb{R}^{m\times n},$$

we summarize the above SVD procedure as follows.

1. *Find the SVD of A: $A = U\Sigma V^T$ and rank$(A) = r$.*
2. *Calculate $\begin{bmatrix}\mathbf{c}_1\\ \mathbf{c}_2\end{bmatrix} = \mathbf{c} = U^T\mathbf{b}$, where $\mathbf{c}_1 \in \mathbb{R}^r$.*
3. *Let $\mathbf{y}_1 = \Sigma_1^{-1}\mathbf{c}_1$, where $\Sigma_1 = \Sigma(1:r, 1:r)$.*
4. *If $r < n$, choose $\mathbf{y}_2 \in \mathbb{R}^{n-r}$ arbitrarily.*
   (For minimum-norm solution, take $\mathbf{y}_2 = 0$.)
5. *Let $\mathbf{y} = \begin{bmatrix}\mathbf{y}_1\\ \mathbf{y}_2\end{bmatrix} \in \mathbb{R}^n$.*
6. *Let $\mathbf{x} = V\mathbf{y} \in \mathbb{R}^n$.*

Note:

- The above procedure is equivalent to solve the LS problem by the pseudoinverse of A, particular when A has full rank.
- The minimum-norm solution reads: $\mathbf{x} = V_1\mathbf{y}_1$.

Example 4.23. *Use the pseudoinverse to solve the LS problem* $\min ||A\mathbf{x}-\mathbf{b}||_2$*, where*

$$A = \begin{bmatrix} 1 & 2 \\ -1 & 1 \\ 1 & 2 \end{bmatrix} \quad \textit{and} \quad \mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

Solution. The eigenvalues of ***$A^T A$*** are

$$\lambda_1 = 3(2+\sqrt{2}), \quad \lambda_2 = 3(2-\sqrt{2}).$$

If $\mathbf{v}_1$ and $\mathbf{v}_2$ are the corresponding eigenvectors of $A^T A$, then

$$\mathbf{u}_j = \frac{1}{\sigma_j} A\mathbf{v}_j = \frac{1}{\sqrt{\lambda_j}} A\mathbf{v}_j, \quad j = 1, 2.$$

The SVD of A reads

$$A \;=\; U\Sigma V^T = \begin{bmatrix} \frac{1+\sqrt{2}}{2\sqrt{3}} & \frac{1-\sqrt{2}}{2\sqrt{3}} \\ \frac{\sqrt{2}-1}{\sqrt{6}} & \frac{\sqrt{2}+1}{\sqrt{6}} \\ \frac{1+\sqrt{2}}{2\sqrt{3}} & \frac{1-\sqrt{2}}{2\sqrt{3}} \end{bmatrix} \begin{bmatrix} \sqrt{\lambda_1} & 0 \\ 0 & \sqrt{\lambda_2} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}-1}{\sqrt{4-2\sqrt{2}}} & -\frac{\sqrt{2}+1}{\sqrt{4+2\sqrt{2}}} \\ \frac{1}{\sqrt{4-2\sqrt{2}}} & \frac{1}{\sqrt{4+2\sqrt{2}}} \end{bmatrix}$$

Thus

$$A^+ \;=\; V\Sigma^{-1}U^T = \begin{bmatrix} \frac{1}{6} & -\frac{2}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & \frac{1}{6} \end{bmatrix}$$

and therefore

$$\mathbf{x} = A^+\mathbf{b} = \begin{bmatrix} \frac{1}{6} & -\frac{2}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & \frac{1}{6} \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} -\frac{2}{3} \\ \frac{4}{3} \end{bmatrix}$$

The corresponding least L^2-error is $\sqrt{2}$, that is,

$$||A\mathbf{x}-\mathbf{b}||_2 = \min_{\mathbf{y}} ||A\mathbf{y}-\mathbf{b}||_2 = \sqrt{2}.$$

4.2.3. Principal component analysis (PCA)

Definition 4.24. Principal component analysis *is a statistical procedure for data* (a set of observations) *that*

(a) uses an **orthogonal transformation** *and*

(b) converts **data of possibly correlated variables** *into* **a set of values of linearly uncorrelated variables** *called* **principal components**.

PCA

- #{principal components} $<$ #{original variables}.
- The transformation is defined in such a way that
  - the first principal component has the largest possible variance,
  - and each succeeding component has the highest variance possible under the constraint that it is orthogonal to (i.e., uncorrelated with) the preceding components.
- Principal components are guaranteed to be independent if the data set is jointly normally distributed.

In other words:

- PCA is a mathematical algorithm that **reduces the dimensionality of the data** while retaining most of the variation in the data set.
- It accomplishes this reduction by identifying **directions**, called **principal components**, along which the variation in the data is maximal.
- By using a few components, each sample can be represented by relatively few numbers instead of by values for thousands of variables.

Example 4.25. *Figure 4.1 below indicates an oil-split zone. Find the semi-axes of the* **minimum-volume enclosing ellipsoid** *(MVEE) or* **equal-volume fitting ellipsoid** *(EVFE) of the oil-splitting.*

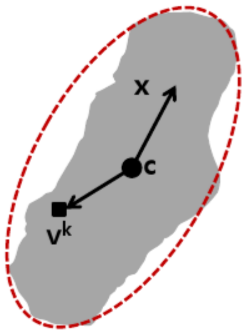


Figure 4.1: An oil-split zone.

Solution. Let the acquired data set (scattered points) be

$$\mathbf{v}^k = (v_{k,1}, v_{k,2}), \quad k = 1, \cdots, m.$$

Let the center of the data points be

$$\mathbf{c} = (c_1, c_2) = \frac{1}{m}(\mathbf{v}^1 + \cdots + \mathbf{v}^m).$$

Then, the direction of the major semi-axis must be the solution of

$$\max_{\mathbf{x}\in\mathbb{R}^2} \sum_{k=1}^{m} |(\mathbf{v}^k - \mathbf{c})\cdot\mathbf{x}|^2, \quad ||\mathbf{x}||_2 = 1. \tag{4.33}$$

This constrained optimization problem can be solved by applying the method of Lagrange multipliers.

$$\nabla f(\mathbf{x}) = \lambda \nabla g(\mathbf{x}), \tag{4.34}$$

where

$$f(\mathbf{x}) = \sum_{k=1}^{m} |(\mathbf{v}^k - \mathbf{c})\cdot\mathbf{x}|^2, \quad g(\mathbf{x}) = ||\mathbf{x}||_2^2.$$

Note that

$$f(\mathbf{x}) = \sum_{k=1}^{m} \big((v_{k,1} - c_1)x_1 + (v_{k,2} - c_2)x_2\big)^2, \tag{4.35}$$

where $\mathbf{x} = (x_1, x_2)$. Thus it follows from (4.34) that

$$A\mathbf{x} = \lambda\mathbf{x}, \tag{4.36}$$

where $A = (a_{ij}) \in \mathbb{R}^{2\times 2}$ with

$$a_{ij} = \sum_{k=1}^{m} (v_{k,i} - c_i)(v_{k,j} - c_j), \;\; i, j = 1, 2.$$

The direction of the major semi-axis of the ellipsoid can be found from the eigenvector of the matrix A corresponding to the larger eigenvalue.

Definition 4.26. *Let us collect x- and y-components separately from the data:*

$$\begin{aligned} X &= (v_{1,1}, v_{2,1}, \cdots, v_{m,1}), \\ Y &= (v_{1,2}, v_{2,2}, \cdots, v_{m,2}). \end{aligned}$$

Then, the matrix A in (4.36) can be rewritten as

$$A = \begin{bmatrix} \textit{Var}(X) & \textit{Cov}(X,Y) \\ \textit{Cov}(X,Y) & \textit{Var}(Y) \end{bmatrix}, \tag{4.37}$$

called the **covariance matrix** *of the data points (X, Y).*

Semi-axes of the ellipsoid:

In order to find the minor semi-axis, the maximization problem (4.33) can be replaced with the corresponding minimization problem. This would be reduced to the same eigenvalue problem given in (4.36). Thus, both major and minor semi-axes of the ellipsoid can be found from the eigenvalue problem.

$\boxed{\textbf{Theorem}}$ **4.27.** *Let A be the covariance matrix with*

$$A\mathbf{x}_\ell = \lambda_\ell \mathbf{x}_\ell, \;\; ||\mathbf{x}_\ell||_2 = 1.$$

Then the semi-axes of the MVEE in $\mathbb{R}^n$ are found as

$$\eta\sqrt{\lambda_\ell}\,\mathbf{x}_\ell, \quad \ell = 1, \cdots, n, \tag{4.38}$$

where $\eta > 0$ is a scaling constant given by

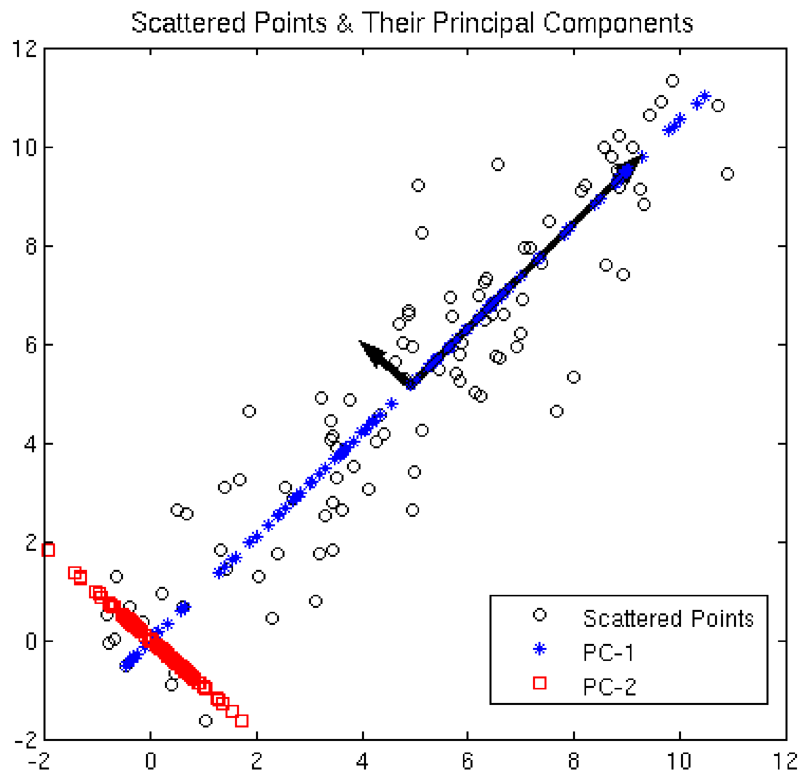
$$\eta^2 = \max_k (\mathbf{v}^k - \mathbf{c})^T A^{-1} (\mathbf{v}^k - \mathbf{c}),$$

provided that $\mathbf{c}$ *is the real center.*

Example 4.28. *A data set of scattered points can be produced and analyzed by*

```
close all; %clear all;
n = 100; h=10/n; P = 1:n;
X = h*P+randn(1,n); Y = h*P+randn(1,n);
%%-----------------------------------------------
plot(X,Y,'ko');
title('Scattered Points & Their Principal Components',...
      'fontsize',13)
%%-----------------------------------------------
A = zeros(n,2);
A(:,1) = X;
A(:,2) = Y;
[U,S,V]= svd(A);
k=1; PC1 = U(:,k)*S(k,k)*V(:,k)';
k=2; PC2 = U(:,k)*S(k,k)*V(:,k)';
hold
plot(PC1(:,1),PC1(:,2),'b*');
plot(PC2(:,1),PC2(:,2),'rs');
legend('Scattered Points','PC-1','PC-2','fontsize',12);
%%-----------------------------------------------
C = cov(X,Y);
[V2,D] = eig(C);
[D,order] = sort(diag(D),'descend');
V2 = V2(:,order); L2 = sqrt(D);
hold
Cntr = [mean(X) mean(Y)];
arrow(Cntr,Cntr+1.5*L2(1)*V2(:,1)','width',3);
arrow(Cntr,Cntr+1.5*L2(2)*V2(:,2)','width',3);
```

Results:



SVD Analysis	**Covariance Analysis**
$S = \begin{bmatrix} 82.9046 & 0 \\ 0 & 8.9832 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \end{bmatrix}$	$L2 = \begin{bmatrix} 4.27206 & 0 \\ 0 & 0.9021 \end{bmatrix}$
$V = \begin{bmatrix} 0.6882 & 0.7255 \\ 0.7255 & -0.6882 \end{bmatrix}$	$V2 = \begin{bmatrix} 0.6826 & -0.7308 \\ 0.7308 & 0.6826 \end{bmatrix}$

- The small mismatch may be due to an approximate setting for the center, for covariance analysis.
- In general, they have different interpretations.
- Look at the ratio of the two singular/eigen- values.
  (Covariance analysis is more geometrical, for this example.)

4.2.4. Image denoising

Now, in this subsection, let's apply the SVD for image denoising. We will see soon that the attempt is a total failure. However, we will learn some important aspects of the SVD through the exercise.

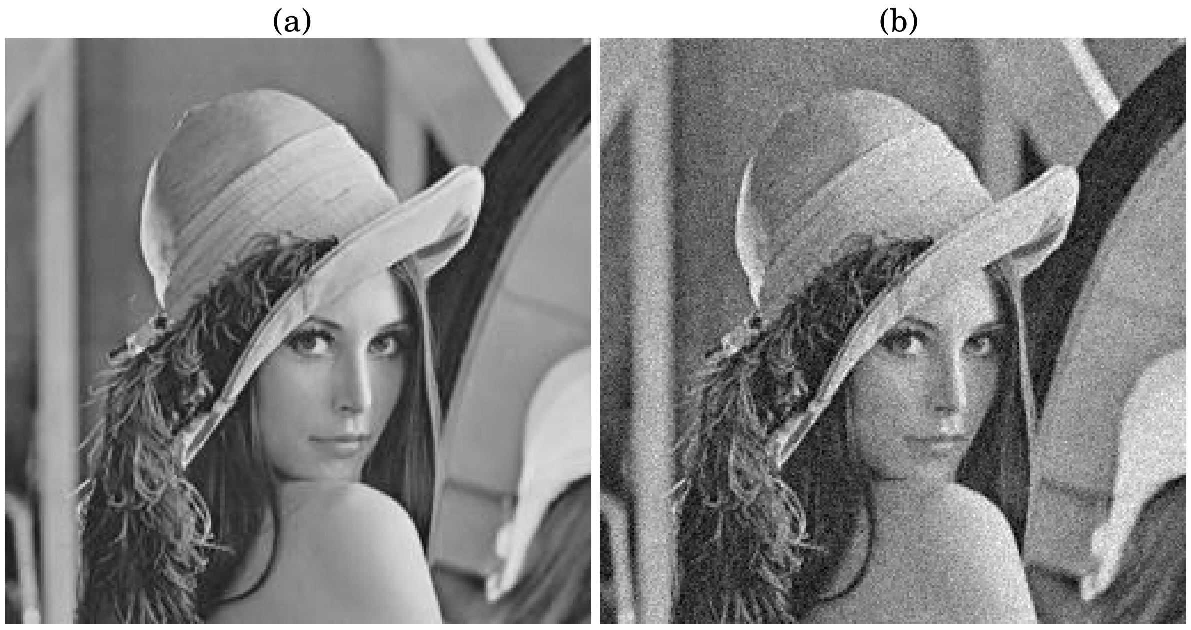


Figure 4.2: Lena: (a) The original image and (b) a noisy image perturbed by Gaussian noise of PSNR=25.7.

In this exercise, each column of the image is viewed as an observation. As in § 4.2.1, we first compute the SVD of the image array A,

$$A = U\Sigma V^T,$$

and for the denoised image, we consider the low-rank approximation

$$A_k = U\Sigma_k V^T = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T.$$

Numerical results

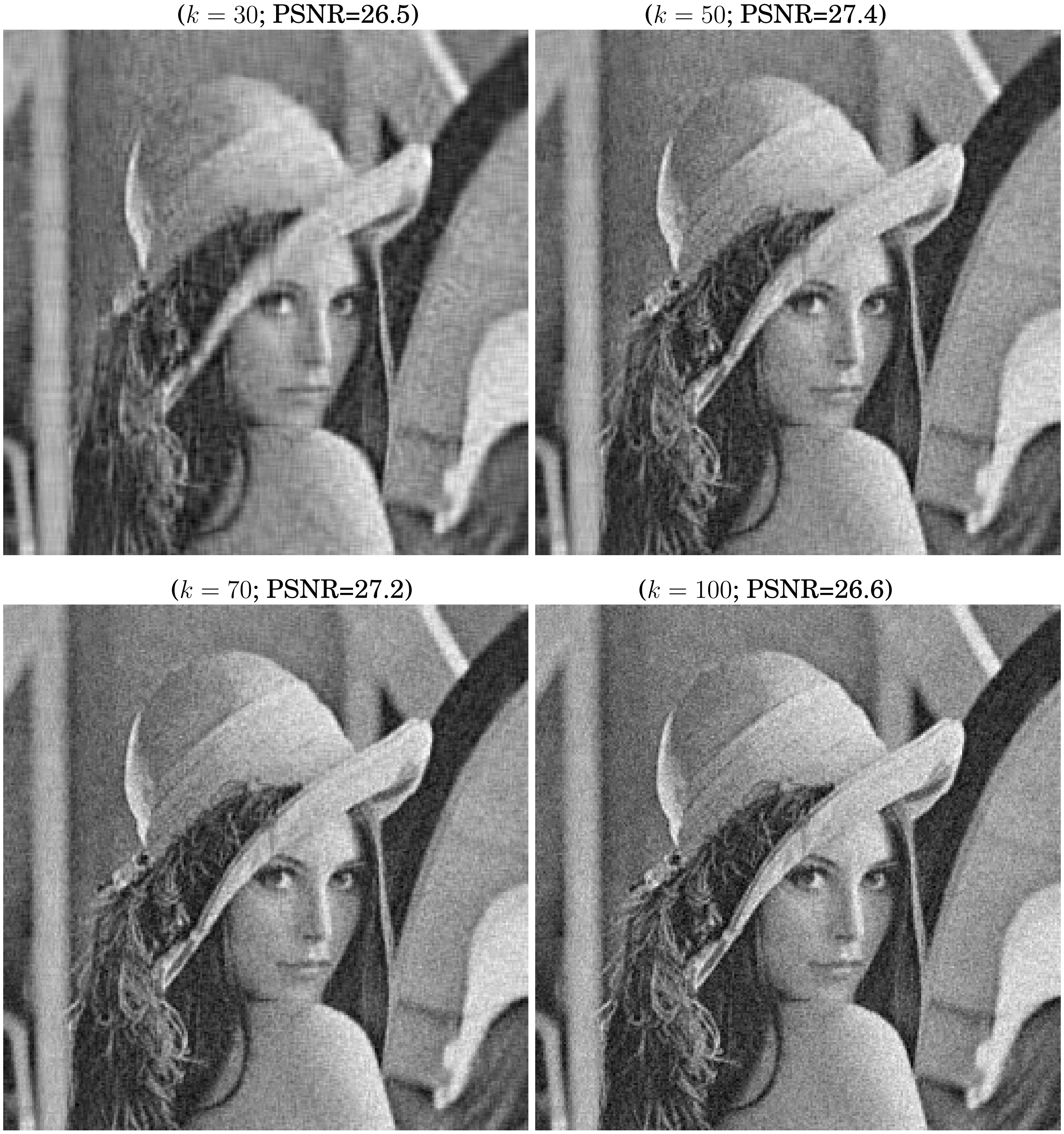
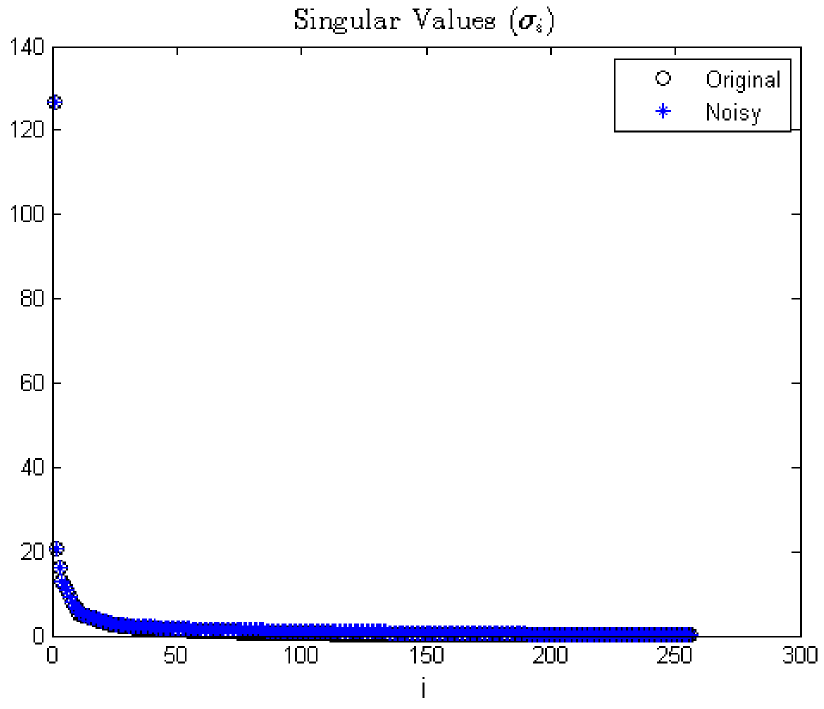


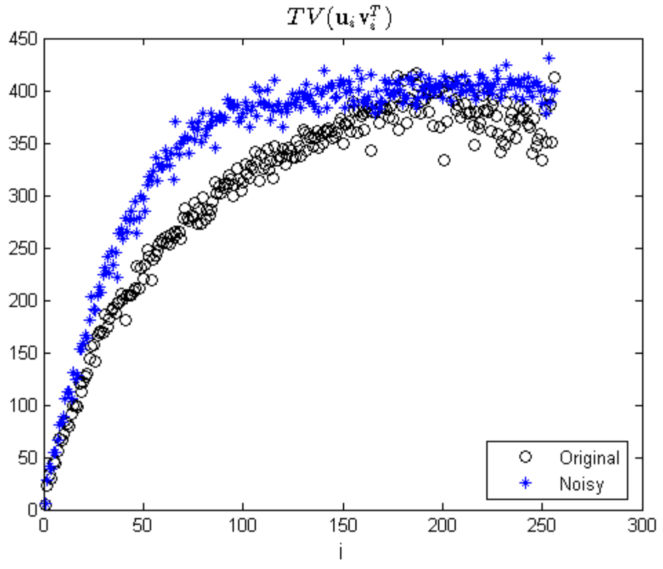
Figure 4.3: Lena: Denoised.

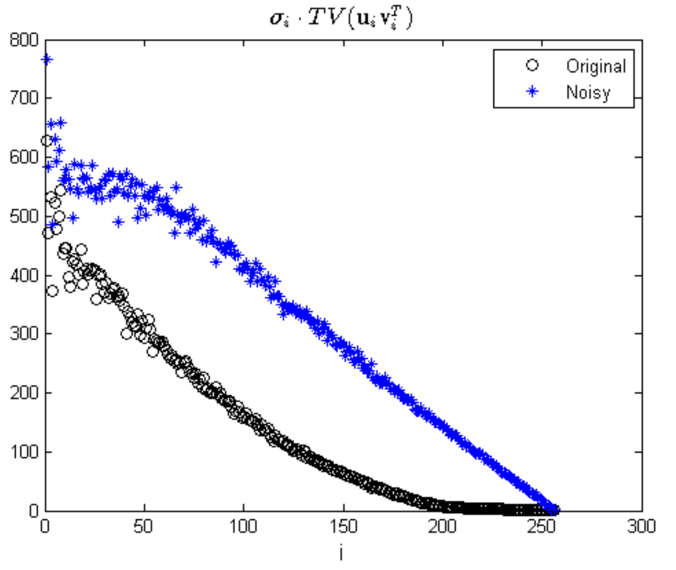
This is a total failure!!

Why a total failure?



$TV(\mathbf{u}_i\mathbf{v}_i^T)$ $\sigma_i \cdot TV(\mathbf{u}_i\mathbf{v}_i^T)$





- Singular vectors are more oscillatory for larger singular values.
- The noise is not localized for certain singular values.
- Columns of an image are **not** those to be considered as correlated, in the first place.

Exercises for Chapter 4

4.1. Let $A \in \mathbb{R}^{m\times n}$. Prove that $||A||_2 = \sigma_1$, the largest singular value of A. **Hint**: Use the following

$$\frac{||A\mathbf{v}_1||_2}{||\mathbf{v}_1||_2} = \frac{\sigma_1||\mathbf{u}_1||_2}{||\mathbf{v}_1||_2} = \sigma_1 \Longrightarrow ||A||_2 \ge \sigma_1$$

and arguments around Equations (4.9) and (4.10) for the opposite directional inequality.

4.2. Recall that the Frobenius matrix norm is defined by

$$||A||_F = \Big(\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2\Big)^{1/2}, \quad A \in \mathbb{R}^{m\times n}.$$

Show that $||A||_F = (\sigma_1^2 + \cdots + \sigma_k^2)^{1/2}$, where σ_j are nonzero singular values of A. **Hint**: First show that if $B = UC$, where U is orthogonal, then $||B||_F = ||C||_F$.

4.3. Prove Lemma 4.16. **Hint**: For (b), let $A\mathbf{v}_i = \lambda_i\mathbf{v}_i$, $i = 1, 2$, and $\lambda_1 \ne \lambda_2$. Then

$$(\lambda_1\mathbf{v}_1)\cdot\mathbf{v}_2 = \underbrace{(A\mathbf{v}_1)\cdot\mathbf{v}_2 = \mathbf{v}_1\cdot(A\mathbf{v}_2)}_{\because\ A \text{ is symmetric}} = \mathbf{v}_1\cdot(\lambda_2\mathbf{v}_2).$$

For (a), you may use a similar argument, but with the dot product being defined for complex values, i.e.,

$$\mathbf{u}\cdot\mathbf{v} = \mathbf{u}^T\,\overline{\mathbf{v}},$$

where $\overline{\text{v}}$ is the complex conjugate of v.

4.4. Use Matlab to generate a random matrix $A \in \mathbb{R}^{8\times 6}$ with rank 4. For example,

```
A = randn(8,4);
A(:,5:6) = A(:,1:2)+A(:,3:4);
[Q,R] = qr(randn(6));
A = A*Q;
```

(a) Print out A on your computer screen. Can you tell by looking if it has (numerical) rank 4?
(b) Use Matlab's "svd" command to obtain the singular values of A. How many are "large?" How many are "tiny?" (You may use the command "format short e" to get a more accurate view of the singular values.)
(c) Use Matlab's "rank" command to confirm that the numerical rank is 4.
(d) Use the "rank" command with a small enough threshold that it returns the value 6. (Type "help rank" for information about how to do this.)

4.5. Let $A \in \mathbb{R}^{2\times 2}$ with singular values $\sigma_1 \ge \sigma_2 > 0$. Show the set $\{A\mathrm{x}\,|\,||x||_2 = 1\}$ (the image of the unit circle) is an ellipse in $\mathbb{R}^2$ whose major and minor semiaxes have lengths σ_1 and σ_2 respectively.

4.6. Work this homework problem using pencil and paper (and exact arithmetic). Let

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

(a) Find the SVD of A. You may use the condensed form given in (4.30).

(b) Calculate the pseudoinverse of A, A^+.

(c) Calculate the minimum-norm solution of the least-squares problem for the overdetermined system $A\mathbf{x} = \mathbf{b}$.

(d) Find a basis for Null(A).

(e) Find all solutions of the least-squares problem.

CHAPTER **5**

Eigenvalue Problems

For eigenvalue problems, we will begin with basic definitions in matrix algebra, including Jordan/Schur canonical forms and diagonalization. Then, we will study perturbation theory in order to understand effects of errors in both data acquisition and computation. After considering Gershgorin's theorem and its applicability, we will work on nonsymmetric eigenvalue problems and finally symmetric eigenvalue problems.

Contents of Chapter 5

5.1. Definitions and Canonical Forms

- For an $n \times n$ matrix A, if a **scalar** λ and a **nonzero** vector x satisfies

$$A\mathbf{x} = \lambda\mathbf{x},$$

  we call λ is a eigenvalue and x a eigenvector of A. $(\boldsymbol{\lambda}, \mathbf{x})$ is called an **eigenpair** of A.

- λ is an eigenvalue of A
  $\Longleftrightarrow$ There $\exists\mathbf{x} \neq \mathbf{0}$ s.t. $A\mathbf{x} = \lambda\mathbf{x}$
  $\Longleftrightarrow$ $(A - \lambda I)\mathbf{x} = \mathbf{0}$ has nontrivial solutions
  $\Longleftrightarrow$ $p(\lambda) = \det(A - \lambda I) = 0.$

  $p(\lambda)$ is called **characteristic polynomial** of A.

- If A has n linearly independent eigenvectors

$$\mathbf{x}_1,\ \mathbf{x}_2, \cdots, \mathbf{x}_n,$$

  then $A \sim \Lambda$ (similar) and has an **EigenValue Decomposition** (EVD):

$$A = X\Lambda X^{-1} \equiv [\mathbf{x}_1\ \mathbf{x}_2\ \cdots\ \mathbf{x}_n] \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix} [\mathbf{x}_1\ \mathbf{x}_2\ \cdots\ \mathbf{x}_n]^{-1}$$

When A is Diagonalizable

- An $n \times n$ matrix A is called **diagonalizable** if $A \sim \Lambda$, a diagonal matrix. Otherwise it is called **defective**.
- The **algebraic multiplicity** of λ_j:
  the multiplicity of the root λ_j in the characteristic equation $\det(A - \lambda I) = 0$.
- The **geometric multiplicity** of λ_j:
  the dimension of the eigenspace for λ_j
  $= dim(Null(A - \lambda_j I))$.

* algebraic multiplicity $\geq$ geometric multiplicity

- Let A is an $n \times n$ matrix. $\Lambda = \text{diag}(\lambda_1, \cdots \lambda_n)$.

  A is diagonalizable
  $\Longleftrightarrow$ A has n lin. indep. e-vectors
  $\Longleftrightarrow$ $\forall \lambda_j$, algebraic multiplicity = geometric multiplicity
  $\Longleftrightarrow$ $S^{-1}AS = \Lambda$ ($\Longleftrightarrow A = S\,\Lambda S^{-1}$)
  $\Longleftrightarrow$ $AS = S\Lambda$; col. of S are the right e-vectors of A
  $\Longleftrightarrow$ $S^{-1}A = \Lambda S^{-1}$; col. of $(S^{-1})^*$ are left e-vectors of A
- Let $S = [\mathbf{x}_1 \ \cdots \ \mathbf{x}_n]$, the matrix of right e-vectors of A. Then $S^{-1} = [\mathbf{y}_1^*/\mathbf{y}_1^*\mathbf{x}_1 \quad \mathbf{y}_2^*/\mathbf{y}_2^*\mathbf{x}_2 \quad \cdots \quad \mathbf{y}_n^*/\mathbf{y}_n^*\mathbf{x}_n]^T$.

Jordan Canonical Forms

- The eigenvector x is also called **right eigenvector**. A **nonzero** vector y satisfying $\mathbf{y}^*A = \lambda\mathbf{y}^*$ is called **left eigenvector** of A. (Recall that $\mathbf{y}^* = y^H = (\bar{\mathbf{y}})^T$ is the conjugate transpose of y.)
- **Jordan Canonical Forms** Given a square matrix A, there exists a nonsingular S such that

$$S^{-1}AS = J, \tag{5.1}$$

where J is in **Jordan canonical form**. This means that J is block diagonal, with

$$J = \text{diag}(J_{n_1}(\lambda_1), J_{n_2}(\lambda_2), \cdots, J_{n_k}(\lambda_k))$$

and

$$J_{n_j}(\lambda_j) = \begin{bmatrix} \lambda_j & 1 & & & \\ & \lambda_j & 1 & & \\ & & \ddots & \ddots & \\ & & & \ddots & 1 \\ & & & & \lambda_j \end{bmatrix}_{n_j\times n_j} \quad \begin{array}{l}\textbf{(Jordan block} \text{ for } \lambda_j) \\ n_j\text{=algebraic multiplicity}\end{array}$$

J is **unique**, up to permutations of its diagonal blocks.

Schur Canonical Forms

- For an $n \times n$ matrix A, there exists a **unitary** matrix Q and an **upper triangular** matrix T such that

$$Q^* AQ = T. \tag{5.2}$$

The eigenvalues of A are the diagonal entries of T.

- **Real Schur Canonical Forms**. If A is real, there exists a real **orthogonal** matrix V such that

$$V^T AV = T \text{ is } \textbf{quasi-upper triangular}.$$

This means that T is block upper triangular with 1×1 and 2×2 blocks on the diagonal.

Its eigenvalues are the eigenvalues of its diagonal blocks. The 1×1 blocks correspond to real eigenvalues, and the 2×2 blocks to complex conjugate pairs of eigenvalues.

Theorem 5.1. (Fundamental Theorem of Algebra). *Every nonconstant polynomial has at least one root (possibly, in the complex field).*

Polynomial Factorization: If P is a nonconstant polynomial of real coefficients, then it can be factorized as a multiple of linear and quadratic factors of which coefficients are all real.

Computing E-vectors from Schur Form

- Let $Q^*AQ = T$ be the Schur form of A. Then if λ is an eigenvalue of T, $T\mathbf{x} = \lambda\mathbf{x}$.
$$AQ\mathbf{x} = QT\mathbf{x} = Q(\lambda\mathbf{x}) = \lambda Q\mathbf{x}$$
So $Q\mathbf{x}$ is an eigenvector of A.
- **To find eigenvectors of A, it suffices to find eigenvectors of T.**

5.2. Perturbation Theory: Bauer-Fike Theorem

$\boxed{\textbf{Theorem}}$ **5.2.** *Let λ be a* **simple** *eigenvalue of A with right eigenvector $\mathbf{x}$ and left eigenvector $\mathbf{y}$, normalized so that $||\mathbf{x}||_2 = ||\mathbf{y}||_2 = 1$. Let $\lambda + \delta\lambda$ be the corresponding eigenvalue of $A + \delta A$. Then*

$$\begin{aligned} \delta\lambda &= \frac{\mathbf{y}^*\delta A\mathbf{x}}{\mathbf{y}^*\mathbf{x}} + \mathcal{O}(||\delta A||^2), \quad \textit{or} \\ |\delta\lambda| &\le \frac{||\delta A||}{|\mathbf{y}^*\mathbf{x}|} + \mathcal{O}(||\delta A||^2) = \sec\Theta(\mathbf{y},\mathbf{x})||\delta A|| + \mathcal{O}(||\delta A||^2) \end{aligned} \tag{5.3}$$

where $\Theta(\mathbf{y},\mathbf{x})$ is the acute angle between $\mathbf{y}$ and $\mathbf{x}$.

Note that $\sec\Theta(\mathbf{y},\mathbf{x}) = 1/|\mathbf{y}^*\mathbf{x}| \equiv c(\lambda)$ is the **condition number** *of the eigenvalue $\boldsymbol{\lambda}$. [$s(\lambda) = |\mathbf{y}^*\mathbf{x}|$ in other texts.]*

Note: The perturbation on the eigenvalue (due to a matrix perturbation) is proportional to the condition number of the eigenvalue.

Corollary **5.3.** *Let A be be* **symmetric** *(or, more generally,* **normal***: $AA^* = A^*A$). Then*

$$|\delta\lambda| \le ||\delta A|| + \mathcal{O}(||\delta A||^2). \tag{5.4}$$

Note: Theorem 5.2 is useful only for sufficiently small $||\delta A||$. We can remove the $\mathcal{O}(||\delta A||^2)$ term and get a simple theorem true for any size perturbation $||\delta A||$, at the cost of increasing the condition number of the eigenvalue by a factor of n.

$\boxed{\textbf{Theorem}}$ **5.4. (Bauer-Fike Theorem).** *Let A have all simple eigenvalues (i.e., be diagonalizable). Call them λ_i, with right and left e-vectors $\mathbf{x}_i$ and $\mathbf{y}_i$, normalized so $||\mathbf{x}_i||_2 = ||\mathbf{y}_i||_2 = 1$. Then the e-values of $A + \delta A$ lie in disks B_i, where B_i has center λ_i and radius $n\frac{||\delta A||}{|\mathbf{y}_i^* \mathbf{x}_i|}$.*

$\boxed{\textbf{Theorem}}$ **5.5. (Bauer-Fike Theorem).** *If μ is an eigenvalue of $A+E \in \mathbb{C}^{n\times n}$ and*

$$X^{-1}AX = D = \textit{diag}(\lambda_1, \cdots, \lambda_n),$$

then

$$\min_{\lambda\in\sigma(A)} |\lambda - \mu| \le ||X||_p \cdot ||X^{-1}||_p \cdot ||E||_p = \kappa_p(X)\ ||E||_p \tag{5.5}$$

where $||\cdot||_p$ denotes any of the p-norms. [10, Theorem 7.2.2].

$\boxed{\textbf{Theorem}}$ **5.6.** *Let λ be a simple e-value of A, with unit right and left e-vectors* x *and* y *and* **condition number** $c = 1/|\mathbf{y}^*\mathbf{x}|$. *Then $\exists\ \delta A$ such that $A + \delta A$ has a multiple e-value at λ, and*

$$\frac{||\delta A||_2}{||A||_2} \le \frac{1}{\sqrt{c^2-1}}.$$

If $c \gg 1$, i.e., the e-value is ill-conditioned, then the upper bound on the distance is $\frac{1}{\sqrt{c^2-1}} \approx \frac{1}{c}$, the reciprocal of the condition number.

Note: Multiple eigenvalues have infinite condition numbers; see Homework 1. Being "close to singular" implies ill-conditioning.

Example 5.7. *Let* $A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 4.001 \end{bmatrix}$, $E = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ .001 & 0 & 0 \end{bmatrix}$. *Then*

$$\begin{aligned} \sigma(A) &= \{1,\ 4,\ 4.001\}, \\ \sigma(A+E) &\approx \{1.0001,\ 4.0582,\ 3.9427\}. \end{aligned}$$

The reciprocals of the condition numbers are

$$\begin{aligned} s(1) &\approx .8, \\ s(4) &\approx .2e-3, \\ s(4.001) &\approx .2e-3, \end{aligned}$$

and

$$\frac{||E||_2}{||A||_2} = \frac{0.001}{8.0762} \approx 0.00012 < .2e-3$$

See Theorem 5.6.

Note: If any eigenvalue has a large condition number, then the matrix of eigenvectors has to have an approximately equally large condition number.

$\boxed{\textbf{Theorem}}$ **5.8.** *Let A be diagonalizable with e-values λ_i and right and left e-vectors $\mathbf{x}_i$, $\mathbf{y}_i$, respectively, normalized so $||\mathbf{x}_i|| = ||\mathbf{y}_i|| = 1$. Suppose that S satisfies*

$$S^{-1}AS = \Lambda = \textbf{\textit{diag}}(\lambda_1, \cdots, \lambda_n).$$

Then

$$\kappa_2(S) \equiv ||S||_2 \cdot ||S^{-1}||_2 \geq \max_i \frac{1}{|\mathbf{y}_i^*\mathbf{x}_i|}. \tag{5.6}$$

If we choose $S = [\mathbf{x}_1, \cdots, \mathbf{x}_n]$, then

$$\kappa_2(S) \leq n \cdot \max_i \frac{1}{|\mathbf{y}_i^*\mathbf{x}_i|}. \tag{5.7}$$

That is, the condition number of S, $\kappa_2(S)$, is within a factor of n of its smallest value.

5.3. Gerschgorin's Theorem

$\boxed{\textbf{Theorem}}$ **5.9. (Gerschgorin's Theorem [9]).** *Let B be an arbitrary matrix. Then the eigenvalues λ of B are located in the union of the n disks defined by*

$$|\lambda - b_{ii}| \le r_i \equiv \sum_{j\neq i} |b_{ij}|, \quad \textit{for } i = 1, \cdots, n. \tag{5.8}$$

Proof. Let $B\mathbf{x} = \lambda\mathbf{x}$, $\mathbf{x} \neq 0$. By scaling x if necessary, we may assume that

$$||\mathbf{x}||_\infty = x_k = 1, \text{ for some } k.$$

Then

$$\sum_{j=1}^{n} b_{kj} x_j = \lambda x_k = \lambda$$

and therefore

$$|\lambda - b_{kk}| = \Big|\sum_{j\neq k} b_{kj} x_j\Big| \le \sum_{j\neq k} |b_{kj} x_j| \le \sum_{j\neq k} |b_{kj}|. \tag{5.9}$$

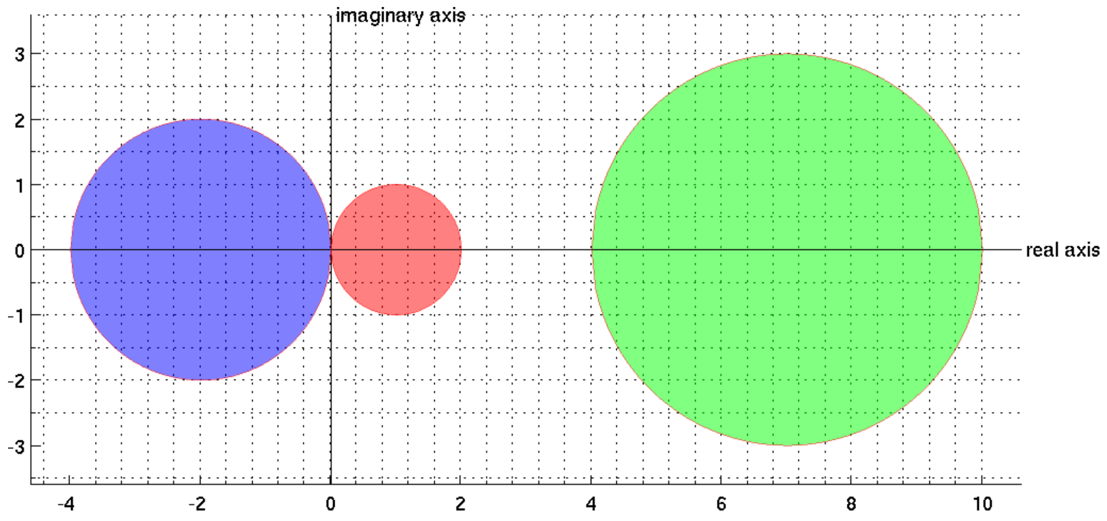
For other eigenvalues, one can derive a similar inequality as in (5.9). □

Gerschgorin Circle

Example 5.10. *Let* $A = \begin{bmatrix} 1 & 0 & -1 \\ -2 & 7 & 1 \\ -1 & 1 & -2 \end{bmatrix}$. *Find the Gerschgorin circles.*

Solution. From the Gerschgorin's theorem,

$$\begin{aligned} C_1 &= \{z : |z-1| \le 1\}, \\ C_2 &= \{z : |z-7| \le 3\}, \\ C_3 &= \{z : |z+2| \le 2\}. \end{aligned}$$



The eigenvalues of A are 7.1475, 1.1947, and -2.3422.

Example 5.11. *The following matrix comes from modeling the heat flow in a long, thin rod, or mechanical vibrations.*

$$A = \begin{bmatrix} a_1 & -c_1 & 0 & 0 & \cdots & 0 \\ -b_2 & a_2 & -c_2 & 0 & \cdots & 0 \\ 0 & -b_3 & a_3 & -c_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & -b_{n-1} & a_{n-1} & -c_{n-1} \\ 0 & 0 & \cdots & 0 & -b_n & a_n \end{bmatrix}$$

where $a_i > 0$, $b_i > 0$ and $c_i > 0$. The matrix A is strictly diagonally dominant, that is,

$$|b_i| + |c_i| < |a_i|, \text{ for } i = 1, 2, \cdots, n.$$

We may assume that $|b_i| + |c_i| = 1 < a_i$, for $i = 1, 2, \cdots, n$. (let $b_1 = c_n = 0$). Show that all the e-values of A are positive and hence A is nonsingular.

Proof. By Gerschgorin's theorem, every eigenvalue λ of $A = (a_{ij})_{n\times n}$ satisfies:

$$\begin{gathered} |\lambda - a_{ii}| \le \sum_{j\neq i} |a_{ij}|, \quad i = 1, 2, \cdots, n \\ |\lambda - a_i| \le |b_i| + |c_i| = 1, \quad i = 1, 2, \cdots, n \\ 0 < -1 + a_i < \lambda < 1 + a_i \end{gathered}$$

So all the e-values of A are positive and hence A is nonsingular. □

5.4. Taussky's Theorem and Irreducibility

Theorem 5.12. (Taussky [22, 1948]) *Let $B = [b_{ij}]$ be an* **irreducible** *$n \times n$ complex matrix. Assume that λ, an eigenvalue of B, is a boundary point of the union of the disks $|z - b_{ii}| \le r_i$. Then, all the n circles $|z - b_{ii}| = r_i$ must pass through the point λ, i.e.,*

$$|\lambda - b_{ii}| = r_i \text{ for all } 1 \le i \le n.$$

Definition 5.13. *A* **permutation matrix** *is a square matrix in which each row and each column has one entry of unity, all others zero.*

Definition 5.14. *For $n \ge 2$, an $n \times n$ complex-valued matrix A is* **reducible** *if there is a permutation matrix P such that*

$$PAP^T = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix},$$

where A_{11} and A_{22} are respectively $r \times r$ and $(n-r) \times (n-r)$ submatrices, $0 < r < n$. If no such permutation matrix exists, then A is **irreducible**.

Geometrical interpretation of irreducibility

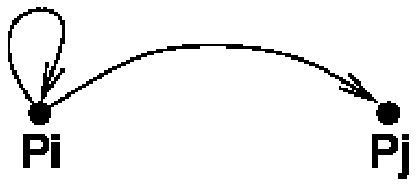


Figure 5.1: The directed paths for nonzero a_{ii} and a_{ij}.

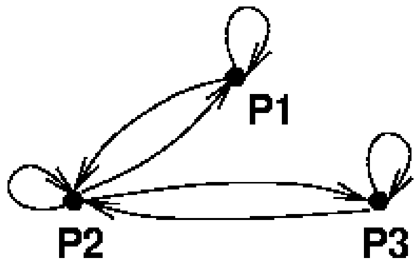


Figure 5.2: The directed graph $G(A)$ for A in (5.10).

- Given $A = (a_{ij}) \in \mathbb{C}^{n\times n}$, consider n distinct points

$$P_1, P_2, \cdots, P_n$$

  in the plane, which we will call **nodes** or **nodal points**.

- For any nonzero entry a_{ij} of A, we connect P_i to P_j by a path $\overrightarrow{P_iP_j}$, directed from the node P_i to the node P_j; a nonzero a_{ii} is joined to itself by a directed loop, as shown in Figure 5.1.
- In this way, every $n \times n$ matrix A can be associated a *directed graph* $G(A)$. For example, the matrix

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \tag{5.10}$$

  has a directed graph shown in Figure 5.2.

Definition 5.15. *A directed graph is* **strongly connected** *if, for any ordered pair of nodes* (P_i, P_j)*, there is a directed path of a finite length*

$$\overrightarrow{P_i P_{k_1}},\ \overrightarrow{P_{k_1} P_{k_2}},\ \cdots,\ \overrightarrow{P_{k_{r-1}} P_{k_r=j}},$$

connecting from P_i *to* P_j*.*

The theorems to be presented in this subsection can be found in [24] along with their proofs.

Theorem 5.16. *An* $n \times n$ *complex-valued matrix* A *is irreducible if and only if its directed graph* $G(A)$ *is strongly connected.*

It is obvious that the matrices obtained from FD/FE methods of the Poisson equation are strongly connected. Therefore the matrices are irreducible.

Example 5.17. *Find the eigenvalue loci for the matrix*

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}.$$

Solution. The matrix A is irreducible, first of all, and

$$r_1 = 1, \quad r_2 = 2, \quad \text{and} \quad r_3 = 1.$$

Since $a_{ii} = 2$, for $i = 1, 2, 3$, the union of the three disks reads

$$\{z : |z - 2| \le 2\} \equiv \Omega.$$

Note $r_1 = r_3 = 1$ and their corresponding disks do not touch the boundary of Ω. So, using the Taussky theorem, we have

$$|\lambda - 2| < 2 \tag{5.11}$$

for all eigenvalues λ of A. Furthermore, since A is symmetric, its eigenvalues must be real. Thus we conclude

$$0 < \lambda < 4. \tag{5.12}$$

5.5. Nonsymmetric Eigenvalue Problems

Residual and Rayleigh Quotient

- Let (λ, x) be an approximation to an eigenpair of A. Then, the **residual** is defined as
$$\mathbf{r} = A\mathbf{x} - \lambda\mathbf{x}.$$
- If x is an approximation to an e-vector of A, then the **Rayleigh quotient**
$$\widetilde{\lambda} = \frac{\mathbf{x}^* A\mathbf{x}}{\mathbf{x}^*\mathbf{x}}$$
is an approximation to the corresponding e-value.
- **Rayleigh quotient minimizes** $||\mathrm{r}|| = ||A\mathrm{x} - \lambda\mathrm{x}||$**.**

5.5.1. Power method

Given $A \in \mathbb{C}^{n\times n}$, let

$$X^{-1}AX = \text{diag}(\lambda_1, \cdots, \lambda_n),$$

where

$$X = [\mathbf{x}_1, \cdots, \mathbf{x}_n], \text{ and } |\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|.$$

The **power method** produces a sequence of vectors $\{\mathbf{q}^{(k)}\}$ as follows,

```
choose $\mathbf{q}^{(0)} \in \mathbb{C}^n$ such that $||\mathbf{q}^{(0)}||_2 = 1$
for $k = 1, 2, \cdots$
    $\mathbf{z} = A\mathbf{q}^{(k-1)}$                        apply $A$
    $\mathbf{q}^{(k)} = \mathbf{z}/||\mathbf{z}||_2$           normalize            $\to \mathbf{q}$
    $\lambda^{(k)} = [\mathbf{q}^{(k)}]^* A\mathbf{q}^{(k)}$   Rayleigh quotient    $\to \lambda_1$
end
```
(5.13)

Note: $\mathbf{q}^{(k)} \to \mathbf{q} = \pm\mathbf{x}_1/||\mathbf{x}_1||$ as $k \to \infty$, where $\mathbf{x}_1$ is the eigenvector corresponding to λ_1.

Convergence of power method

Note $A^k\mathbf{x}_j = \lambda_j^k \mathbf{x}_j$. Let

$$\mathbf{q}^{(0)} = \alpha_1\mathbf{x}_1 + \alpha_2\mathbf{x}_n + \cdots + \alpha_n\mathbf{x}_n, \quad \alpha_1 \neq 0.$$

Then

$$A^k\mathbf{q}^{(0)} = \sum_{j=1}^{n} \alpha_j A^k \mathbf{x}_j = \alpha_1\lambda_1^k \Big(\mathbf{x}_1 + \sum_{j=2}^{n} \frac{\alpha_j}{\alpha_1}\left(\frac{\lambda_j}{\lambda_1}\right)^k \mathbf{x}_j\Big)$$

Since $\mathbf{q}^{(k)} \in \text{Span}\left\{A^k\mathbf{q}^{(0)}\right\}$,

$$dist\Big(\text{Span}\{\mathbf{q}^{(k)}\}, \text{Span}\{\mathbf{x}_1\}\Big) = \mathcal{O}\Big(\left|\frac{\lambda_2}{\lambda_1}\right|^k\Big).$$

Thus, we obtain

$$\left|\lambda_1 - \lambda^{(k)}\right| = \mathcal{O}\Big(\left|\frac{\lambda_2}{\lambda_1}\right|^k\Big). \tag{5.14}$$

Error estimates for the power method

- The estimate of the error $|\lambda^{(k)} - \lambda_1|$ can be obtained by applying Bauer-Fike theorem, more specifically, Theorem 5.2 on p. 175.

  Let $\mathbf{r}^{(k)} = A\mathbf{q}^{(k)} - \lambda^{(k)}\mathbf{q}^{(k)}$. Observe that

$$(A + E^{(k)})\mathbf{q}^{(k)} = \lambda^{(k)}\mathbf{q}^{(k)}, \text{ for } E^{(k)} = -\mathbf{r}^{(k)}\left(\mathbf{q}^{(k)}\right)^{*}. \tag{5.15}$$

  Thus $\lambda^{(k)}$ is an e-value of $A + E^{(k)}$, and

$$\left|\lambda^{(k)} - \lambda_1\right| \approx \frac{||E^{(k)}||_2}{s(\lambda_1)} = \frac{||\mathbf{r}^{(k)}||_2}{s(\lambda_1)}$$

  where $s(\lambda_1)$ is the reciprocal of the condition number of λ_1.

- If we use the power method to generate approximate right/left dominant e-vectors, then it is possible to obtain an estimate of $s(\lambda_1)$. In particular, if $\mathbf{w}^{(k)}$ is a unit 2-norm in the direction of $(A^*)^k\mathbf{w}^{(0)}$, then

$$s(\lambda_1) \approx |(\mathbf{w}^{(k)})^*\mathbf{q}^{(k)}|. \tag{5.16}$$

Example 5.18. *Let* $A = \begin{bmatrix} -31 & -35 & 16 \\ -10 & -8 & 4 \\ -100 & -104 & 49 \end{bmatrix}$*, then* $\sigma(A) = \{9, 3, -2\}$*. Applying the power method with* $\mathbf{q}^{(0)} = [1, 0, 0]^T$*.*

Solution. We have

k	$\lambda^{(k)}$
1	7.6900
2	9.6377
3	8.9687
4	9.0410
5	9.0023
6	9.0033
7	9.0005
8	9.0003
9	9.0001
10	9.0000

Power method converges at $k = 10$ (correct to 4 decimal places).

Computed result:

eigenvalue: $\lambda^{(10)} = 9.000$,

eigenvector: $\mathbf{x} = \mathbf{q}^{(10)} = \begin{bmatrix} -0.3714 \\ -0.0000 \\ -0.9285 \end{bmatrix}$

residual: $||\mathbf{r}^{(10)}|| = 9.1365e - 06.$

Remark 5.19. *Power iteration is of* **limited** *use due to several reasons.*

- *It converges to the eigenpair* **only** *for the eigenvalue with largest absolute magnitude.*
- *The usefulness of the power method depends upon the* **ratio** $|\lambda_2|/|\lambda_1|$. *The smaller the ratio, the faster the convergence. But if there are two largest eigenvalue are* **close** *in magnitude, the convergence will be very* **slow**.
- *For example, if A is real and the largest eigenvalue is complex, then there will be two* **complex conjugate** *e-values with the same largest absolute magnitude $|\lambda_1| = |\lambda_2|$. In this case, the power method will* **not** *converge. And it won't work in the extreme case of an orthogonal matrix with all the eigenvalues the same absolute value.*
- *Overall, the method is useful when A is* **large and sparse**, *and when there is a* **big gap** *between $|\lambda_1|$ and $|\lambda_2|$.*

5.5.2. Inverse power method

To overcome the drawbacks of power method, we may apply it to $(A - \sigma I)^{-1}$ instead of A, where σ is called a **shift**. Then it will converge to the e-value **closest** to σ, rather than just λ_1.

It is called **inverse iteration** or **inverse power iteration**.

Algorithm 5.20. (Inverse Power Iteration).

```
input $\mathbf{x}_0$;                                   initial guess
$i = 0$;
do repeat
    solve $(A - \sigma I)\mathbf{y}_{i+1} = \mathbf{x}_i$
    $\mathbf{x}_{i+1} = \mathbf{y}_{i+1}/||\mathbf{y}_{i+1}||_2$;        approx. e-vector
    $\eta_{i+1} = \mathbf{x}_{i+1}^T A\mathbf{x}_{i+1}$;                 approx. e-value
    $i = i + 1$;
until convergence
```
(5.17)

Convergence of Inverse Iterations

- Let $A = S\Lambda S^{-1}$. Then
$$A - \sigma I = S(\Lambda - \sigma I)S^{-1}$$
and therefore
$$(A - \sigma I)^{-1} = S(\Lambda - \sigma I)^{-1}S^{-1}.$$
Thus $(A - \sigma I)^{-1}$ has the same e-vectors $\mathbf{s}_j$ as A with e-values $(\lambda_j - \sigma)^{-1}$.
- If $|\lambda_k - \sigma| < |\lambda_i - \sigma|$ for all $i \neq k$, then
$$\max_{i\neq k} \frac{\lambda_k - \sigma}{\lambda_i - \sigma} < 1.$$
Let $\mathbf{x}_0 = \sum_i \alpha_i \mathbf{s}_i$. Then,

$$\begin{aligned}
(A-\sigma I)^{-j}\mathbf{x}_0 &= (S(\Lambda-\sigma I)^{-j}S^{-1})\,S\begin{bmatrix}\alpha_1\\ \alpha_2\\ \vdots\\ \alpha_n\end{bmatrix}\\
&= S\begin{bmatrix}\alpha_1(\lambda_1-\sigma)^{-j}\\ \alpha_2(\lambda_2-\sigma)^{-j}\\ \vdots\\ \alpha_n(\lambda_n-\sigma)^{-j}\end{bmatrix}\\
&= \alpha_k(\lambda_k-\sigma)^{-j}S\begin{bmatrix}\frac{\alpha_1}{\alpha_k}\left(\frac{\lambda_k-\sigma}{\lambda_1-\sigma}\right)^j\\ \vdots\\ 1\\ \vdots\\ \frac{\alpha_n}{\alpha_k}\left(\frac{\lambda_k-\sigma}{\lambda_n-\sigma}\right)^j\end{bmatrix}\\
&\to \beta S\mathbf{e}_k = \beta\mathbf{s}_k\\
\eta_j &= \mathbf{x}_j^T A\mathbf{x}_j \to \lambda_k
\end{aligned} \tag{5.18}$$

Convergence of Inverse Iterations (2)

- Inverse iteration can be used in conjunction with the QR algorithm as follows [10, p.363].
  - * Compute the **Hessenberg decomposition**

    $$U_0^T A U_0 = H,$$

    where $U_0^T U_0 = I$ and $H = [h_{ij}]$ is a Hessenberg matrix, i.e., $h_{ij} = 0$ for $i > j + 1$.
  - * Apply the **double implicit shift Francis QR iteration** to H without accumulating transformations. (See [10, § 7.5.5].)
  - * For each computed e-value λ whose corresponding e-vector is sought, apply the inverse power iteration, Algorithm 5.20, with $A = H$ and $\sigma = \lambda$ to produce a vector z such that

    $$H\mathbf{z} \approx \sigma\mathbf{z}.$$
  - * Set $\mathbf{x} = U_0\mathbf{z}$.
- Inverse iteration with H is very economical because
  - * we do not have to accumulate transformations during double Francis iteration,
  - * we can factor matrices of the form $H - \lambda I$ in $\mathcal{O}(n^2)$ flops, and
  - * only one iteration is typically required to produce an adequate approximate e-vector.

5.5.3. Orthogonal/Subspace iteration

Orthogonal iteration (or **subspace iteration**, or **simultaneous iteration**) – converges to a $(p > 1)$-dimensional **invariant subspace**, rather than one eigenvector at a time [10, p.363].

Definition 5.21. *An* **invariant subspace** *of A is a subspace X of $\mathbb{R}^n$, with the property that $\mathbf{x} \in X$ implies that $A\mathbf{x} \in X$. We also write this as*

$$AX \subseteq X. \tag{5.19}$$

Algorithm 5.22. (Orthogonal iteration).

```
$Z_0$ — an $n \times p$ orthogonal matrix
$i = 0$
do repeat
    $Y_{i+1} = AZ_i$
    factor $Y_{i+1} = Z_{i+1}R_{i+1}$ using $QR$
    // $Z_{i+1}$ spans an approx. invariant subsp.
    $i = i + 1$
until convergence
```
(5.20)

- If $p = 1$, the orthogonal iteration becomes the power iteration.

Convergence of the orthogonal iteration

Theorem **5.23.** *Consider running* **orthogonal iteration** *on matrix A with $p = n$ and $Z_0 = I$. If all the e-values of A have* **distinct** *absolute values and if all the principal submatrices $S(1:j,\ 1:j)$ have* **full rank**, *then $A_i \equiv Z_i^T A Z_i$ converges to the* **Schur form** *of A, i.e., an upper triangular matrix with the e-values on the diagonal. The e-values will appear in decreasing order of absolute value.*

Proof. **(Sketch)**. Z_i is a square orthogonal matrix, so A and $A_i = Z_i^T A Z_i$ are similar. Write $Z_i = [Z_{1i}, Z_{2i}]$, where Z_{1i} has p columns, so

$$A_i = Z_i^T A Z_i = \begin{bmatrix} Z_{1i}^T A Z_{1i} & Z_{1i}^T A Z_{2i} \\ Z_{2i}^T A Z_{1i} & Z_{2i}^T A Z_{2i} \end{bmatrix}$$

Since Span$\{Z_{1i}\}$ converges to an invariant subspace of A, Span$\{AZ_{1i}\}$ converges to the same subspace. Thus, $Z_{2i}^T A Z_{1i}$ converges to $Z_{2i}^T Z_{1i} = 0$. This is true for all $p < n$, every subdiagonal entry of A_i converges to zero, so A_i converges to upper triangular form, i.e., Schur form. $\square$

Remarks:

- In **Theorem 5.23**, $\boldsymbol{S(1:j,\ 1:j)}$ must be nonsingular. Suppose that A is diagonal with the eigenvalues not in decreasing order. Then orthogonal iteration yields $Z_i = \text{diag}(\pm 1)$ and $A_i = A$ for all i, so the eigenvalues do not move into decreasing order.
- **Theorem 5.23** also requires that the e-values of A must be distinct in absolute values. If A is orthogonal and all its e-values have absolute value 1, then the algorithm leaves A_i essentially unchanged. (The rows and columns may be multiplied by -1.)

5.5.4. QR iteration

Algorithm 5.24. (QR iteration — without shifts).

```
$A_0$                        — original matrix
$i = 0$
do repeat
   factor $A_i = Q_i R_i$    — $QR$ decomp. of $A_i$
   $A_{i+1} = R_i Q_i$       — Combine $Q$ & $R$ reversely        (5.21)
   $i = i + 1$
until convergence
```

- Note that
$$A_{i+1} = R_i Q_i = Q_i^T (Q_i R_i) Q_i = Q_i^T A_i Q_i.$$
Thus, A_{i+1} and A_i are orthogonally similar.
- **Lemma 5.25.** *$A_i = Z_i^T A Z_i$, where Z_i is the matrix computed from orthogonal iteration (Algorithm 5.22). Thus A_i converges to* **Schur form** *if all the e-values have* **different absolute** *values.*

Speedup of QR iteration

Algorithm 5.26. **(QR iteration with a shift).**

```
$A_0$   (original $n \times n$ matrix)
$i = 0$
do repeat
    choose a shift $\sigma_i$ near an e-value of $A$
    factor $A_i - \sigma_i I = Q_i R_i$
    $A_{i+1} = R_i Q_i + \sigma_i I$
    $i = i + 1$
until convergence
```
(5.22)

- Note that

$$\begin{aligned} A_{i+1} &= R_i Q_i + \sigma_i I = Q_i^T (Q_i R_i) Q_i + \sigma_i Q_i^T Q_i \\ &= Q_i^T (Q_i R_i + \sigma_i I) Q_i = Q_i^T A_i Q_i. \end{aligned}$$

  Thus, $A_{i+1} \sim A_i$ orthogonally.

- If R_i is nonsingular,

$$\begin{aligned} A_{i+1} &= R_i Q_i + \sigma_i I = R_i Q_i R_i R_i^{-1} + \sigma_i R_i R_i^{-1} \\ &= R_i (Q_i R_i + \sigma_i I) R_i^{-1} = R_i A_i R_i^{-1} \end{aligned}$$

- The QR iteration is **still** too expensive.
- We may choose $\sigma_i = A_i(n,n)$, for a faster convergence.

QR and Orthogonal Iterations, in Matlab

QR_Orthogonal.m
```
lambda = [1 2 4 8 32];
n = 5;
X = randn(n);
A = X*diag(lambda)*inv(X)
itmax = 100; tol = 1.e-12;

%---------------------------------
[T,iter] = qr_iteration(A,itmax,tol);
fprintf('QR-Iteration:\n')
T,iter

%---------------------------------
[T,iter] = qr_iteration_shift(A,itmax,tol);
fprintf('QR-Iteration-with-Shift:\n')
T,iter

%---------------------------------
[Q,iter] = orthogonal_iteration(A,itmax,tol);
fprintf('Orthogonal-Iteration:\n')
Schur = Q'*A*Q, iter
```

OUT
```
A =
  -31.1300  -98.4723   25.9829  -26.0856  -56.8437
   23.3689   68.1640  -13.6507   15.2984   34.0583
   -0.6924   -2.7223    2.7794    0.6017   -0.8296
   13.7263   33.0531   -4.7360   11.0045   16.8833
   -9.6383  -20.4111   -0.9314   -2.4357   -3.8179

QR-Iteration:
T =
   32.0000   -7.6229  -12.7664   -9.5433 -150.4702
    0.0000    8.0000    0.7922    8.7506   11.1408
    0.0000    0.0000    4.0000    1.2222   -3.4860
   -0.0000   -0.0000   -0.0000    2.0000   -2.2486
    0.0000   -0.0000    0.0000    0.0000    1.0000
iter = 40

QR-Iteration-with-Shift:
T =
   32.0000   -7.6229  -12.7664   -9.5433  150.4702
    0.0000    8.0000    0.7922    8.7506  -11.1408
    0.0000    0.0000    4.0000    1.2222    3.4860
   -0.0000   -0.0000   -0.0000    2.0000    2.2486
   -0.0000    0.0000   -0.0000   -0.0000    1.0000
iter = 27

Orthogonal-Iteration:
Schur =
   32.0000   -7.6229   12.7664    9.5433  150.4702
   -0.0000    8.0000   -0.7922   -8.7506  -11.1408
   -0.0000   -0.0000    4.0000    1.2222   -3.4860
    0.0000    0.0000   -0.0000    2.0000   -2.2486
    0.0000    0.0000    0.0000    0.0000    1.0000
iter = 43
```

qr_iteration.m

```
function [T,iter] = qr_iteration(A,itmax,tol)

T = A; n = size(T,1);
tnn0 =T(n,n);

for iter = 1:itmax,
    [Q,R] = qr(T);
    T = R*Q;
        tnn1 = T(n,n);
        if abs(tnn1-tnn0)<tol, break
        else, tnn0=tnn1; end
end
```

qr_iteration_shift.m

```
function [T,iter] = qr_iteration_shift(A,itmax,tol)

T = A; n = size(T,1);
tnn0 =T(n,n);

for iter = 1:itmax,
    shift = T(n,n)/2;
    [Q,R] = qr(T - shift*eye(n));
    T = R*Q + shift*eye(n);
        tnn1 = T(n,n);
        if abs(tnn1-tnn0)<tol, break
        else, tnn0=tnn1; end
end
```

orthogonal_iteration.m

```
function [Q,iter] = orthogonal_iteration(A,itmax,tol)

Q = eye(size(A,1));
ynn0 = 1.e20;

for iter = 1:itmax
    Y = A*Q;
    [Q,R] = qr(Y);
        ynn1 = Y(end,end);
        if abs(ynn1-ynn0)<tol, break
        else, ynn0=ynn1; end
end
```

5.5.5. Making QR iteration practical

- **"Practical" QR Algorithm** [23, Part V, p.212]

```
$(Q^{(0)})^T A^{(0)} Q^{(0)} = A$   ($A^{(0)}$ is a tridiag. of $A$)
for $k = 1, 2, \cdots$
    pick a shift $\mu^{(k)}$   (e.g. $\mu^{(k)} = A_{nn}^{(k-1)}$)
    factor $A^{(k-1)} - \mu^{(k)} I = Q^{(k)} R^{(k)}$
    $A^{(k)} = R^{(k)} Q^{(k)} + \mu^{(k)} I$
    if $A_{j,\,j+1}^{(k)} \to 0$
        set $A_{j,\,j+1} = A_{j+1,\,j} = 0$ to obtain
        $\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} = A^{(k)}$
        and now apply the $QR$ to $A_1$, $A_2$
    end if
end for
```
(5.23)

Making QR iteration more practical

- **Issues:**
  - The convergence of the algorithm is **slow**.
  - The the iteration is very **expensive**. At each step, the QR decomposition costs $\mathcal{O}(n^3)$ flops. To find n eigenvalues, we need to have $\mathcal{O}(n^4)$ flops.
- **Improvement:**
  - Reduce the matrix to **upper Hessenberg form**. Apply QR **implicitly**. Cost is reduced from $\mathcal{O}(n^4)$ to $\mathcal{O}(n^3)$.
  - Simultaneous shifts for **complex** e-values.
  - Convergence occurs when subdiagonal entries of A_i are “small enough”.

Details will be described in later subsections.

5.5.6. Hessenberg reduction

Similarity transforms:

Definition 5.27. *Two matrices A, $B \in \mathbb{C}^{n\times n}$ are* **similar** *if there is a nonsingular matrix $S \in \mathbb{C}^{n\times n}$ such that*

$$B = S^{-1}AS. \tag{5.24}$$

Equation (5.24) is called a **similarity transformation** *and S is called the* **transforming matrix***.*

Theorem 5.28. *Similar matrices have the same eigenvalues.*

Proof. See Homework 2. □

Theorem 5.29. *Suppose $B = S^{-1}AS$. Then $\mathbf{v}$ is an eigenvector of A with eigenvalue λ if and only if $S^{-1}\mathbf{v}$ is an eigenvector of B with associated eigenvalue λ.*

Hessenberg reduction: Find an orthogonal matrix Q such that

$$Q^*AQ = H, \tag{5.25}$$

where $H = [h_{ij}]$ is an **upper Hessenberg** matrix, i.e., $h_{ij} = 0$ whenever $i > j+1$.

Hessenberg reduction – via Householder

- **A bad idea**

$$\underset{A}{\begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ X & X & X & X & X \\ X & X & X & X & X \\ X & X & X & X & X \end{bmatrix}} \xrightarrow{Q_1^* \cdot} \underset{Q_1^* A}{\begin{bmatrix} X & X & X & X & X \\ 0 & X & X & X & X \\ 0 & X & X & X & X \\ 0 & X & X & X & X \\ 0 & X & X & X & X \end{bmatrix}} \xrightarrow{\cdot Q_1} \underset{Q_1^* A Q_1}{\begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ X & X & X & X & X \\ X & X & X & X & X \\ X & X & X & X & X \end{bmatrix}}$$

- **A good idea**

$$\underset{A}{\begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ X & X & X & X & X \\ X & X & X & X & X \\ X & X & X & X & X \end{bmatrix}} \xrightarrow{Q_1^* \cdot} \underset{Q_1^* A}{\begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ 0 & X & X & X & X \\ 0 & X & X & X & X \\ 0 & X & X & X & X \end{bmatrix}} \xrightarrow{\cdot Q_1} \underset{Q_1^* A Q_1}{\begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ 0 & X & X & X & X \\ 0 & X & X & X & X \\ 0 & X & X & X & X \end{bmatrix}}$$

- **A good idea** (cont'd)

$$\underset{\boldsymbol{Q_1^* A Q_1}}{\begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ & X & X & X & X \\ & X & X & X & X \\ & X & X & X & X \end{bmatrix}} \xrightarrow{\boldsymbol{Q_2^*}\cdot} \underset{\boldsymbol{Q_2^*}Q_1^* A Q_1}{\begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ & \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} \\ & \mathbf{0} & \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} \\ & \mathbf{0} & \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} \end{bmatrix}} \xrightarrow{\cdot\boldsymbol{Q_2}} \underset{\boldsymbol{Q_1^*}Q_1^* A Q_1\boldsymbol{Q_2}}{\begin{bmatrix} X & X & \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} \\ X & X & \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} \\ & X & \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} \\ & & \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} \\ & & \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} \end{bmatrix}}$$

$$\to \cdots \to \underset{\underbrace{\boldsymbol{Q_{m-2}^* \cdots Q_2^* Q_1^*}}_{\boldsymbol{Q^*}}\, \boldsymbol{A}\, \underbrace{\boldsymbol{Q_1 Q_2 \cdots Q_{m-2}}}_{\boldsymbol{Q}}}{\begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ & X & X & X & X \\ & & X & X & X \\ & & & X & X \end{bmatrix}} = \boldsymbol{H}$$

- **Reference:** [7, 23]

Hessenberg reduction — A general algorithm

Reduction to upper Hessenberg form via Householder

Algorithm 5.30.

```
if $Q$ is desired, set $Q = I$;
for $i = 1 : n-2$
    $\mathbf{u}_i = \mathbf{House}(A(i+1:n,\ i))$;
    $P_i = I - 2\mathbf{u}_i\mathbf{u}_i^T$;    $/ * Q = diag(I_{i\times i}, P_i) * /$
    $A(i+1:n,\ i:n) = P_i \cdot A(i+1:n,\ i:n)$;
    $A(1:n,\ i+1:n) = A(1:n,\ i+1:n) \cdot P_i$;
    if $Q$ is desired
        $Q(i+1:n,\ i:n) = P_i \cdot Q(i+1:n,\ i:n)$;
            $/ * Q = Q_i \cdot Q * /$
    end if
end for
```

- This algorithm **does not form P_i explicitly**, but instead multiplies by $I - 2\mathbf{u}_i\mathbf{u}_i^T$ via matrix vector multiplications, and the $\mathbf{u}_i$ vectors **can be stored below the subdiagonal** (see the QR part of the lec. notes).
- **Proposition 5.31.** *Hessenberg form is preserved by QR iteration.*

Definition 5.32. *A Hessenberg matrix H is* **unreduced** *if all subdiagonals are nonzero.*

- If H is reduced (due to $h_{i+1,i} = 0$), then its eigenvalues are those of its leading $i \times i$ Hessenberg submatrix and its trailing $(n-i)\times(n-i)$ Hessenberg submatrix.

$$H_{9,9} = \left[\begin{array}{cccc|ccccc} x & x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x & x \\ & x & x & x & x & x & x & x & x \\ & & x & x & x & x & x & x & x \\ \hline & & & \mathbf{0} & x & x & x & x & x \\ & & & & x & x & x & x & x \\ & & & & & x & x & x & x \\ & & & & & & x & x & x \\ & & & & & & & x & x \end{array}\right] = \left[\begin{array}{c|c} H_{4,4} & * \\ \hline O & H_{5,5} \end{array}\right]$$

- We will consider **unreduced** Hessenberg only, without loss of generality.

Hessenberg reduction via Householder

- **Costs:** $\frac{10}{3}n^3 + \mathcal{O}(n^2)$, or
  $\frac{14}{3}n^3 + \mathcal{O}(n^2)$ when $Q = Q_{n-1} \cdots Q_1$ is also computed.
- **Convergence** [28]: The computed Hessenberg matrix $\hat{H}$ satisfies

$$\hat{H} = Q^T(A + E)Q,$$

  where Q is orthogonal and

$$||E||_F \leq cn^2\mathbf{u}||Q||_F$$

  with c a small constant.
- The **advantage** of Hessenberg form under QR iteration is that
  - it **costs only** $6n^2 + \mathcal{O}(n)$ flops instead of $\mathcal{O}(n^3)$, and
  - **its form is preserved** so that the matrix remains upper Hessenberg.
- This algorithm is available as
  - **hess**: Matlab
  - **sgehrd**: LAPACK

$\boxed{QR \text{ via Givens, for Hessenberg matrices}}$

- Pictorially, the algorithm may look like

$$\underset{\boldsymbol{H_1}}{\begin{bmatrix} x & x & x & x \\ x & x & x & x \\ & x & x & x \\ & & x & x \end{bmatrix}} \xrightarrow{\boldsymbol{G_1^*}\cdot} \underset{\boldsymbol{G_1^*}H_1}{\begin{bmatrix} \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \\ 0 & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \\ & x & x & x \\ & & x & x \end{bmatrix}} \xrightarrow{\boldsymbol{G_2^*}\cdot} \underset{\boldsymbol{G_2^*}\boldsymbol{G_1^*}H_1}{\begin{bmatrix} \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \\ & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \\ & 0 & \boldsymbol{x} & \boldsymbol{x} \\ & & x & x \end{bmatrix}} \xrightarrow{\boldsymbol{G_3^*}\cdot} \underset{\boxed{\boldsymbol{G_3^*}\boldsymbol{G_2^*}\boldsymbol{G_1^*}H_1 = R}}{\begin{bmatrix} \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \\ & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \\ & & \boldsymbol{x} & \boldsymbol{x} \\ & & 0 & \boldsymbol{x} \end{bmatrix}}$$

$$\underset{\boldsymbol{R}}{\begin{bmatrix} x & x & x & x \\ & x & x & x \\ & & x & x \\ & & & x \end{bmatrix}} \xrightarrow{\cdot\boldsymbol{G_1}} \underset{R\boldsymbol{G_1}}{\begin{bmatrix} \boldsymbol{x} & \boldsymbol{x} & x & x \\ \boldsymbol{x} & \boldsymbol{x} & x & x \\ & & x & x \\ & & & x \end{bmatrix}} \xrightarrow{\cdot\boldsymbol{G_2}} \underset{R\boldsymbol{G_1}\boldsymbol{G_2}}{\begin{bmatrix} \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & x \\ \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & x \\ & \boldsymbol{x} & \boldsymbol{x} & x \\ & & & x \end{bmatrix}} \xrightarrow{\cdot\boldsymbol{G_3}} \underset{R\boldsymbol{G_1}\boldsymbol{G_2}\boldsymbol{G_3} = H_2}{\begin{bmatrix} \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \\ \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \\ & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \\ & & \boldsymbol{x} & \boldsymbol{x} \end{bmatrix}}$$

- $H_2 = R\boldsymbol{G_1}\boldsymbol{G_2}\boldsymbol{G_3} = \boldsymbol{G_3^*}\boldsymbol{G_2^*}\boldsymbol{G_1^*}H_1\boldsymbol{G_1}\boldsymbol{G_2}\boldsymbol{G_3} = Q^*H_1Q$
  $\boxed{Q = \boldsymbol{G_1}\boldsymbol{G_2}\boldsymbol{G_3}}$

QR via Givens, for Hessenberg (cont'd)

Algorithm **5.33.** *Let $A \in \mathbb{R}^{n\times n}$ be* **upper Hessenberg**. *Then the following algorithm overwrites A with*

$$Q^T A = R,$$

where Q is **orthogonal** *and R is* **upper triangular**. *$Q = G_1 \cdots G_{n-1}$ is a product of Givens rotations where G_j has the form $G_j = G(j, j+1, \theta_j)$.*

```
for $j = 1 : n-1$
    $[c, s] =$ givens$(A(j,j), A(j+1,j))$
    $A(j:j+1,\ j:n) = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} A(j:j+1,\ j:n)$
end                                                        (5.26)
for $j = 1 : n-1$
    $A(1:j+1,\ j:j+1) = A(1:j+1,\ j:j+1) \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$
end
```

- This algorithm requires about $2 \cdot 3n^2 = \boldsymbol{6n^2}$ flops.

$\boxed{\textbf{Example}}$ **5.34.** *Find QR decomposition of* $H_1 = \begin{bmatrix} 3 & 1 & 2 \\ 4 & 2 & 3 \\ 0 & .01 & 1 \end{bmatrix}$.

Solution.

$$H_1 \xrightarrow{G_1'\cdot} \underbrace{\begin{bmatrix} 5.0000 & 2.2000 & 3.6000 \\ 0 & 0.4000 & 0.2000 \\ 0 & 0.0100 & 1.0000 \end{bmatrix}}_{G_1'H_1} \xrightarrow{G_2'\cdot} \underbrace{\begin{bmatrix} 5.0000 & 2.2000 & 3.6000 \\ 0 & 0.4001 & 0.2249 \\ 0 & 0 & 0.9947 \end{bmatrix}}_{G_2'G_1'H_1=R}$$

$$R \xrightarrow{\cdot G_1} \underbrace{\begin{bmatrix} 4.7600 & -2.6800 & 3.6000 \\ 0.3201 & 0.2401 & 0.2249 \\ 0 & 0 & 0.9947 \end{bmatrix}}_{RG_1} \xrightarrow{\cdot G_2} \underbrace{\begin{bmatrix} 4.7600 & -2.5892 & 3.6659 \\ 0.3201 & 0.2456 & 0.2189 \\ 0 & 0.0249 & 0.9944 \end{bmatrix}}_{RG_1G_2=H_2}$$

where

$$G_1 = \begin{bmatrix} .6 & -.8 & 0 \\ .8 & .6 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad G_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & .9997 & -.0250 \\ 0 & .0250 & .9997 \end{bmatrix}.$$

Then,

$$H_2 = R\boldsymbol{G_1}\boldsymbol{G_2} = \boldsymbol{G_2'}\boldsymbol{G_1'}H_1\boldsymbol{G_1}\boldsymbol{G_2} = Q'H_1Q$$

and $Q = \boldsymbol{G_1}\boldsymbol{G_2}$, which does not have to be constructed explicitly in practice.

5.5.7. Tridiagonal and bidiagonal reduction

- **Tridiagonal reduction:** If A is **symmetric**, the Hessenberg reduction process leaves A symmetric at each step, so zeros are created in symmetric positions.

$$\underbrace{\begin{bmatrix} \boldsymbol{*} & * & 0 & 0 & 0 & 0 & 0 & 0 \\ * & \boldsymbol{*} & * & 0 & 0 & 0 & 0 & 0 \\ 0 & * & \boldsymbol{*} & * & 0 & 0 & 0 & 0 \\ 0 & 0 & * & \boldsymbol{*} & * & 0 & 0 & 0 \\ 0 & 0 & 0 & * & \boldsymbol{*} & * & 0 & 0 \\ 0 & 0 & 0 & 0 & * & \boldsymbol{*} & * & 0 \\ 0 & 0 & 0 & 0 & 0 & * & \boldsymbol{*} & * \\ 0 & 0 & 0 & 0 & 0 & 0 & * & \boldsymbol{*} \end{bmatrix}}_{\text{Tridiagonal Matrix}} \qquad \underbrace{\left[\begin{array}{ccccc} \boldsymbol{*} & * & 0 & 0 & 0 \\ 0 & \boldsymbol{*} & * & 0 & 0 \\ 0 & 0 & \boldsymbol{*} & * & 0 \\ 0 & 0 & 0 & \boldsymbol{*} & * \\ 0 & 0 & 0 & 0 & \boldsymbol{*} \\ \hline 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right]}_{\text{Bidiagonal Matrix}}$$

- **Bidiagonal reduction:** Compute **orthogonal** matrices Q and V such that QAV is bidiagonal.

Tridiagonal reduction algorithm

- **Algorithm 5.35. (Householder tridiagonalization algorithm).** *If* $A \in \mathbb{R}^{n\times n}$ *is* **symmetric**, *then the following algorithm overwrites* A *with* $T = Q^TAQ$, *where* T *is tridiagonal and* $Q = H_1 \cdots H_{n-2}$ *is the product of the Householder transformations.*

```
for $k = 1 : n-2$
    $[\mathbf{v}, \beta] = \mathbf{house}(A(k+1:n,\ k))$
    $\mathbf{p} = \beta A(k+1:n,\ k+1:n)\mathbf{v}$
    $\mathbf{w} = \mathbf{p} - (\beta \mathbf{p}^T\mathbf{v}/2)\mathbf{v}$                                   (5.27)
    $A(k+1,\ k) = ||A(k+1:n,\ k)||_2;\ A(k,\ k+1) = A(k+1,\ k);$
    $A(k+1:n,\ k+1:n) = A(k+1:n,\ k+1:n) - \mathbf{v}\mathbf{w}^T - \mathbf{w}\mathbf{v}^T$
end
```

- The matrix Q can be stored in factored form in the subdiagonal portion of A.
- **Cost:** $\frac{4}{3}n^3 + \mathcal{O}(n^2)$, or
  $\frac{8}{3}n^3 + \mathcal{O}(n^2)$ to form $Q_{n-1} \cdots Q_1$ as well.
- LAPACK routine: **ssytrd**.

- **Example:**

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & .6 & .8 \\ 0 & .8 & -.6 \end{bmatrix}^T \begin{bmatrix} 1 & 3 & 4 \\ 3 & 2 & 8 \\ 4 & 8 & 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & .6 & .8 \\ 0 & .8 & -.6 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 0 \\ 5 & 10.32 & 1.76 \\ 0 & 1.76 & -5.32 \end{bmatrix}$$

- Note that if T has a zero subdiagonal, then the eigenproblem splits into a pair of smaller eigenproblems. In particular, if $t_{k+1,k} = 0$, then

$$\sigma(T) = \sigma(T(1:k, 1:k)) \cup \sigma(k+1:n, k+1:n)).$$

  If T has no zero subdiagonal entries, it is unreduced.

- Let $\hat{T}$ denote the computed version of T obtained by Algorithm 5.35. Then

$$\hat{T} = \tilde{Q}^T (A + E) \tilde{Q},$$

  where $\tilde{Q}$ is exactly orthogonal and E is symmetric matrix satisfying $||E||_F \le c\mathbf{u}||A||_F$ where c is small constant [28].

Bidiagonal reduction

- **Bidiagonal reduction:** Compute **orthogonal** matrices Q and V such that QAV is bidiagonal.

$$A_{7\times 5} \to \begin{bmatrix} * & * & 0 & 0 & 0 \\ 0 & * & * & 0 & 0 \\ 0 & 0 & * & * & 0 \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * \\ \hline 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \qquad A_{5\times 5} \to \begin{bmatrix} * & * & 0 & 0 & 0 \\ 0 & * & * & 0 & 0 \\ 0 & 0 & * & * & 0 \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * \end{bmatrix}$$

- If A is $n \times n$, and we get **orthogonal** matrices $Q = Q_{n-1} \cdots Q_1$, $V = V_1 \cdots V_{n-2}$, such that

$$QAV = \tilde{A} \tag{5.28}$$

is **upper bidiagonal**.

  - **Eigenvalues/eigenvectors**:

$$\tilde{A}^T\tilde{A} = V^TA^TQ^TQAV = V^TA^TAV$$

$\Longrightarrow$ $\tilde{A}^T\tilde{A}$ has the same **eigenvalues** as A^TA;
$\Longrightarrow$$\tilde{A}$ has the **same singular values** as A.

  - **Cost**: $\frac{8}{3}n^3 + \mathcal{O}(n^2)$, & $4n^3 + \mathcal{O}(n^2)$ to compute Q and V.
  - LAPACK routine `sgebrd`.

- **Bidiagonal Reduction Process:**

$$\underset{A}{\begin{bmatrix} X & X & X & X \\ X & X & X & X \\ X & X & X & X \\ X & X & X & X \end{bmatrix}} \xrightarrow{\boldsymbol{Q_1^T}\cdot} \underset{\boldsymbol{Q_1^T}A}{\begin{bmatrix} \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} \\ \mathbf{0} & \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} \\ \mathbf{0} & \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} \\ \mathbf{0} & \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} \end{bmatrix}} \xrightarrow{\cdot\boldsymbol{V_1}} \underset{\boldsymbol{Q_1^T}A\boldsymbol{V_1}}{\begin{bmatrix} X & \boldsymbol{X} & \mathbf{0} & \mathbf{0} \\ 0 & \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} \\ 0 & \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} \\ 0 & \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} \end{bmatrix}}$$

$\boldsymbol{Q_1}$ is a **Householder refection**, and $\boldsymbol{V_1}$ is a **Householder refection** such that leaves the 1st column of Q_1A **unchanged**.

$$\xrightarrow{\boldsymbol{Q_2^T}\cdot} \underset{\boldsymbol{Q_2^T}\boldsymbol{Q_1^T}A\boldsymbol{V_1}}{\begin{bmatrix} X & X & 0 & 0 \\ 0 & \boldsymbol{X} & \boldsymbol{X} & \boldsymbol{X} \\ 0 & \mathbf{0} & \boldsymbol{X} & \boldsymbol{X} \\ 0 & \mathbf{0} & \boldsymbol{X} & \boldsymbol{X} \end{bmatrix}} \xrightarrow{\cdot\boldsymbol{V_2}} \underset{\boldsymbol{Q_2^T}\boldsymbol{Q_1^T}A\boldsymbol{V_1}\boldsymbol{V_2}}{\begin{bmatrix} X & X & 0 & 0 \\ 0 & X & \boldsymbol{X} & \mathbf{0} \\ 0 & 0 & \boldsymbol{X} & \boldsymbol{X} \\ 0 & 0 & \boldsymbol{X} & \boldsymbol{X} \end{bmatrix}} \xrightarrow{\boldsymbol{Q_3^T}\cdot} \underset{\boldsymbol{Q_3^T}\boldsymbol{Q_2^T}\boldsymbol{Q_1^T}A\boldsymbol{V_1}\boldsymbol{V_2} = \boldsymbol{B}}{\begin{bmatrix} X & X & 0 & 0 \\ 0 & X & X & 0 \\ 0 & 0 & \boldsymbol{X} & \boldsymbol{X} \\ 0 & 0 & \mathbf{0} & \boldsymbol{X} \end{bmatrix}}$$

- $A \to Q^TAV = B$ — **upper bidiagonal**, where

$$Q = \boldsymbol{Q_1}\boldsymbol{Q_2}\boldsymbol{Q_3}, \quad V = \boldsymbol{V_1}\boldsymbol{V_2}.$$

Algorithm 5.36. (Householder bidiagonalization algorithm) [10]. Given $A \in \mathbb{R}^{m\times n}$ $(m \ge n)$, the following algorithm overwrite A with

$$Q_B^T A V_B = B,$$

where B is upper bidiagonal and $Q_B = Q_1 \cdots Q_n$, and $V_B = V_1 \cdots V_{n-2}$. The essential part of Q_j's Householder vector is stored in $A(j+1:m,\ j)$ and the essential part of V_j's Householder vector is stored in $A(j,\ j+2:n)$.

```
for $j = 1 : n-1$
    $[\mathbf{v}, \beta] =$ house$(A(j:m, j))$
    $A(j:m,\ j:n) = (I_{m-j+1} - \beta \mathbf{v}\mathbf{v}^T) A(j:m,\ j:n)$
    $A(j+1:m,\ j) = \mathbf{v}(2:m-j+1)$
    if $j \le n-2$
        $[\mathbf{v}, \beta] =$ house$(A(j,\ j+1:n)^T)$
        $A(j:m,\ j+1:n) = A(j:m,\ j+1:n)(I_{n-j} - \beta \mathbf{v}\mathbf{v}^T)$
        $A(j,\ j+2:n) = \mathbf{v}(2:n-j)^T$
    end
end
```
(5.29)

- This algorithm requires about $4mn^2 - 4n^3/3$ flops.

Example 5.37. *If the Householder Bidiagonal algorithm is applied to A, and correct to 4 significant digits, then*

$$A = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix} \to B \approx \begin{bmatrix} 5.477 & 23.80 & 0. \\ 0. & 7.264 & -.7515 \\ 0. & 0. & 0. \\ 0. & 0. & 0. \end{bmatrix}, \quad Q^T AV = B$$

$$V \approx \begin{bmatrix} 1. & 0. & 0. \\ 0. & .5369 & .8437 \\ 0. & .8437 & -.5369 \end{bmatrix}, \quad Q \approx \begin{bmatrix} .1826 & .8165 & -.5400 & .0917 \\ .3651 & .4082 & .7883 & .2803 \\ .5477 & -.0000 & .0434 & -.8355 \\ .7303 & -.4082 & -.2917 & .4636 \end{bmatrix}$$

- ***R*-Bidiagonalization:** If $m \gg n$, we first transform A to an upper triangular matrix first,

  $$Q^T A = \begin{bmatrix} R_1 \\ 0 \end{bmatrix},$$

  and then apply the Householder Bidiagonalization Algorithm to the square upper triangular matrix R_1.

  Total cost: $2mn^2 + 2n^3$, which is less than $4mn^2 - 4n^3/3$ whenever $m \geq 5n/3$.

5.5.8. QR iteration with implicit shifts – Implicit Q theorem

In this subsection, we show how to implement QR iteration effectively for an upper Hessenberg matrix H.

- The implementation is *implicit* in the sense that we do not explicitly compute the QR factorization of H, but rather construct Q as a product of Givens rotations and simple orthogonal matrices.
- the *implicit* **Q** *theorem* shows that this implicitly constructed Q is the Q we want.

$\boxed{\textbf{Theorem}}$ **5.38. (Implicit Q theorem).** *Suppose that $Q^T AQ = H$ is* **unreduced upper Hessenberg***. Then columns 2 through n of Q are determined* **uniquely** *(up to signs) by the first column of Q.*

Implication of the theorem: To compute $A_{i+1} = Q_i^T A_i Q_i$ from A_i (in the QR iteration), we will need only to

- compute the 1st column of Q_i (which is parallel to the 1st column of $A_i - \sigma_i I$ and so can be gotten just by normalizing this column vector).
- choose other columns of Q_i such that Q_i is orthogonal and A_{i+1} is unreduced Hessenberg.

Proof. (Sketch of the Proof of Implicit Q theorem). Let

$$Q^T AQ = H, \quad V^T AV = G$$

be unreduced upper Hessenberg, Q and V be orthogonal, and the 1st columns of Q and V are equal. Define

$$W = V^T Q = [\mathbf{w}_1, \cdots, \mathbf{w}_n].$$

Then W is orthogonal and $GW = WH$ and therefore we have

$$h_{i+1,i}\mathbf{w}_{i+1} = G\mathbf{w}_i - \sum_{j=1}^{i} h_{ji}\mathbf{w}_j, \quad i = 1, 2, \cdots, n-1$$

Since $\mathrm{w}_1 = \mathrm{e}_1$ and G is upper Hessenberg, W is upper triangular. Since W is also orthogonal, W is diagonal and $W = \mathrm{diag}(\pm 1, \cdots, \pm 1)$. □

Implicit Single Shift QR Algorithm

It is also called the **bulge-and-chase** algorithm.

Example 5.39. *To see how to use the implicit Q theorem to compute A_1 from $A_0 = A$, we will use a 5×5 matrix.*

1. *Choose G_1 such that $G_1^T A G_1 = A_1$:*

$$\underbrace{\begin{bmatrix} c_1 & s_1 & & & \\ -s_1 & c_1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}}_{G_1^T} \underbrace{\begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ & X & X & X & X \\ & & X & X & X \\ & & & X & X \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} c_1 & -s_1 & & & \\ s_1 & c_1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}}_{G_1} = \underbrace{\begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ \oplus & X & X & X & X \\ & & X & X & X \\ & & & X & X \end{bmatrix}}_{A_1}$$

   * *We will discuss how to choose c_1 and s_1 below; for now they may be any Givens rotation.*

   * *The $\oplus$ position (3,1) is called a **bulge** and needs to be eliminated to restore Hessenberg form.*

2. *Choose G_2 such that $G_2^T A_1 G_2 = A_2$:*

$$\underbrace{\begin{bmatrix} 1 & & & & \\ & c_2 & s_2 & & \\ & -s_2 & c_2 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}}_{G_2^T} \underbrace{\begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ \oplus & X & X & X & X \\ & & X & X & X \\ & & & X & X \end{bmatrix}}_{A_1} \underbrace{\begin{bmatrix} 1 & & & & \\ & c_2 & -s_2 & & \\ & s_2 & c_2 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}}_{G_2} = \underbrace{\begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ & X & X & X & X \\ & \oplus & X & X & X \\ & & & X & X \end{bmatrix}}_{A_2}$$

   *Thus the bulge has been **chased** from (3,1) to (4,2).*

3. *Choose G_3 such that $G_3^T A_2 G_3 = A_3$:*

$$\underbrace{\begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & c_3 & s_3 & \\ & & -s_3 & c_3 & \\ & & & & 1 \end{bmatrix}}_{G_3^T} \underbrace{\begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ & X & X & X & X \\ & \oplus & X & X & X \\ & & & X & X \end{bmatrix}}_{A_2} \underbrace{\begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & c_3 & -s_3 & \\ & & s_3 & c_3 & \\ & & & & 1 \end{bmatrix}}_{G_3} = \underbrace{\begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ & X & X & X & X \\ & & X & X & X \\ & & \oplus & X & X \end{bmatrix}}_{A_3}$$

The bulge has been chased from (4,2) to (5,3).

4. *Choose G_4 such that $G_4^T A_3 G_4 = A_4$:*

$$\underbrace{\begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & c_4 & s_4 \\ & & & -s_4 & c_4 \end{bmatrix}}_{G_4^T} \underbrace{\begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ & X & X & X & X \\ & & X & X & X \\ & & \oplus & X & X \end{bmatrix}}_{A_3} \underbrace{\begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & c_4 & -s_4 \\ & & & s_4 & c_4 \end{bmatrix}}_{G_4} = \underbrace{\begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ & X & X & X & X \\ & & X & X & X \\ & & & X & X \end{bmatrix}}_{A_4}$$

So, we are back to the upper Hessenberg form.

- *Let $Q = G_1 G_2 G_3 G_4$. Then, $Q^T A Q = A_4$.*
  The first columns of Q is $[c_1, s_1, 0, \cdots, 0]^T$. which by implicit Q theorem has uniquely determined the other columns of Q (up to signs).
- *We now choose the first column of Q to be proportional to the first column of $A - \sigma I$, $[a_{11} - \sigma, a_{21}, 0, \cdots, 0]^T$.*
  Then, Q is the same as in the QR of $A - \sigma I$!
- **Cost:** $6n^2 + \mathcal{O}(n)$ *for any $n \times n$ matrix A.*

Example 5.40. *Use the implicit single shift QR algorithm to find the QR factorization of*

$$A = \begin{bmatrix} 1 & 6 & 11 & 16 & 21 \\ 2 & 7 & 12 & 17 & 22 \\ 0 & 8 & 13 & 18 & 23 \\ 0 & 0 & 14 & 19 & 24 \\ 0 & 0 & 0 & 20 & 25 \end{bmatrix}. \tag{5.30}$$

Solution.

- **Step 1:** **bulge** $\oplus = -7.1554$

```
G1 =
   -0.4472    0.8944         0         0         0
   -0.8944   -0.4472         0         0         0
         0         0    1.0000         0         0
         0         0         0    1.0000         0
         0         0         0         0    1.0000
G1'*A0 =
   -2.2361   -8.9443  -15.6525  -22.3607  -29.0689
         0    2.2361    4.4721    6.7082    8.9443
         0    8.0000   13.0000   18.0000   23.0000
         0         0   14.0000   19.0000   24.0000
         0         0         0   20.0000   25.0000
A1 = G1'*A0*G1 =
    9.0000    2.0000  -15.6525  -22.3607  -29.0689
   -2.0000   -1.0000    4.4721    6.7082    8.9443
   -7.1554   -3.5777   13.0000   18.0000   23.0000
         0         0   14.0000   19.0000   24.0000
         0         0         0   20.0000   25.0000
```

- **Step 2:** **bulge** $\oplus = -13.4832$

```
G2 =
    1.0000         0         0         0         0
         0   -0.2692    0.9631         0         0
         0   -0.9631   -0.2692         0         0
         0         0         0    1.0000         0
         0         0         0         0    1.0000
G2'*A1 =
    9.0000    2.0000  -15.6525  -22.3607  -29.0689
    7.4297    3.7148  -13.7240  -19.1414  -24.5587
         0         0    0.8076    1.6151    2.4227
         0         0   14.0000   19.0000   24.0000
         0         0         0   20.0000   25.0000
A2 = G2'*A1*G2 =
    9.0000   14.5363    6.1397  -22.3607  -29.0689
    7.4297   12.2174    7.2721  -19.1414  -24.5587
         0   -0.7778   -0.2174    1.6151    2.4227
         0  -13.4832   -3.7687   19.0000   24.0000
         0         0         0   20.0000   25.0000
```

- **Step 3:** **bulge** $\oplus = -19.9668$

```
G3 =
    1.0000         0         0         0         0
         0    1.0000         0         0         0
         0         0   -0.0576    0.9983         0
         0         0   -0.9983   -0.0576         0
         0         0         0         0    1.0000
G3'*A2 =
    9.0000   14.5363    6.1397  -22.3607  -29.0689
    7.4297   12.2174    7.2721  -19.1414  -24.5587
         0   13.5056    3.7749  -19.0615  -24.0997
         0         0         0    0.5183    1.0366
```

```
            0          0          0    20.0000    25.0000
A3 = G3'*A2*G3 =
       9.0000    14.5363    21.9700     7.4172   -29.0689
       7.4297    12.2174    18.6908     8.3623   -24.5587
            0    13.5056    18.8125     4.8664   -24.0997
            0          0    -0.5174    -0.0298     1.0366
            0          0   -19.9668    -1.1518    25.0000
```

- **Step 4:** **chasing bulge** is completed.

```
G4 =
       1.0000          0          0          0          0
            0     1.0000          0          0          0
            0          0     1.0000          0          0
            0          0          0    -0.0259     0.9997
            0          0          0    -0.9997    -0.0259
G4'*A3 =
       9.0000    14.5363    21.9700     7.4172   -29.0689
       7.4297    12.2174    18.6908     8.3623   -24.5587
            0    13.5056    18.8125     4.8664   -24.0997
            0          0    19.9735     1.1521   -25.0185
            0          0          0          0     0.3886
A4 = G4'*A3*G4 =
       9.0000    14.5363    21.9700    28.8670     8.1678
       7.4297    12.2174    18.6908    24.3338     8.9957
            0    13.5056    18.8125    23.9655     5.4891
            0          0    19.9735    24.9802     1.7999
            0          0          0    -0.3885    -0.0101
```

Thus, $Q^T AQ = A_4$, where

$$\begin{aligned} Q &= G_1 G_2 G_3 G_4 \\ &= \begin{bmatrix} \mathbf{-0.4472} & -0.2408 & -0.0496 & -0.0223 & 0.8597 \\ \mathbf{-0.8944} & 0.1204 & 0.0248 & 0.0111 & -0.4298 \\ 0 & \mathbf{-0.9631} & 0.0155 & 0.0070 & -0.2687 \\ 0 & 0 & \mathbf{-0.9983} & 0.0015 & -0.0576 \\ 0 & 0 & 0 & \mathbf{-0.9997} & -0.0259 \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{c_1} & -0.2408 & -0.0496 & -0.0223 & 0.8597 \\ \boldsymbol{s_1} & 0.1204 & 0.0248 & 0.0111 & -0.4298 \\ 0 & \boldsymbol{s_2} & 0.0155 & 0.0070 & -0.2687 \\ 0 & 0 & \boldsymbol{s_3} & 0.0015 & -0.0576 \\ 0 & 0 & 0 & \boldsymbol{s_4} & -0.0259 \end{bmatrix} \end{aligned}$$

and therefore $\boxed{A_4 = RQ}$, when $A = QR$.

Implicit Double Shift QR Algorithm

Double shifting has been considered to maintain real arithmetic, for computing the eigenvalues of companion and fellow matrices.

- Companion and fellow matrices are Hessenberg matrices, that can be decomposed into the sum of a unitary and a rank-1 matrix.
- The Hessenberg, the unitary as well as the rank-1 structures are preserved under a step of the QR-method. This makes these matrices suitable for the design of a fast QR-method.
- Several techniques already exist for performing a QR-step. The implementation of these methods is highly dependent on the representation used.

Definition 5.41. Given an nth-orer monic polynomial

$$a(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} + x^n, \tag{5.31}$$

its **companion matrix** is defined as

$$A = \begin{bmatrix} 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & \cdots & 0 & -a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -a_{n-1} \end{bmatrix}. \tag{5.32}$$

The eigenvalues of A are the roots of the polynomial $a(x)$.

Definition 5.42. Fellow matrices are rank-1 perturbations of unitary Hessenberg matrices. That is, a fellow matrix F is expressed as

$$F = H + \mathbf{u}\mathbf{v}^T, \tag{5.33}$$

where H is a unitary Hessenberg matrix and **u** and **v** are vectors.

5.6. Symmetric Eigenvalue Problems

5.6.1. Algorithms for the symmetric eigenproblem – Direct methods

1. **Tridiagonal QR iteration:** This algorithm finds all the eigenvalues, and optionally all the eigenvectors, of a symmetric tridiagonal matrix.
   - Implemented efficiently, it is currently the **fastest practical method** to find all the eigenvalues of a symmetric tridiagonal matrix, taking $\mathcal{O}(n^2)$ flops.
   - But for finding all the eigenvectors as well, QR iteration takes a little over $6n^3$ flops on average and is only the fastest algorithm for **small** matrices, up to about $n = 25$.

   Matlab command: **eig**
   LAPACK routines:
   ssyev (for dense matrices),
   sstev (for tridiagonal matrices).

2. **Rayleigh quotient iteration:** This algorithm underlies the QR iteration. But it can be used separately (without assuming that the matrix has first been reduced to tridiagonal form).
3. **Divide-and-conquer method:** This is currently the fastest method to find all the eigenvalues and eigenvectors of symmetric tridiagonal matrices larger than $n = 25$.

   LAPACK routines:

   sstevd, defaults to QR iteration for smaller matrices.

   In the worst case, divide-and-conquer requires $\mathcal{O}(n^3)$ flops, but in practice the constant is quite small. Over a large set of random test cases, it appears to take only $\mathcal{O}(n^{2.3})$ flops in average, and as low as $\mathcal{O}(n^2)$ for some eigenvalue distributions.

   In theory, divide-and-conquer could be implemented to run in $\mathcal{O}(n \cdot \log^p n)$.

4. **Bisection and inverse iteration:** Bisection may be used to find just a subset of the eigenvalues of a symmetric tridiagonal matrix, say, those in an interval $[a, b]$ or $[\alpha_i, \alpha_{i-j}]$.

   It needs only $\mathcal{O}(nk)$ flops, where k is the number of eigenvalues desired. Thus Bisection can be much faster than QR iteration when $k \ll n$, since QR iteration requires $\mathcal{O}(n^2)$ flops.

   Inverse iteration can then be used to find the corresponding eigenvectors. In the best case, when the eigenvalues are "well separated" inverse iteration also costs only $\mathcal{O}(nk)$ flops.

   But in the worst case, when many eigenvalues are clustered close together, inverse iteration takes $\mathcal{O}(nk^2)$ flops and **does not even guarantee the accuracy**.

5. **Jacobi's method:** This method is historically the oldest method for the eigenproblem, dating to 1846. It is usually much slower than any of the above methods, taking $\mathcal{O}(n^3)$ flops with a large constant. But it is sometimes much more accurate than the above methods. It does not require tridiagonalization of the matrix.

5.6.2. Tridiagonal QR iteration

- **Tridiagonal QR iteration:**

  1. Given $A = A^T$, use a variant of the Householder tridiagonalization algorithm (Algorithm 5.35 on p. 216) to find orthogonal Q so that

  $$QAQ^T = T$$

     is tridiagonal.

  2. Apply QR iteration to T to get a sequence

  $$T = T_0,\, T_1,\, T_2, \cdots$$

     of tridiagonal matrices converging to diagonal form.

- The QR iteration keeps all the T_i tridiagonal. Indeed, since QAQ^T is symmetric and upper Hessenberg, it must also be lower Hessenberg, i.e., tridiagonal. This keeps each the QR iteration very inexpensive.

- **Tridiagonal QR iteration – Operation count:**
  - Reducing A to symmetric tridiagonal form T costs $\frac{4}{3}n^3 + \mathcal{O}(n^2)$ flops, or $\frac{8}{3}n^3 + \mathcal{O}(n^2)$ flops if eigenvectors are also desired.
  - One tridiagonal QR iteration with a single shift ("bulge chasing") costs $6n$ flops.
  - Finding all eigenvalues of T takes only 2 QR steps per eigenvalue on average, for a total of $6n^2$ flops.
  - Finding all eigenvalues and eigenvectors of T costs $6n^3 + \mathcal{O}(n^2)$ flops.
  - **The total cost** to find just **eigenvalues** of A is $\dfrac{4}{3}n^3 + \mathcal{O}(n^2)$ flops.
  - **The total cost** to find all the **eigenvalues and eigenvectors** of A is $8\dfrac{2}{3}n^3 + \mathcal{O}(n^2)$ flops.

- **Tridiagonal QR iteration – How to chose the shifts:** Denote the ith iterate by $T_i = \begin{bmatrix} a_1 & b_1 & & \\ b_1 & \ddots & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & b_{n-1} & a_n \end{bmatrix}$.
  - The simplest choice of shift would be $\sigma_i = a_n$; this is the single shift QR iteration (see § 5.5.8). It is **cubically convergent** for almost all matrices.
  - **Wilkinson's shift**: Let the shift σ_i be the eigenvalue of $\begin{bmatrix} a_{n-1} & b_{n-1} \\ b_{n-1} & a_n \end{bmatrix}$ that is closest to a_n.
  - **Wilkinson Theorem**: QR iteration with Wilkinson's shift is globally, and at least linearly, convergent. It is asymptotically cubically convergent for almost all matrices.

Tridiagonal QR iteration with Wilkinson's shift

- **Matlab Program:** $T \in \mathbb{R}^{n\times n}$ — symmetric tridiagonal, m — number of iterations

```
% Dr. James Demmel's matlab code tridiQR.m % (slightly modified)
%
% Store last subdiagonal and last diagonal entries of QR iterates in NN
NN = [ T(n,n-1), T(n,n) ];
% Perform QR iteration with Wilkinson's shift
for i = 1:m
% Compute the shift
  lc = T(n-1:n, n-1:n);  elc = eig(lc);  %calculate the e-values of lc
  if ( abs(T(n,n)-elc(1)) < abs(T(n,n)-elc(2)) )
      shift = elc(1);
  else
      shift = elc(2);
  end
% Perform QR iteration; enforce symmetry explicitly
  [q,r] = qr(T - shift*eye(n)); T = r*q + shift*eye(n);
  T = tril(triu(T,-1),1); T = (T + T')/2;
% Update NN
  NN = [NN;[T(n,n-1),T(n,n)]]; %append T(n,n-1),T(n,n) to NN
end
NN = [NN,NN(:,2)-NN(m+1,2)];
```

Example 5.43. *Here is an illustration of the convergence of the tridiagonal QR iteration, starting with the tridiagonal matrix $T = T_0$:*

$$\begin{aligned} T_0 &= \begin{bmatrix} 0.2493 & 1.2630 & 0 & 0 \\ 1.2630 & 0.9688 & -0.8281 & 0 \\ 0 & -0.8281 & 0.4854 & -3.1883 \\ 0 & 0 & -3.1883 & -0.9156 \end{bmatrix} \\ &\to \begin{bmatrix} 1.9871 & 0.7751 & 0 & 0 \\ 0.7751 & 1.7049 & -1.7207 & 0 \\ 0 & -1.7207 & 0.6421 & -0.0000 \\ 0 & 0 & -0.0000 & \mathbf{-3.5463} \end{bmatrix} = T_3 \end{aligned} \quad \textbf{(5.34)}$$

- The following table shows the last off-diagonal entry of each T_i, the last diagonal entry of T_i, and the difference between the last diagonal entry and its ultimate value ($\alpha \approx -3.54627$). The cubic convergence of the error to zero (in the last column) is evident.

i	$T_i(4,3)$	$T_i(4,4)$	$T_i(4,4) - \alpha$
1	−3.188300000000000	−0.915630000000000	2.630637193722710
2	−0.056978959466637	−3.545728453189497	0.000538740533214
3	−0.000000246512081	−3.546267193722698	0.000000000000012

- By the way,

$$\textbf{eig}(T_0) = [\mathbf{-3.5463} \quad -0.7091 \quad 1.7565 \quad 3.2868]$$

5.6.3. Rayleigh quotient iteration

- $\boxed{\textbf{Algorithm}}$ **5.44. (Rayleigh quotient iteration).**
Given $\mathbf{x}_0$ *with* $||\mathbf{x}_0||_2 = 1$*, and a user-supplied stopping tolerance tol, we iterate*

```
$\rho_0 = \rho(\mathbf{x}_0, A) = \dfrac{\mathbf{x}_0^T A \mathbf{x}_0}{\mathbf{x}_0^T \mathbf{x}_0}$
$i = 0$
repeat
    $\mathbf{y}_i = (A - \rho_{i-1} I)^{-1} \mathbf{x}_{i-1}$                    (5.35)
    $\mathbf{x}_i = \mathbf{y}_i / ||\mathbf{y}_i||_2$
    $\rho_i = \rho(\mathbf{x}_i, A)$
    $i = i + 1$
until convergence $(||A\mathbf{x}_i - \rho_i \mathbf{x}_i||_2 < tol)$
```

- $\boxed{\textbf{Theorem}}$ **5.45.** *Rayleigh quotient iteration is locally* **cubically convergent**. *That is, the number of correct digits* **triples** *at each step once the error is small enough and the eigenvalue is simple.*

5.6.4. Divide-and-conquer

- **Divide-and-conquer** is the **fastest** algorithm so far to find all e-values and e-vectors of a tridiagonal matrix (dim ≥ 25).
- The idea is to compute the Schur decomposition

$$Q^T T Q = \Lambda = \text{diag}(\lambda_1, \cdots, \lambda_n), \quad Q^T Q = I, \tag{5.36}$$

  for tridiagonal T by

  * "tearing" T in half;
  * computing the Schur decompositions of the 2 parts;
  * combining the 2 half-sized Schur decompositions into the required full size Schur decomposition.

  The overall procedure is suitable for **parallel computation**.

Exercises for Chapter 5

5.1. Let, for $n \geq 2$ and for a small $\varepsilon > 0$,

$$A = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & 0 & 1 \\ \varepsilon & & & 0 \end{bmatrix} \in \mathbb{R}^{n\times n}. \tag{5.37}$$

(a) Verify that the characteristic polynomial of A is given by

$$\phi(\lambda) = \lambda^n - \varepsilon = 0.$$

So $\lambda = \sqrt[n]{\varepsilon}$ (n possible values).

(b) Verify that

$$\left.\frac{d\lambda}{d\varepsilon}\right|_{\varepsilon=0} = \infty. \tag{5.38}$$

This implies that the condition number of λ (a multiple eigenvalue) is infinite.

Note: The eigenvalue λ (the nth root of ε) grows much faster than any multiple of ε when ε is small. Thus the condition number of λ must be large. More formally, you can prove that the condition number of λ is infinite by using a calculus technique, as in (5.38).

5.2. Prove Theorem 5.28 on page 206.

Hint: Let A and B are similar, i.e., $B = S^{-1}AS$. To show that A and B have the same eigenvalues, it suffices to show that they have the same characteristic polynomial. Thus what you have to do is to show

$$\det(B - \lambda I) = \det(S^{-1}AS - \lambda I). \tag{5.39}$$

5.3. Let $A = \begin{bmatrix} 1 & 1 & 1 \\ -1 & 9 & 2 \\ 0 & -1 & 2 \end{bmatrix}$.

(a) Using Matlab, apply the inverse power iteration with the shift $\sigma = 9$, starting with $\mathbf{x}_0 = [1\ 1\ 1]^T$. For simplicity, just form

$$B = (A - 9I)^{-1} \quad \texttt{(B=inv(A-9*eye(3));)}$$

and then iterate at least 10 iterations with B.

(b) Use `[V,D]=eig(A)` to get the true dominant eigenvector $\mathbf{v}$. Calculate the errors and ratios

$$||\mathbf{x}_j - \mathbf{v}||_2, \quad \frac{||\mathbf{x}_{j+1} - \mathbf{v}||_2}{||\mathbf{x}_j - \mathbf{v}||_2}; \quad j = 1, 2, \cdots .$$

Compute the theoretical convergence rate (from the known eigenvalues) and compare it with these errors/ratios.

5.4. Let

$$A = \begin{bmatrix} -31 & -35 & 16 \\ -10 & -8 & 4 \\ -100 & -104 & 49 \end{bmatrix}. \tag{5.40}$$

Then, we have its spectrum $\sigma(A) = \{9, 3, -2\}$. Applying the orthogonal iteration, Algorithm 5.22, with $\mathbf{Z}_0 = [\mathbf{e}_1\ \mathbf{e}_2]$.

5.5. This problem is concerned with the QR iteration for the same matrix A in Homework 4.

(a) Apply Algorithm 5.24 to find eigenvalues of A.
(b) Apply Algorithm 5.26, with the shift $\sigma_i = A_i(3,3)$, to find eigenvalues of A.
(c) Compare their convergence speeds.

5.6. Consider the matrix in Example 5.40

$$A = \begin{bmatrix} 1 & 6 & 11 & 16 & 21 \\ 2 & 7 & 12 & 17 & 22 \\ 0 & 8 & 13 & 18 & 23 \\ 0 & 0 & 14 & 19 & 24 \\ 0 & 0 & 0 & 20 & 1 \end{bmatrix}.$$

(a) Use `[V,D]=eig(A)` to get the true eigenvalues and eigenvectors of A.
(b) Use the implicit single shift QR iteration to find eigenvalues of A. Compare the results with the true eigenvalues.

CHAPTER **6**

Iterative Methods for Linear Systems

Contents of Chapter 6

6.1. A Model Problem

This section is a revisit of Section 2.1.

Let $\Omega = (a_x, b_x) \times (a_y, b_y)$ in 2D space. Consider the model problem

$$\begin{array}{ll} \text{(a)} & -\nabla \cdot (a\nabla u) + cu = f, \quad \mathbf{x} \in \Omega \\ \text{(b)} & \qquad au_\nu + \beta u = g, \qquad \mathbf{x} \in \Gamma, \end{array} \tag{6.1}$$

where the diffusivity $a(\mathbf{x}) > 0$ and the coefficient $c(\mathbf{x}) \ge 0$.

- When $c \equiv 0$ and $\beta \equiv 0$, the problem (6.1) has infinitely many solutions.
  - If $u(\mathbf{x})$ is a solution, so is $u(\mathbf{x}) + C$, for $\forall\, C \in \mathbb{R}$.
  - Also we can see that the corresponding algebraic system is singular.
  - The singularity is not a big issue in numerical simulation; one may impose a Dirichlet condition at a grid point on the boundary.
- We may assume that (6.1) admits a unique solution.
- For simplicity, the 2nd-order **finite difference method (FDM)** will be used for discretization.

To explain the main feature of the central FDM, we may start with the problem (6.1) with the constant diffusivity, i.e., $a(\mathbf{x}) \equiv 1$.

6.1.1. Constant-coefficient problems

Consider the following simplified problem ($a \equiv 1$):

$$\begin{array}{rcll} -u_{xx} - u_{yy} + cu &=& f(x,y), & (x,y) \in \Omega, \\ u_\nu + \beta u &=& g(x,y), & (x,y) \in \Gamma, \end{array} \tag{6.2}$$

Furthermore, we may start with the 1D problem:

$$\begin{array}{lrcll} \text{(a)} & -u_{xx} + cu &=& f, & x \in (a_x, b_x), \\ \text{(b)} & -u_x + \beta u &=& g, & x = a_x, \\ \text{(c)} & u_x + \beta u &=& g, & x = b_x. \end{array} \tag{6.3}$$

FD Approximation

- Select n_x equally spaced grid points on the interval $[a_x, b_x]$:

$$x_i = a_x + ih_x, \quad i = 0, 1, \cdots, n_x, \quad h_x = \frac{b_x - a_x}{n_x}.$$

- Let $u_i = u(x_i)$ and recall the central second-order FD scheme for u_{xx} at x_i:

$$-u_{xx}(x_i) \approx \frac{-u_{i-1} + 2u_i - u_{i+1}}{h_x^2} + \frac{u_{xxxx}(x_i)}{12} h_x^2 + \cdots . \tag{6.4}$$

- Apply the FD scheme for (6.3.a) to have

$$-u_{i-1} + (2 + h_x^2 c)u_i - u_{i+1} = h_x^2 f_i. \tag{6.5}$$

- However, we will meet ghost grid values at the end points. For example, at the point $a_x = x_0$, the formula becomes

$$-u_{-1} + (2 + h_x^2 c)u_0 - u_1 = h_x^2 f_0. \tag{6.6}$$

  Here the value u_{-1} is not defined and we call it a **ghost grid value**.

- Now, let's replace the value by using the boundary condition (6.3.b). The central FD scheme for u_x at x_0 reads

$$u_x(x_0) \approx \frac{u_1 - u_{-1}}{2h_x}, \quad \text{Trunc.Err} = -\frac{u_{xxx}(x_0)}{6} h_x^2 + \cdots . \tag{6.7}$$

- Thus he equation (6.3.b) can be approximated (at x_0)

$$u_{-1} + 2h_x \beta u_0 - u_1 = 2h_x g_0. \tag{6.8}$$

- Hence it follows from (6.6) and (6.8) that

$$(2 + h_x^2 c + 2h_x\beta)u_0 - 2u_1 = h_x^2 f_0 + 2h_x g_0. \tag{6.9}$$

  The same can be considered for the algebraic equation at the point x_n.

The Algebraic System

The problem (6.3) is reduced to finding the solution $\mathbf{u}_1$ satisfying

$$A_1\mathbf{u}_1 = \mathbf{b}_1, \tag{6.10}$$

where

$$A_1 = \begin{bmatrix} 2+h_x^2c+2h_x\beta & -2 & & & \\ -1 & 2+h_x^2c & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2+h_x^2c & -1 \\ & & & -2 & 2+h_x^2c+2h_x\beta \end{bmatrix},$$

and

$$\mathbf{b}_1 = \begin{bmatrix} h_x^2 f_0 \\ h_x^2 f_1 \\ \vdots \\ h_x^2 f_{n_x-1} \\ h_x^2 f_{n_x} \end{bmatrix} + \begin{bmatrix} 2h_x g_0 \\ 0 \\ \vdots \\ 0 \\ 2h_x g_{n_x} \end{bmatrix}.$$

Such a technique of removing ghost grid values is called **outer bordering**. We can use it for the 2D problem (6.2) along the boundary grid points.

Symmetrization: The matrix A_1 is not symmetric! You can symmetrize it by dividing the first and the last rows of $[A_1|\mathbf{b}_1]$ by 2. For the 2D problem, you have to apply “division by 2" along each side of boundaries. (So, the algebraic equations corresponding to the corner points would be divided by a total factor of 4, for a symmetric algebraic system.)

6.1.2. General diffusion coefficients

Let the 1D problem read

$$\begin{array}{llcll} \text{(a)} & -(au_x)_x + cu & = & f, & x \in (a_x, b_x), \\ \text{(b)} & -au_x + \beta u & = & g, & x = a_x, \\ \text{(c)} & au_x + \beta u & = & g, & x = b_x. \end{array} \tag{6.11}$$

The central FD scheme for $(au_x)_x$ can be obtained as follows.

- The term (au_x) can be viewed as a function and approximated as

$$(au_x)_x(x_i) \approx \frac{(au_x)_{i+1/2} - (au_x)_{i-1/2}}{h_x} + \mathcal{O}(h_x^2), \tag{6.12}$$

  where $(au_x)_{i+1/2}$ denotes the value of (au_x) evaluated at $x_{i+1/2} := (x_i + x_{i+1})/2$.
- The terms $(au_x)_{i+1/2}$ and $(au_x)_{i-1/2}$ can be again approximated as

$$\begin{array}{lcl} (au_x)_{i+1/2} & \approx & a_{i+1/2}\dfrac{u_{i+1} - u_i}{h_x} - a_{i+1/2}\dfrac{u_{xxx}(x_{i+1/2})}{3!}\left(\dfrac{h_x}{2}\right)^2 + \cdots, \\ (au_x)_{i-1/2} & \approx & a_{i-1/2}\dfrac{u_i - u_{i-1}}{h_x} - a_{i-1/2}\dfrac{u_{xxx}(x_{i-1/2})}{3!}\left(\dfrac{h_x}{2}\right)^2 + \cdots. \end{array} \tag{6.13}$$

- Combine the above two equations to have

$$-(au_x)_x(x_i) \approx \frac{-a_{i-1/2}u_{i-1} + (a_{i-1/2} + a_{i+1/2})u_i - a_{i+1/2}u_{i+1}}{h_x^2}, \tag{6.14}$$

  of which the overall truncation error becomes $\mathcal{O}(h_x^2)$. See Exercise 6.1 on page 295.

Note:

- The y-directional approximation can be done in the same fashion.
- The reader should also notice that the quantities $a_{i+1/2}$ evaluated at mid-points are not available in general.
- We may replace it by the arithmetic/harmonic average of a_i and a_{i+1}:

$$a_{i+1/2} \approx \frac{a_i + a_{i+1}}{2} \quad \text{or} \quad \left[\frac{1}{2}\left(\frac{1}{a_i} + \frac{1}{a_{i+1}}\right)\right]^{-1}. \tag{6.15}$$

- The harmonic average is preferred; the resulting system holds the conservation property, particularly the **flux conservation**.

6.1.3. FD schemes for mixed derivatives

The linear elliptic equation in its general form is given as

$$-\nabla \cdot (A(\mathbf{x})\nabla u) + \mathbf{b} \cdot \nabla u + cu = f, \quad \mathbf{x} \in \Omega \subset \mathbb{R}^d, \tag{6.16}$$

where $1 \le d \le 3$ and

$$-\nabla \cdot (A(\mathbf{x})\nabla u) = -\sum_{i,j} \frac{\partial}{\partial x_i}\left(a_{ij}(\mathbf{x}) \frac{\partial u}{\partial x_j} \right).$$

Thus we must approximate the mixed derives whenever they appear.

As an example, we consider a second-order FD scheme for $(au_x)_y$ on a mesh of grid size $h_x \times h_y$:

$$\begin{aligned} (au_x)_y(\mathbf{x}_{pq}) &\approx \frac{au_x(\mathbf{x}_{p,q+1}) - au_x(\mathbf{x}_{p,q-1})}{2h_y} + \mathcal{O}(h_y^2) \\ &\approx \frac{a_{p,q+1}(u_{p+1,q+1} - u_{p-1,q+1}) - a_{p,q-1}(u_{p+1,q-1} - u_{p-1,q-1})}{4h_xh_y} \\ &\quad +\mathcal{O}(h_x^2) + \mathcal{O}(h_y^2). \end{aligned} \tag{6.17}$$

- There may involve difficulties in FD approximations when the diffusion coefficient A is a full tensor.
- Scalar coefficients can also become a full tensor when coordinates are changed.

6.1.4. L^∞-norm error estimates for FD schemes

Let Ω be a rectangular domain in 2D and $\Gamma = \partial\Omega$. Consider

$$\begin{aligned} -\Delta u &= f, \quad \mathbf{x} \in \Omega, \\ u &= g, \quad \mathbf{x} \in \Gamma, \end{aligned} \tag{6.18}$$

where $\mathbf{x} = (x, y) = (x_1, x_2)$ and

$$\Delta = \nabla \cdot \nabla = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} = \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2}.$$

Let Δ_h be the discrete five-point Laplacian:

$$\begin{aligned} \Delta_h u_{pq} &= (\delta_x^2 + \delta_y^2) u_{pq} \\ &:= \frac{u_{p-1,q} - 2u_{pq} + u_{p+1,q}}{h_x^2} + \frac{u_{p,q-1} - 2u_{pq} + u_{p,q+1}}{h_y^2}. \end{aligned} \tag{6.19}$$

Consistency: Let u_h be the FD solution of (6.18), i.e.,

$$\begin{aligned} -\Delta_h u_h &= f, \quad \mathbf{x} \in \Omega_h, \\ u_h &= g, \quad \mathbf{x} \in \Gamma_h, \end{aligned} \tag{6.20}$$

where Ω_h and Γ_h are the sets of grid points on Ω° and Γ, respectively. Note that the exact solution u of (6.18) satisfies

$$-\Delta_h u = f + \mathcal{O}(h^2 \partial^4 u), \quad \mathbf{x} \in \Omega_h. \tag{6.21}$$

Thus it follows from (6.20) and (6.21) that for some $C > 0$ independent of h,

$$\|\Delta_h (u - u_h)\|_{\infty,\Omega_h} \le C h^2 \|\partial^4 u\|_{\infty,\Omega_h}, \tag{6.22}$$

where $\|\cdot\|_{\infty,\Omega_h}$ denotes the maximum norm measured on the grid points Ω_h.

Convergence: We are more interested in an error estimate for $(u - u_h)$ rather than for $\Delta_h(u - u_h)$. We begin with the following lemma.

Lemma 6.1. *Let Ω is a rectangular domain and v_h be a discrete function defined on a grid Ω_h of Ω with $v_h = 0$ on the boundary Γ_h. Then*

$$\|v_h\|_{\infty,\Omega_h} \le C\|\Delta_h v_h\|_{\infty,\Omega_h}, \tag{6.23}$$

for some $C > 0$ independent on h.

Clearly, $(u - u_h)$ in (6.22) can be considered as a discrete function on the unit square with $u - u_h = 0$ on Γ_h. Therefore, with a aid of Lemma 6.1, one can conclude

Theorem 6.2. *Let u and u_h be the solutions of (6.18) and (6.20), respectively. Then*

$$\|u - u_h\|_{\infty,\Omega_h} \le Ch^2\|\partial^4 u\|_{\infty,\Omega_h}, \tag{6.24}$$

for some $C > 0$ independent on the grid size h.

Generalization:

The above theorem can be expanded for more general elliptic problems of the form

$$\begin{aligned} Lu := -\nabla\cdot(A(\mathbf{x})\nabla u) + \mathbf{b}(\mathbf{x})\cdot\nabla u = f, &\quad \mathbf{x}\in\Omega,\\ u = g, &\quad \mathbf{x}\in\Gamma, \end{aligned} \tag{6.25}$$

where $A(\mathbf{x}) = \text{diag}(a_{11}(\mathbf{x}), a_{22}(\mathbf{x}))$.

Let L_h be the five-point central discretization of L and u_h be the solution of

$$\begin{aligned} L_h u_h = f, &\quad \mathbf{x}\in\Omega_h,\\ u_h = g, &\quad \mathbf{x}\in\Gamma_h. \end{aligned} \tag{6.26}$$

Theorem 6.3. *Let u and u_h be the solutions of (6.25) and (6.26), respectively. Assume h is sufficiently small for the case $\mathrm{b}\neq 0$. Then*

$$\|u-u_h\|_{\infty,\Omega_h} \le Ch^2, \tag{6.27}$$

for some $C = C(\Omega, \partial^3 u, \partial^4 u) > 0$ independent on the grid size h.

6.1.5. The Algebraic System for FDM

Let $\Omega = [a_x, b_x] \times [a_y, b_y]$ and $\Gamma = \partial\Omega$. Consider (6.18):

$$\begin{aligned} -\Delta u &= f, \quad \mathbf{x} \in \Omega, \\ u &= g, \quad \mathbf{x} \in \Gamma. \end{aligned} \tag{6.28}$$

Define, for some positive integers n_x, n_y,

$$h_x = \frac{b_x - a_x}{n_x}, \quad h_y = \frac{b_y - a_y}{n_y}$$

and

$$\begin{aligned} x_p &= a_x + p\,h_x, \quad p = 0, 1, \cdots, n_x \\ y_q &= a_y + q\,h_y, \quad q = 0, 1, \cdots, n_y \end{aligned}$$

Let Δ_h be the discrete five-point Laplacian (6.19):

$$\begin{aligned} \Delta_h u_{pq} &= (\delta_x^2 + \delta_y^2) u_{pq} \\ &:= \frac{u_{p-1,q} - 2u_{pq} + u_{p+1,q}}{h_x^2} + \frac{u_{p,q-1} - 2u_{pq} + u_{p,q+1}}{h_y^2}. \end{aligned} \tag{6.29}$$

Then, when the grid points are ordered row-wise, the algebraic system for the FDM reads

$$A\mathbf{u} = \mathbf{b}, \tag{6.30}$$

where

$$A = \begin{bmatrix} B & -I/h_y^2 & & & 0 \\ -I/h_y^2 & B & -I/h_y^2 & & \\ & \ddots & \ddots & \ddots & \\ & & -I/h_y^2 & B & -I/h_y^2 \\ 0 & & & -I/h_y^2 & B \end{bmatrix} \tag{6.31}$$

with I being the identity matrix of dimension $n_x - 1$ and B being a matrix of order $n_x - 1$ given by

$$B = \begin{bmatrix} d & -1/h_x^2 & & & 0 \\ -1/h_x^2 & d & -1/h_x^2 & & \\ & \ddots & \ddots & \ddots & \\ & & -1/h_x^2 & d & -1/h_x^2 \\ 0 & & & -1/h_x^2 & d \end{bmatrix} \tag{6.32}$$

where $d = \dfrac{2}{h_x^2} + \dfrac{2}{h_y^2}$.
On the other hand,

$$\begin{aligned} b_{pq} &= f_{pq} + \frac{g_{p-1,q}}{h_x^2}\delta_{p-1,0} + \frac{g_{p+1,q}}{h_x^2}\delta_{p+1,n_x} \\ &\quad + \frac{g_{p,q-1}}{h_y^2}\delta_{q-1,0} + \frac{g_{p,q+1}}{h_y^2}\delta_{q+1,n_y} \end{aligned} \tag{6.33}$$

Here, the **global point index** for the row-wise ordering of the interior points, $i = 0, 1, 2, \cdots$, becomes

$$i = (q-1) * (n_x - 1) + p - 1 \tag{6.34}$$

Saving and managing the algebraic system

- For the FDM we just considered, the total number of interior nodal points is
$$(n_x - 1) * (n_y - 1)$$
Thus, you may try to open the matrix and other arrays based on this number.
- Saving nonzero entries only, the matrix A can be stored in an array of the form
$$A[M][5] \text{ or } A[n_y - 1][n_x - 1][5], \tag{6.35}$$
where $M = (n_x - 1) * (n_y - 1)$.
- However, it is often more convenient when the memory objects are opened incorporating all the nodal points (including those on boundaries). You may open the matrix as
$$A[n_y + 1][n_x + 1][5]. \tag{6.36}$$

The matrix A in (6.31) can be saved, in Python, as

```python
rx, ry = 1/hx**2, 1/hy**2
d = 2*(rx+ry)
for q in range(1,ny):
    for p in range(1,nx):
        A[q][p][0] = -ry
        A[q][p][1] = -rx
        A[q][p][2] =  d
        A[q][p][3] = -rx
        A[q][p][4] = -ry
```

Let the solution vector **u** be opened in `u[ny+1][nx+1]` and initialized along the boundaries. Then, the **Gauss-Seidel iteration** can be carried out as

Gauss-Seidel

```
import numpy as np; import copy
from numpy import abs,sqrt,pi,sin,cos

# the Jacobi matrix
T = copy.deepcopy(A) # np.ndarray((ny+1,nx+1,5),float)

for q in range(1,ny):
    for p in range(1,nx):
        for c in [0,1,3,4]:
            T[q][p][c] = -T[q][p][c]/T[q][p][2]

# A function for the Gauss-Seidel iteration
def Gauss_Seidel(T,u,itmax=1):
    ny,nx = leng(u)-1, len(u[0])-1
    for it in range(0,itmax):
        for q in range(1,ny):
            for p in range(1,nx):
                u[q][p] = T[q][p][0]*u[q-1][p] \
                        +T[q][p][1]*u[q][p-1] \
                        +T[q][p][3]*u[q][p+1] \
                        +T[q][p][4]*u[q+1][p]
```

Positiveness

Definition 6.4. *An $n \times n$ complex-valued matrix $A = [a_{ij}]$ is* **diagonally dominant** *if*

$$|a_{ii}| \geq \Lambda_i := \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, \tag{6.37}$$

for all $1 \leq i \leq n$. An $n \times n$ matrix A is irreducibly diagonally dominant if A is irreducible and diagonally dominant, with strict inequality holding in (6.37) for at least one i.

Theorem 6.5. *Let A be an $n \times n$ strictly or irreducibly diagonally dominant complex-valued matrix. Then, A is nonsingular. If all the diagonal entries of A are in addition positive real, then the real parts of all eigenvalues of A are positive.*

Corollary 6.6. *A Hermitian matrix satisfying the conditions in Theorem 6.5 is positive definite.*

Corollary 6.7. *The FD/FE matrices from diffusion equations (including the Poisson equation) are positive definite, when it is symmetric.*

6.2. Iterative Linear Solvers

6.2.1. Basic concepts for iterative solvers

We consider iterative methods in a more general mathematical setting. A general type of iterative process for solving the algebraic system

$$A\mathbf{x} = \mathbf{b}, \quad A \in \mathbb{R}^{n\times n} \tag{6.38}$$

can be described as follows.

- Split the matrix A as
$$A = M - N, \tag{6.39}$$
where M is an invertible matrix.
- Then, the system (6.38) can be expressed equivalently as
$$M\mathbf{x} = N\mathbf{x} + \mathbf{b}. \tag{6.40}$$
- Associated with the splitting is an iterative method
$$M\mathbf{x}^k = N\mathbf{x}^{k-1} + \mathbf{b} \quad \Longrightarrow \quad \mathbf{x}^k = M^{-1}N\mathbf{x}^{k-1} + M^{-1}\mathbf{b} \tag{6.41}$$
for a given initialization $\mathbf{x}^0$, $k \ge 1$.
- Since $N = M - A$, (6.41) can be rewritten as
$$\begin{array}{rcl} \mathbf{x}^k &=& (I - M^{-1}A)\mathbf{x}^{k-1} + M^{-1}\mathbf{b} \\ && \text{or} \\ \mathbf{x}^k &=& \mathbf{x}^{k-1} + M^{-1}(\mathbf{b} - A\mathbf{x}^{k-1}) \end{array} \tag{6.42}$$
Here, the matrix $I - M^{-1}A$ $(= M^{-1}N)$ is called the **iteration matrix**.

We shall say that the iterative method in (6.42) is **convergent** if it converges for any initial vector x^0. A sequence of vectors $\{\mathrm{x}^1, \mathrm{x}^2, \cdots\}$ will be computed from (6.42), and our objective is to choose M so that these two conditions are met:

1. The sequence $\{\mathrm{x}^k\}$ is easily computed. (Of course, M must be invertible.)
2. The sequence $\{\mathrm{x}^k\}$ converges rapidly to the solution.

Both conditions can be satisfied if ① M **is easy to invert** and ② M^{-1} **approximates** A^{-1} **well**.

Convergence: Recall (6.42):

$$\mathbf{x}^k = (I - M^{-1}A)\mathbf{x}^{k-1} + M^{-1}\mathbf{b} \quad (k \geq 1).$$

If the sequence $\{\mathbf{x}^k\}$ converges, say to a vector $\mathbf{x}$, then it follows from (6.42) that

$$\mathbf{x} = (I - M^{-1}A)\mathbf{x} + M^{-1}\mathbf{b} \tag{6.43}$$

Thus, by letting $\mathbf{e}^k = \mathbf{x} - \mathbf{x}^k$, we have

$$\mathbf{e}^k = (I - M^{-1}A)\mathbf{e}^{k-1} \quad (k \geq 1), \tag{6.44}$$

which implies

$$\begin{aligned} ||\mathbf{e}^k|| &\leq ||I - M^{-1}A||\,||\mathbf{e}^{k-1}|| \\ &\leq ||I - M^{-1}A||^2\,||\mathbf{e}^{k-2}|| \\ &\leq \cdots \leq ||I - M^{-1}A||^k\,||\mathbf{e}^0|| \end{aligned} \tag{6.45}$$

Thus, it can be concluded as in the following theorem.

$\boxed{\textbf{Theorem}}$ **6.8. (Sufficient condition for convergence)**.
If $||I - M^{-1}A|| = ||M^{-1}N|| < 1$ for some induced matrix norm, then the sequence produced by (6.42) converges to the solution of $A\mathrm{x} = \mathrm{b}$ for any initial vector x^0.

Note: Let $\delta = ||I - M^{-1}A||$.

- **(Choice of M)** When

$$M^{-1}A \approx I, \text{ or equivalently } M^{-1} \approx A^{-1},$$

  the quantity δ will become smaller and therefore the iteration converges faster.

- **(Stopping criterion)** If $\delta = ||I - M^{-1}A|| < 1$, then **it is safe to halt the iterative process when $||\mathbf{x}^k - \mathbf{x}^{k-1}||$ is small.** Indeed, since

$$\mathbf{e}^k = (I - M^{-1}A)\mathbf{e}^{k-1} = (I - M^{-1}A)(\mathbf{e}^k + \mathbf{x}^k - \mathbf{x}^{k-1}),$$

  we can obtain

$$||\mathbf{e}^k|| \le \delta(||\mathbf{e}^k|| + ||\mathbf{x}^k - \mathbf{x}^{k-1}||)$$

  which implies

$$||\mathbf{e}^k|| \le \frac{\delta}{1-\delta}\,||\mathbf{x}^k - \mathbf{x}^{k-1}||. \tag{6.46}$$

Theorem 6.9. *The iteration (6.42) converges if and only if*

$$\rho(I - M^{-1}A) = \rho(M^{-1}N) < 1. \tag{6.47}$$

Note: An iterative algorithm converges if and only if its iteration matrix is convergent.

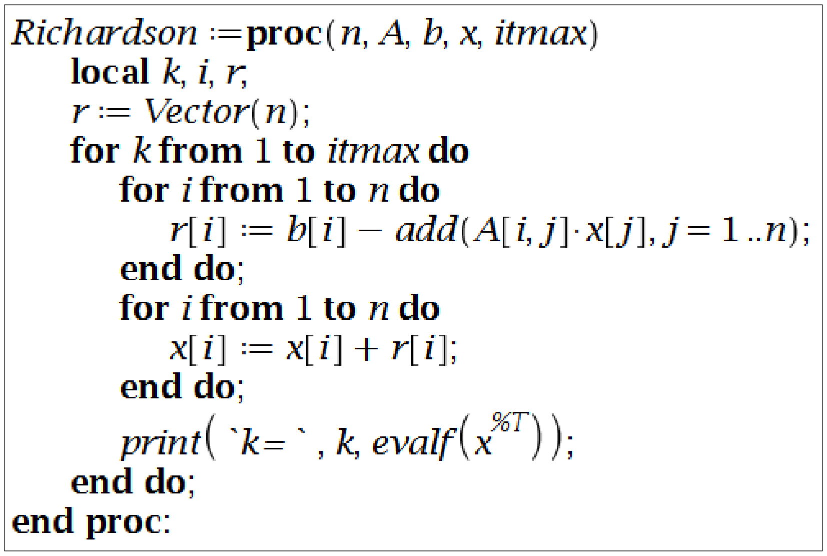
6.2.2. Richardson method: the simplest iteration

The iterative method is called the **Richardson method** if M is simply chosen to be the identity matrix:

$$M = I \;\text{ and }\; N = I - A.$$

In this case, the second equation in (6.42) reads

$$\mathbf{x}^k \;=\; \mathbf{x}^{k-1} + (\mathbf{b} - A\mathbf{x}^{k-1}). \tag{6.48}$$



```
Richardson :=proc(n, A, b, x, itmax)
    local k, i, r;
    r := Vector(n);
    for k from 1 to itmax do
        for i from 1 to n do
            r[i] := b[i] - add(A[i, j]·x[j], j = 1..n);
        end do;
        for i from 1 to n do
            x[i] := x[i] + r[i];
        end do;
        print( `k= ` , k, evalf(x^%T) );
    end do;
end proc:
```

Figure 6.1: A maple implementation for the Richardson method.

Results of Richardson

```
A := 1/6*Matrix([[6, 3, 2], [2, 6, 3], [3, 2, 6]]):
b := 1/6*Vector([11, 11, 11]):
x := Vector([0, 0, 0]):
Richardson(3, A, b, x, 10)
        k=, 1, [1.833333333, 1.833333333, 1.833333333]
        k=, 2, [0.3055555556, 0.3055555556, 0.3055555556]
        k=, 3, [1.578703704, 1.578703704, 1.578703704]
        k=, 4, [0.5177469136, 0.5177469136, 0.5177469136]
        k=, 5, [1.401877572, 1.401877572, 1.401877572]
        k=, 6, [0.6651020233, 0.6651020233, 0.6651020233]
        k=, 7, [1.279081647, 1.279081647, 1.279081647]
        k=, 8, [0.7674319606, 0.7674319606, 0.7674319606]
        k=, 9, [1.193806699, 1.193806699, 1.193806699]
       k=, 10, [0.8384944171, 0.8384944171, 0.8384944171]
```

Eigenvalues of the iteration matrix
```
with(LinearAlgebra):  Id := Matrix(3, shape = identity):
evalf(Eigenvalues(Id - A));
           [         -0.8333333333       ]
           [0.4166666667 - 0.1443375673 I]
           [0.4166666667 + 0.1443375673 I]
```

Note: Eigenvalues of A must be $\begin{bmatrix} 1.833333333 \\ 0.5833333333 + i\,0.1443375673 \\ 0.5833333333 - i\,0.1443375673 \end{bmatrix}$.
Thus all eigenvalues λ of A are in the open disk $\{z \in \mathbb{C} : |z-1| < 1\}$, which is a sufficient and necessary condition for the convergence of the Richardson method.

Generalization of the Richardson method

Consider
$$Ax = \mathbf{b}, \tag{6.49}$$
where some eigenvalues of A are **not** in $\{z \in \mathbb{C} : |z-1| < 1\}$.

- First, scale (6.49) with a constant η:
$$\eta A\mathbf{x} = \eta\mathbf{b}, \tag{6.50}$$
where all the eigenvalues of ηA are in the open disk.
- Then, apply the Richardson method to (6.50):
with $M = I$ and $N = I - \eta A$, the iteration reads
$$\begin{aligned} \mathbf{x}^k &= \mathbf{x}^{k-1} + (\eta\mathbf{b} - \eta A\mathbf{x}^{k-1}) \\ &= \mathbf{x}^{k-1} + \eta(\mathbf{b} - A\mathbf{x}^{k-1}). \end{aligned} \tag{6.51}$$
Thus the Richardson method converges by choosing an appropriate **scaling factor** η.

Self-study 6.10. Let $A \in \mathbb{R}^{n\times n}$ be a **definite matrix**. Prove that the **generalized Richardson method** is convergent for the solution of $Ax = b$.

Solution.

Regular Splitting and M-matrices

Definition 6.11. *For $n \times n$ real matrices, A, M, and N, $A = M - N$ is a* **regular splitting** *of A if M is nonsingular with $M^{-1} \geq 0$, and $N \geq 0$.*

Theorem 6.12. *If $A = M - N$ is a regular splitting of A and $A^{-1} \geq 0$, then*

$$\rho(M^{-1}N) = \frac{\rho(A^{-1}N)}{1 + \rho(A^{-1}N)} < 1. \tag{6.52}$$

Thus, the matrix $M^{-1}N$ is convergent and the associated iterative method of (6.42) converges for any initial value $\mathbf{x}^0$.

Definition 6.13. *An $n \times n$ real matrix $A = [a_{ij}]$ with $a_{ij} \leq 0$ for all $i \neq j$ is an* **M-matrix** *if A is nonsingular and $A^{-1} \geq 0$.*

Theorem 6.14. *Let $A = (a_{ij})$ be an $n \times n$ M-matrix. If M is any $n \times n$ matrix obtained by setting certain off-diagonal entries of A to zero, then $A = M - N$ is a regular splitting of A and $\rho(M^{-1}N) < 1$.*

Theorem 6.15. *Let A be an $n \times n$ real matrix with $A^{-1} > 0$, and $A = M_1 - N_1 = M_2 - N_2$ be two regular splittings of A. If $N_2 \geq N_1 \geq 0$, where neither $N_2 - N_1$ nor N_1 is null, then*

$$0 < \rho(M_1^{-1}N_1) < \rho(M_2^{-1}N_2) < 1. \tag{6.53}$$

Note: Results of the kind of Theorem 6.15 are called **comparison theorem** in the literature.

6.3. Relaxation Methods

Definition 6.16. A **matrix splitting** is an expression which represents a given matrix as a sum or difference of matrices: $A = M - N$.

6.3.1. Point relaxation methods

Relaxation methods: We first express $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ as the matrix sum:

$$A = D - E - F, \tag{6.54}$$

where

$$\begin{aligned} D &= \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix} & -E &= \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ a_{n1} & \cdots & a_{n,n-1} & 0 \end{bmatrix} \\ -F &= \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{n-1,n} \\ 0 & \cdots & 0 & 0 \end{bmatrix} \end{aligned}$$

Then, **relaxation methods** can be formulated by selecting M and N for the matrix splitting. Examples are:

Table 6.1: Common relaxation methods.

	M	N
Jacobi method	D	$E + F$
Gauss-Seidel method	$D - E$	F
SOR method	$\frac{1}{\omega}D - E$, $\omega \in (0, 2)$	$\frac{1-\omega}{\omega}D + F$

Here, **SOR** stands for **successive over relaxation**.

Jacobi Method

The **Jacobi method** is formulated with $M = D$ and $N = E + F$:

$$D\mathbf{x}^k = (E + F)\mathbf{x}^{k-1} + \mathbf{b}. \tag{6.55}$$

The i-th component of (6.55) reads

$$a_{ii}x_i^k = b_i + \sum_{j=1}^{i-1}(-a_{ij}x_j^{k-1}) + \sum_{j=i+1}^{n}(-a_{ij}x_j^{k-1}), \tag{6.56}$$

or, equivalently,

$$x_i^k = \frac{1}{a_{ii}}\Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k-1} - \sum_{j=i+1}^{n} a_{ij}x_j^{k-1}\Big) \tag{6.57}$$

Example 6.17. Let $A := \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$ and $\mathbf{b} := \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix}$. Use the Jacobi method to find $\mathbf{x}^k$, $k = 1, 2, 3$, beginning from $\mathbf{x}^0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$.

Solution.

Manual checking in Maple

```
x0 := Vector([1,1,1]):
x1 := Vector(3): x2 := Vector(3): x3 := Vector(3):

x1[1] := (-A[1, 2]*x0[2] - A[1, 3]*x0[3] + b[1])/A[1, 1];
x1[2] := (-A[2, 1]*x0[1] - A[2, 3]*x0[3] + b[2])/A[2, 2];
x1[3] := (-A[3, 1]*x0[1] - A[3, 2]*x0[2] + b[3])/A[3, 3];
x1^%T
                              [1, 1, 3]

x2[1] := (-A[1, 2]*x1[2] - A[1, 3]*x1[3] + b[1])/A[1, 1];
x2[2] := (-A[2, 1]*x1[1] - A[2, 3]*x1[3] + b[2])/A[2, 2];
x2[3] := (-A[3, 1]*x1[1] - A[3, 2]*x1[2] + b[3])/A[3, 3];
x2^%T
                              [1, 2, 3]

x3[1] := (-A[1, 2]*x2[2] - A[1, 3]*x2[3] + b[1])/A[1, 1];
x3[2] := (-A[2, 1]*x2[1] - A[2, 3]*x2[3] + b[2])/A[2, 2];
```

```
x3[3] := (-A[3, 1]*x2[1] - A[3, 2]*x2[2] + b[3])/A[3, 3];
x3^%T
                               [3      7]
                               [-, 2, -]
                               [2      2]
```

Note: The true solution is $[2,3,4]^T$.

Jacobi: Maple implementation

```
Jacobi := proc(n, A, b, x, tol, itmax)
    local i, j, k, id1, id2, xx;
    xx := Matrix(n, 2);
    for i from 1 to n do
        xx[i, 1] := x[i];
    end do;
    for k from 1 to itmax do
        id1 := modp(k + 1, 2) + 1;
        id2 := modp(k, 2) + 1;
        for i from 1 to n do
            xx[i, id2] := (b[i] − add(A[i, j]·xx[j, id1], j = 1 ..i − 1)
                                 - add(A[i, j]·xx[j, id1], j = i + 1 ..n)) / A[i, i];
        end do;
        print(`k= `, k, evalf(xx[1 ..n, id2]^%T));
    end do;
end proc:
```

Figure 6.2: Maple implementation for the Jacobi method.

```
Results of Jacobi
Jacobi(3, A, b, x0, tol, 10)
                      k=, 1, [1., 1., 3.]
                      k=, 2, [1., 2., 3.]
             k=, 3, [1.500000000, 2., 3.500000000]
         k=, 4, [1.500000000, 2.500000000, 3.500000000]
         k=, 5, [1.750000000, 2.500000000, 3.750000000]
         k=, 6, [1.750000000, 2.750000000, 3.750000000]
         k=, 7, [1.875000000, 2.750000000, 3.875000000]
         k=, 8, [1.875000000, 2.875000000, 3.875000000]
         k=, 9, [1.937500000, 2.875000000, 3.937500000]
        k=, 10, [1.937500000, 2.937500000, 3.937500000]
```

Jacobi: the ℓ^∞-error = 0.0625

Gauss-Seidel Method

The **Gauss-Seidel method** is formulated with $M = D - E$ and $N = F$:

$$(D - E)\mathbf{x}^k = F\mathbf{x}^{k-1} + \mathbf{b}. \tag{6.58}$$

Note that (6.58) can be equivalently written as

$$D\mathbf{x}^k = \mathbf{b} + E\mathbf{x}^k + F\mathbf{x}^{k-1}. \tag{6.59}$$

The i-th component of (6.59) reads

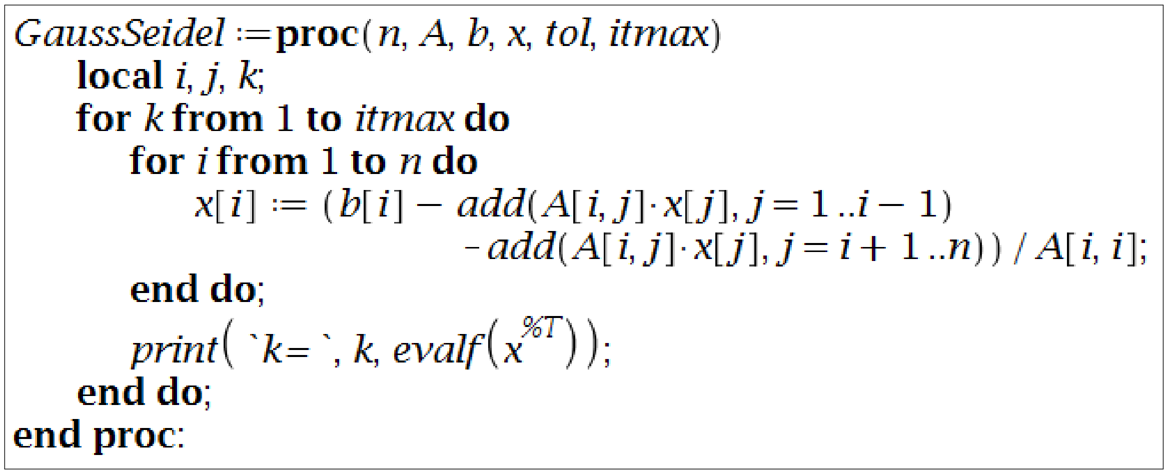
$$a_{ii}x_i^k \;=\; b_i + \sum_{j=1}^{i-1}(-a_{ij}x_j^k) + \sum_{j=i+1}^{n}(-a_{ij}x_j^{k-1}), \tag{6.60}$$

or, equivalently,

$$x_i^k \;=\; \frac{1}{a_{ii}}\Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^k - \sum_{j=i+1}^{n} a_{ij}x_j^{k-1}\Big) \tag{6.61}$$

The difference is that the SOR method utilizes updated values.

Gauss-Seidel (GS): Maple implementation



```
GaussSeidel := proc(n, A, b, x, tol, itmax)
    local i, j, k;
    for k from 1 to itmax do
        for i from 1 to n do
            x[i] := (b[i] - add(A[i, j]·x[j], j = 1 ..i - 1)
                        - add(A[i, j]·x[j], j = i + 1 ..n)) / A[i, i];
        end do;
        print( `k= `, k, evalf(x^%T));
    end do;
end proc:
```

Figure 6.3: Maple implementation for the Gauss-Seidel method.

Results of GaussSeidel

```
GaussSeidel(3, A, b, x0, tol, 10)
                      k=, 1, [1., 1., 3.]
                  k=, 2, [1., 2., 3.500000000]
         k=, 3, [1.500000000, 2.500000000, 3.750000000]
         k=, 4, [1.750000000, 2.750000000, 3.875000000]
         k=, 5, [1.875000000, 2.875000000, 3.937500000]
         k=, 6, [1.937500000, 2.937500000, 3.968750000]
         k=, 7, [1.968750000, 2.968750000, 3.984375000]
         k=, 8, [1.984375000, 2.984375000, 3.992187500]
         k=, 9, [1.992187500, 2.992187500, 3.996093750]
        k=, 10, [1.996093750, 2.996093750, 3.998046875]
```

Gauss-Seidel: the ℓ^∞-error ≈ 0.0039 = 3.9E-3.

Note: By comparison with the result of the Jacobi method, we may conclude that Gauss-Seidel method is **twice faster** than the Jacobi method.

Successive Over Relaxation (SOR) Method

The **successive over relaxation** (**SOR**) is formulated with $M = \frac{1}{\omega}D - E$ and $N = \frac{1-\omega}{\omega}D + F$:

$$(D - \omega E)\mathbf{x}^k = \big((1-\omega)D + \omega F\big)\mathbf{x}^{k-1} + \omega\mathbf{b}. \tag{6.62}$$

Note that (6.62) can be equivalently written as

$$D\mathbf{x}^k = (1-\omega)D\mathbf{x}^{k-1} + \omega(\mathbf{b} + E\mathbf{x}^k + F\mathbf{x}^{k-1}). \tag{6.63}$$

The i-th component of (6.63) reads

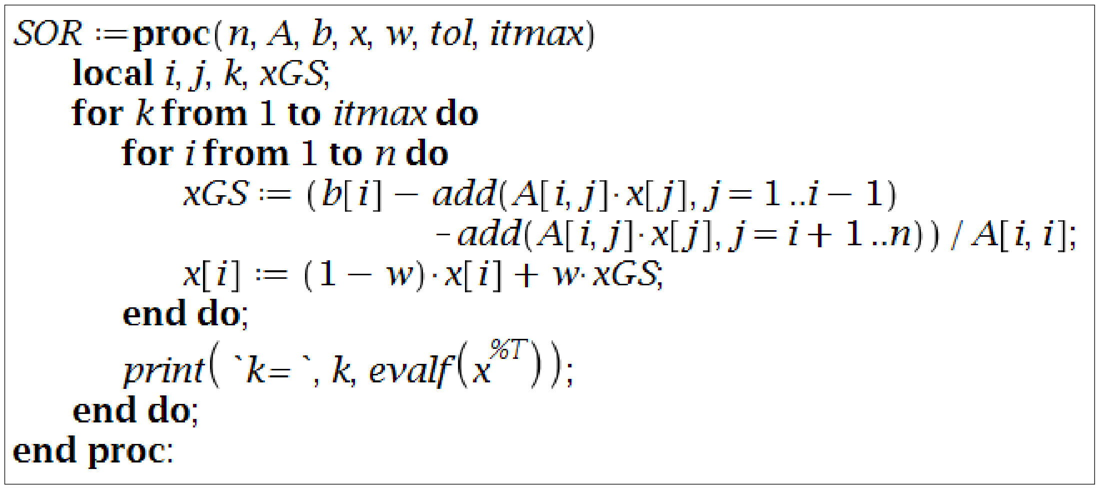
$$a_{ii}x_i^k \;=\; (1-\omega)a_{ii}x_i^{k-1} + \omega\Big(b_i + \sum_{j=1}^{i-1}(-a_{ij}x_j^k) + \sum_{j=i+1}^{n}(-a_{ij}x_j^{k-1})\Big), \tag{6.64}$$

or, equivalently,

$$\begin{aligned} x_{GS,i}^k &= \frac{1}{a_{ii}}\Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^k - \sum_{j=i+1}^{n} a_{ij}x_j^{k-1}\Big) \\ x_i^k &= (1-\omega)x_i^{k-1} + \omega x_{GS,i}^k \end{aligned} \tag{6.65}$$

Note that when $\omega = 1$, the SOR becomes the Gauss-Seidel method.

SOR: Maple implementation



```
SOR :=proc(n, A, b, x, w, tol, itmax)
    local i, j, k, xGS;
    for k from 1 to itmax do
        for i from 1 to n do
            xGS := (b[i] - add(A[i, j]·x[j], j = 1 ..i - 1)
                         - add(A[i, j]·x[j], j = i + 1 ..n)) / A[i, i];
            x[i] := (1 - w)·x[i] + w·xGS;
        end do;
        print(`k= `, k, evalf(x^%T));
    end do;
end proc:
```

Figure 6.4: Maple implementation for the SOR method.

Results of SOR
```
SOR(3, A, b, x0, 1.2, tol, 10)
             k=, 1, [1.0, 1.000000000, 3.400000000]
         k=, 2, [1.000000000, 2.440000000, 3.784000000]
         k=, 3, [1.864000000, 2.900800000, 3.983680000]
         k=, 4, [1.967680000, 2.990656000, 3.997657600]
         k=, 5, [2.000857600, 3.000977920, 4.001055232]
         k=, 6, [2.000415232, 3.000686694, 4.000200970]
         k=, 7, [2.000328970, 3.000180625, 4.000068180]
         k=, 8, [2.000042580, 3.000030331, 4.000004563]
         k=, 9, [2.000009683, 3.000002482, 4.000000576]
        k=, 10, [1.999999552, 2.999999581, 3.999999633]
```

SOR: the ℓ^∞-error $\approx$ 0.00000045 = 4.5E-7.

Note: The SOR with $\omega = 1.2$:

- It is much faster than the Jacobi and GS methods.
- **Question**: How can we find the optimal parameter $\widehat{\omega}$?
  We will see it soon.

Convergence Theory

Theorem 6.18. *For and $\mathbf{x}^0 \in \mathbb{R}^n$, the sequence defined by*

$$\mathbf{x}^k = T\,\mathbf{x}^{k-1} + \mathbf{c} \tag{6.66}$$

converges to the unique solution of $\mathbf{x} = T\,\mathbf{x} + \mathbf{c}$ if and only if $\rho(T) < 1$. In this case, the iterates satisfy

$$||\mathbf{x} - \mathbf{x}^k|| \le ||T||^k\, ||\mathbf{x} - \mathbf{x}^0||. \tag{6.67}$$

For example:

Relaxation method	T (Iteration matrix)
Jacobi method	$T_J \;=\; D^{-1}(E+F)$
Gauss-Seidel method	$T_{GS} \;=\; (D-E)^{-1}F$
SOR method	$T_{SOR} \;=\; (D-\omega E)^{-1}\big[(1-\omega)D + \omega F\big]$

Theorem 6.19. (Stein and Rosenberg, 1948) [21]. *One and only one of the following mutually exclusive relations is valid:*

1. $\rho(T_J) = \rho(T_{GS}) = 0$
2. $0 < \rho(T_{GS}) < \rho(T_J) < 1$
3. $\rho(T_J) = \rho(T_{GS}) = 1$
4. $1 < \rho(T_J) < \rho(T_{GS})$

Theorem 6.20. (Ostrowski, 1954) [16]. *Let A be symmetric. Then,*

$$\rho(T_{SOR}) < 1 \iff A \text{ is is positive definite and } 0 < \omega < 2. \tag{6.68}$$

Parameter 6.21. (Optimal ω for the SOR). For algebraic systems of good properties, it is theoretically known that the convergence of the SOR can be optimized when

$$\widehat{\omega} = \frac{2}{1 + \sqrt{1 - \rho(T_J)^2}}. \tag{6.69}$$

However, in many cases you can find a better ω for a given algebraic system.

Note: Let $0 < \rho(T_J) < 1$. Then the theoretically optimal SOR parameter

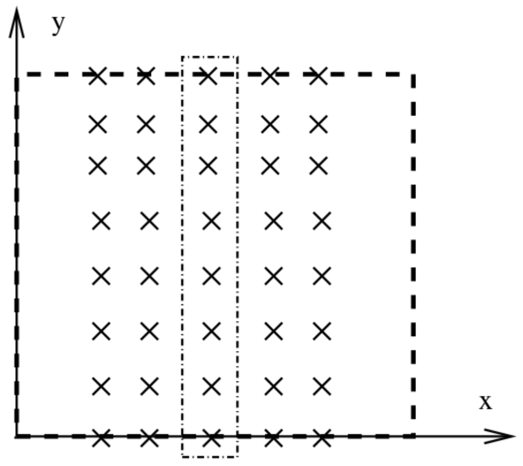
$$\begin{aligned} &1 < \widehat{\omega} < 2, \\ &\widehat{\omega} \approx 1 + \frac{1}{4}\rho(T_J)^2 + \frac{1}{8}\rho(T_J)^4. \end{aligned} \tag{6.70}$$

Remark 6.22.

- When $\omega > 1$, the blending of the SOR, the second equation in (6.65), is an extrapolation. It is how the algorithm is named.
- On the other hand, when $\omega < 1$, the algorithm is also called the **successive under relaxation (SUR)**.

6.3.2. Line relaxation methods

- The standard Jacobi, Gauss-Seidel, and SOR schemes are called **point relaxation methods**.
- We can compute a whole line of new values using a direct method, e.g., Gauss elimination.
- this leads to **line relaxation methods**.



Algebraic interpretation: As in §6.1.5, consider

$$\begin{aligned} -\Delta u &= f, \quad \mathbf{x} \in \Omega, \\ u &= g, \quad \mathbf{x} \in \Gamma, \end{aligned} \tag{6.71}$$

where Ω is a rectangular domain in $\mathbb{R}^2$, and its discrete five-point Laplacian

$$\begin{aligned} \Delta_h u_{pq} &= (\delta_x^2 + \delta_y^2) u_{pq} \\ &:= \frac{u_{p-1,q} - 2u_{pq} + u_{p+1,q}}{h_x^2} + \frac{u_{p,q-1} - 2u_{pq} + u_{p,q+1}}{h_y^2}. \end{aligned} \tag{6.72}$$

Then, for the *column-wise point ordering*, the algebraic system for the FDM reads

$$A\mathbf{u} = \mathbf{b}, \tag{6.73}$$

where

$$A = \begin{bmatrix} C & -I/h_x^2 & & & 0 \\ -I/h_x^2 & C & -I/h_x^2 & & \\ & \ddots & \ddots & \ddots & \\ & & -I/h_x^2 & C & -I/h_x^2 \\ 0 & & & -I/h_x^2 & C \end{bmatrix} \tag{6.74}$$

with I being the identity matrix of dimension $n_y - 1$ and C being a matrix of order $n_x - 1$ given by

$$C = \begin{bmatrix} d & -1/h_y^2 & & & 0 \\ -1/h_y^2 & d & -1/h_y^2 & & \\ & \ddots & \ddots & \ddots & \\ & & -1/h_y^2 & d & -1/h_y^2 \\ 0 & & & -1/h_y^2 & d \end{bmatrix} \tag{6.75}$$

where $d = \dfrac{2}{h_x^2} + \dfrac{2}{h_y^2}$.

- A line relaxation method can be viewed as a (standard) relaxation method which deals with the matrix C like a single entry of a tridiagonal matrix.
- Once a point relaxation method converges, its line method converges **twice faster asymptotically**.
- Line methods can employ the line solver in alternating directions of (x, y).

Convergence comparison: For (6.71) on p.274, we choose

$$\Omega = (0,1)^2, \quad n = n_x = n_y.$$

The following table includes the spectral radii of iteration matrices $\rho(T)$ and the required iteration counts k for the convergence to satisfy the tolerance $\|e^k\|/\|e^0\| < 10^{-6}$.

Table 6.2: Convergence comparison

	Point Jacobi		Line Jacobi		Point GS		Line GS	
n	$\rho(T)$	k	$\rho(T)$	k	$\rho(T)$	k	$\rho(T)$	k
5	0.8090	66	0.6793	36	0.6545	33	0.4614	18
10	0.9511	276	0.9067	142	0.9045	138	0.8221	71
20	0.9877	1116	0.9757	562	0.9755	558	0.9519	281
40	0.9969	4475	0.9939	2241	0.9938	2238	0.9877	1121

Final remarks for relaxation methods

- GS methods converge *asymptotically* twice faster than Jacobi methods, in either point or line iterations. SOR is yet faster and the line SOR is again twice faster.
- Relaxation methods sweep over either points or groups of points. For a faster convergence, you may let them visit the points in an order followed by the opposite order.
- For line methods, the tridiagonal matrix can be stored in a 3-column array, instead of a square big-fat array.

6.4. Krylov Subspace Methods

Definition 6.23. A matrix $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ is said to be **positive definite** if

$$\mathbf{x}^T A\mathbf{x} = \sum_{i,j=1}^{n} x_i a_{ij} x_j > 0, \quad \forall\, \mathbf{x} \in \mathbb{R}^n, \quad \mathbf{x} \neq 0. \tag{6.76}$$

For solving a linear system

$$A\mathbf{x} = \mathbf{b}, \tag{6.77}$$

where A is **symmetric positive definite**, Krylov subspace methods update the iterates as follows.

Given an initial guess $\mathbf{x}_0 \in \mathbb{R}^n$, find successive approximations $\mathbf{x}_k \in \mathbb{R}^n$ of the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \quad k = 0, 1, \cdots, \tag{6.78}$$

where $\mathbf{p}_k$ is the *search direction* and $\alpha_k > 0$ is the *step length*.

- Different methods differ in the choice of the search direction and the step length.
- In this section, we consider the **gradient descent method** the **conjugate gradient (CG) method**, and **preconditioned CG method**.
- For other Krylov subspace methods, see e.g. [3, 13].

Remark 6.24. The algebraic system (6.77) admits a unique solution $\mathbf{x} \in \mathbb{R}^n$, which is equivalently characterized by

$$\mathbf{x} = \arg\min_{\boldsymbol{\eta}\in\mathbb{R}^n} f(\boldsymbol{\eta}), \quad f(\boldsymbol{\eta}) = \frac{1}{2}\boldsymbol{\eta}\cdot A\boldsymbol{\eta} - \mathbf{b}\cdot\boldsymbol{\eta}. \tag{6.79}$$

6.4.1. Gradient descent (GD) method

The **gradient descent method** is also known as the **steepest descent method** or the **Richardson's method**.

Derivation of the GD method

- We denote the **gradient** and **Hessian** of f by f' and f'', respectively:

$$f'(\boldsymbol{\eta}) = A\boldsymbol{\eta} - \mathbf{b}, \quad f''(\boldsymbol{\eta}) = A. \tag{6.80}$$

- Given $\mathbf{x}_{k+1}$ as in (6.78), we have by Taylor's formula

$$\begin{aligned} f(\mathbf{x}_{k+1}) &= f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \\ &= f(\mathbf{x}_k) + \alpha_k f'(\mathbf{x}_k) \cdot \mathbf{p}_k + \frac{\alpha_k^2}{2} \mathbf{p}_k \cdot f''(\boldsymbol{\xi}) \mathbf{p}_k, \end{aligned} \tag{6.81}$$

  for some $\boldsymbol{\xi}$.
- Since $f''(\boldsymbol{\eta})\ (= A)$ is bounded,

$$f(\mathbf{x}_{k+1}) = f(\mathbf{x}_k) + \alpha_k f'(\mathbf{x}_k) \cdot \mathbf{p}_k + \mathcal{O}(\alpha_k^2), \quad \text{as } \alpha_k \to 0.$$

- **The goal** is to find $\mathbf{p}_k$ and α_k such that

$$f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k), \tag{6.82}$$

  which can be achieved if

$$f'(\mathbf{x}_k) \cdot \mathbf{p}_k < 0 \tag{6.83}$$

  and α_k is sufficiently small.
- **Choice**: When $f'(\mathbf{x}_k) \neq 0$, (6.83) holds, if we choose:

$$\mathbf{p}_k = -f'(\mathbf{x}_k) = \mathbf{b} - A\mathbf{x}_k =: \mathbf{r}_k \tag{6.84}$$

  That is, the search direction is the **negative gradient**, the steepest descent direction.

Optimal step length: We may determine α_k such that

$$f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) = \min_{\alpha} f(\mathbf{x}_k + \alpha \mathbf{p}_k), \tag{6.85}$$

in which case α_k is said to be **optimal**.

If α_k is optimal, then

$$\begin{aligned} 0 \;&=\; \frac{\mathbf{d}}{\mathbf{d}\alpha} f(\mathbf{x}_k + \alpha \mathbf{p}_k)\Big|_{\alpha=\alpha_k} = f'(\mathbf{x}_k + \alpha_k \mathbf{p}_k)\cdot \mathbf{p}_k \\ &=\; (A(\mathbf{x}_k + \alpha_k \mathbf{p}_k) - \mathbf{b})\cdot \mathbf{p}_k \\ &=\; (A\mathbf{x}_k - \mathbf{b})\cdot \mathbf{p}_k + \alpha_k \mathbf{p}_k \cdot A\mathbf{p}_k. \end{aligned} \tag{6.86}$$

So,

$$\alpha_k = \frac{\mathbf{r}_k \cdot \mathbf{p}_k}{\mathbf{p}_k \cdot A\mathbf{p}_k}. \tag{6.87}$$

Theorem 6.25. (Convergence of the GD method): *The GD method converges, satisfying*

$$\|\,\mathbf{x} - \mathbf{x}_k\,\|_2 \le \left(1 - \frac{1}{\kappa(A)}\right)^k \|\,\mathbf{x} - \mathbf{x}_0\,\|_2. \tag{6.88}$$

Thus, the number of iterations required to reduce the error by a factor of ε is in the order of the condition number of A:

$$k \;\ge\; \kappa(A)\log\frac{1}{\varepsilon}. \tag{6.89}$$

6.4.2. Conjugate gradient (CG) method

In this method the search directions $\mathbf{p}_k$ are **conjugate**, i.e.,

$$\mathbf{p}_i \cdot A\mathbf{p}_j = 0, \quad i \ne j, \tag{6.90}$$

and the step length α_k is chosen to be optimal.

Algorithm 6.26. (CG Algorithm, V.1)

```
Select $\mathbf{x}_0$, $\varepsilon$;
$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\mathbf{p}_0 = \mathbf{r}_0$;
Do $k = 0, 1, \cdots$
    $\alpha_k = \mathbf{r}_k \cdot \mathbf{p}_k / \mathbf{p}_k \cdot A\mathbf{p}_k$;            (CG1)
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$;                 (CG2)
    $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k$;                (CG3)        (6.91)
    if $\| \mathbf{r}_{k+1} \|_2 < \varepsilon \| \mathbf{r}_0 \|_2$, stop;
    $\beta_k = -\mathbf{r}_{k+1} \cdot A\mathbf{p}_k / \mathbf{p}_k \cdot A\mathbf{p}_k$;   (CG4)
    $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$;              (CG5)
End Do
```

Remark 6.27. (CG Algorithm, V.1)

- In practice, $\mathbf{q}_k = A\mathbf{p}_k$ is computed only once, and saved.
- $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$, by definition. So,

$$\begin{aligned} \mathbf{r}_{k+1} &= \mathbf{b} - A\mathbf{x}_{k+1} = \mathbf{b} - A(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \\ &= \mathbf{b} - A\mathbf{x}_k - \alpha_k A\mathbf{p}_k = \mathbf{r}_k - \alpha_k A\mathbf{p}_k. \qquad \text{(CG3)} \end{aligned}$$

- α_k in (CG1) is optimal as shown in (6.87). Also it satisfies $\mathbf{r}_{k+1} \cdot \mathbf{p}_k = 0$. You may easily verify it using $\mathbf{r}_{k+1}$ in (CG3).
- β_k in (CG4) is determined such that $\mathbf{p}_{k+1} \cdot A\mathbf{p}_k = 0$. Verify it using $\mathbf{p}_{k+1}$ in (CG5).

Theorem 6.28. *For* $m = 0, 1, \cdots$,

$$\begin{aligned} \text{span}\{\mathbf{p}_0, \cdots, \mathbf{p}_m\} &= \text{span}\{\mathbf{r}_0, \cdots, \mathbf{r}_m\} \\ &= \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, \cdots, A^m \mathbf{r}_0\}. \end{aligned} \tag{6.92}$$

Theorem 6.29. *The search directions and the residuals satisfy the orthogonality,*

$$\mathbf{p}_i \cdot A\mathbf{p}_j = 0; \quad \mathbf{r}_i \cdot \mathbf{r}_j = 0, \quad i \neq j. \tag{6.93}$$

Theorem 6.30. *For some $m \le n$, we have $A\mathbf{x}_m = \mathbf{b}$ and*

$$\|\,\mathbf{x}-\mathbf{x}_k\,\|_A \le 2\left(\frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1}\right)^k \|\,\mathbf{x}-\mathbf{x}_0\,\|_A. \tag{6.94}$$

So the required iteration number to reduce the error by a factor of ε is

$$k \ge \frac{1}{2}\sqrt{\kappa(A)}\,\log\frac{2}{\varepsilon}. \tag{6.95}$$

Proofs of the above theorems can be found in e.g. [12].

Simplification of the CG method: Using the properties and identities involved in the method, one can derive a more popular form of the CG method.

Algorithm 6.31. (CG Algorithm, V.2)

```
Select $\mathbf{x}_0,\ \varepsilon$;
$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0,\ \mathbf{p}_0 = \mathbf{r}_0$;
Compute $\rho_0 = \mathbf{r}_0 \cdot \mathbf{r}_0$;
Do $k = 0, 1, \cdots$
    $\alpha_k = \rho_k / \mathbf{p}_k \cdot A\mathbf{p}_k$;
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$;
    $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k$;                    (6.96)
    if $\|\,\mathbf{r}_{k+1}\,\|_2 < \varepsilon \,\|\,\mathbf{r}_0\,\|_2$, stop;
    $\rho_{k+1} = \mathbf{r}_{k+1} \cdot \mathbf{r}_{k+1}$;
    $\beta_k = \rho_{k+1}/\rho_k$;
    $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$;
End Do
```

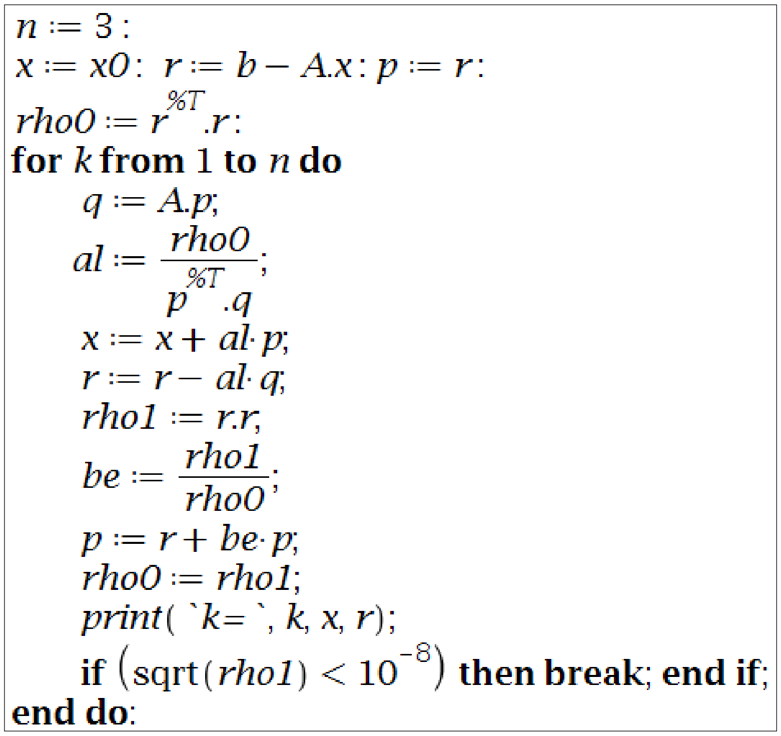
Note:

$$\begin{aligned}
\mathbf{r}_k \cdot \mathbf{p}_k &= \mathbf{r}_k \cdot (\mathbf{r}_k + \beta_{k-1}\mathbf{p}_{k-1}) = \mathbf{r}_k \cdot \mathbf{r}_k, \\
\beta_k &= -\mathbf{r}_{k+1} \cdot A\mathbf{p}_k / \mathbf{p}_k \cdot A\mathbf{p}_k = -\mathbf{r}_{k+1} \cdot A\mathbf{p}_k \,\frac{\alpha_k}{\rho_k} \\
&= \mathbf{r}_{k+1} \cdot (\mathbf{r}_{k+1} - \mathbf{r}_k)/\rho_k = \rho_{k+1}/\rho_k.
\end{aligned} \tag{6.97}$$

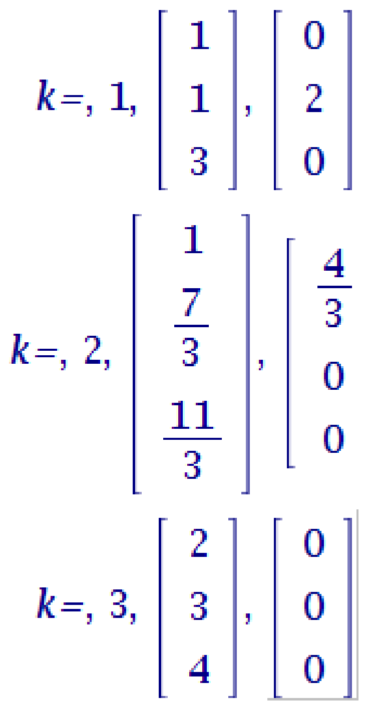
Example 6.32. (Revisit to Example 6.17)

Let $A := \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$ and $\mathbf{b} := \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix}$. Use the CG method to find $\mathbf{x}^k$, $k = 1, 2, 3$, beginning from $\mathbf{x}^0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$.

Solution.



```
n := 3:
x := x0: r := b - A.x: p := r:
rho0 := r^%T.r:
for k from 1 to n do
      q := A.p;
      al := rho0/(p^%T.q);
      x := x + al·p;
      r := r - al·q;
      rho1 := r.r;
      be := rho1/rho0;
      p := r + be·p;
      rho0 := rho1;
      print(`k=`, k, x, r);
      if (sqrt(rho1) < 10^(-8)) then break; end if;
end do:
```



$$k=,\ 1,\ \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix},\ \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix}$$

$$k=,\ 2,\ \begin{bmatrix} 1 \\ \frac{7}{3} \\ \frac{11}{3} \end{bmatrix},\ \begin{bmatrix} \frac{4}{3} \\ 0 \\ 0 \end{bmatrix}$$

$$k=,\ 3,\ \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix},\ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Remark 6.33.

- For the example, the CG **converged completely** in three iterations.
- The CG method was originally developed **as a direct solver**.

6.4.3. Preconditioned CG method

- The condition number of A, $\kappa(A)$, is the critical factor for the convergence of the CG method.
- If we can find a matrix M such that
$$M \approx A$$
and it is easy to invert, we may try to apply the CG algorithm to the following system
$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}. \tag{6.98}$$
- Since
$$\kappa(M^{-1}A) \ll \kappa(A) \tag{6.99}$$
(hopefully, $\kappa(M^{-1}A) \approx 1$), the CG algorithm will converge much faster.

In practice, we do not have to multiply M^{-1} to the original algebraic system and the algorithm can be implemented as

Algorithm 6.34. (Preconditioned CG)

```
Select $\mathbf{x}_0,\ \varepsilon$;
$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0,\ M\mathbf{z}_0 = \mathbf{r}_0$;
$\mathbf{p}_0 = \mathbf{z}_0$, compute $\rho_0 = \mathbf{z}_0^*\mathbf{r}_0$;
Do $k = 0, 1, \cdots$
    $\alpha_k = \rho_k / \mathbf{p}_k^* A\mathbf{p}_k$;
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$;
    $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k$;                (6.100)
    if $\| \mathbf{r}_{k+1} \|_2 < \varepsilon \| \mathbf{r}_0 \|_2$, stop;
    $M\mathbf{z}_{k+1} = \mathbf{r}_{k+1}$;
    $\rho_{k+1} = \mathbf{z}_{k+1}^*\mathbf{r}_{k+1}$;
    $\beta_k = \rho_{k+1}/\rho_k$;
    $\mathbf{p}_{k+1} = \mathbf{z}_{k+1} + \beta_k\mathbf{p}_k$;
End Do
```

Here the superscript * indicates the transpose complex-conjugate; it is the transpose for real-valued systems.

6.5. Applications of Richardson Extrapolation for PDEs

Problem 6.35. The **Richardson extrapolation** is a numerical procedure that produces **a numerical solution of a higher-order accuracy**, when two numerical solutions are available from a mesh and its refined one. By applying extrapolation recursively, the Richardson extrapolation becomes a sequence acceleration method that improves the rate of convergence of a sequence.

- **The Issue**: The Richardson extrapolation can compute the solution of a higher-order accuracy **only on the course mesh**.
- Various **multiple coarse grid (MCG)** updating strategies have been considered in the literature [5].
- An MCG updating method can be expensive computationally.
- The extrapolated solutions often show a desired order of accuracy, when they compared with each other. **However, their error can be larger than that of the numerical solution obtained using the original high-order scheme.**

Quesiton. Can we get higher-order solutions on finer meshes, without using a multiple coarse grid updating method?

Example 6.36. Consider

$$\begin{gathered} -u''(x) = \pi^2 \sin(\pi x), \quad x \in (0,1) \\ u(0) = 0, \quad u(1) = 2, \end{gathered} \tag{6.101}$$

for which the exact solution is $u(x) = \sin(\pi x) + 2x$.

(a) Partition $[0, 1]$ into n equal subintervals. That is,

$$h = \frac{1}{n}; \quad x_i = i \cdot h, \;\; i = 0, 1, \cdots, n.$$

Let $\mathbf{u}_h$ be such that $\mathbf{u}_h[i] = u(x_i)$.

(b) Approximate $-u''(x_i)$ by using the formula: For each $i = 1, 2, \cdots, n-1$,

$$-u''(x_i) = \frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} + K_2 h^2 + K_4 h^4 + \cdots = f_i. \quad \textbf{(6.102)}$$

where

$$K_j = 2 \cdot \frac{u^{(j+2)}(x_i)}{(j+2)!}, \quad j = 2, 4, \cdots . \quad \textbf{(6.103)}$$

(c) Assemble the above for an algebraic system to solve:

$$A\mathbf{v}_h = \mathbf{b} \quad \textbf{(6.104)}$$

(d) Find $\mathbf{v}_h$, $\mathbf{v}_{h/2}$, and $\mathbf{v}_{h/4}$.

(e) Apply the Richardson Extrapolation and measure the errors.

Remark 6.37. In Example 6.36:

- Solutions of a fourth-order accuracy can be obtained on the meshes of grid sizes h and $h/2$.
- Solutions of a sixth-order accuracy can be computed **only on the coarsest mesh** of grid size h.

Remark 6.38. Equation (6.102) can be rewritten as

$$u_i = \frac{u_{i-1} + u_{i+1}}{2} + \frac{h^2}{2}(f_i - K_2 h^2) - \frac{1}{2} K_4 h^6 - \cdots , \tag{6.105}$$

where

$$K_2 = \frac{u^{(4)}(x_i)}{12} \quad \text{and} \quad K_4 = \frac{u^{(6)}(x_i)}{360}. \tag{6.106}$$

Since $u^{(4)} = -f''$, for example, K_2 is approximated as

$$K_2 = \frac{u^{(4)}(x_i)}{12} = -\frac{1}{12} f''(x_i) = -\frac{1}{12} \frac{f_{i-1} - 2f_i + f_{i+1}}{h^2} + \mathcal{O}(h^2), \tag{6.107}$$

and therefore

$$f_i - K_2 h^2 = \frac{1}{12}(f_{i-1} + 10 f_i + f_{i+1}) + \mathcal{O}(h^4), \tag{6.108}$$

from which we have

$$u_i = \frac{u_{i-1} + u_{i+1}}{2} + \frac{h^2}{24}(f_{i-1} + 10 f_i + f_{i+1}) + \mathcal{O}(h^6). \tag{6.109}$$

$\boxed{\textbf{Algorithm}}$ **6.39.** In order to get a sixth-order solution on the mid mesh, we first should have *meaningful* fourth-order solutions on the mid and fine meshes.

1. Let $\mathbf{w}_h = \frac{1}{3}(4\mathbf{v}_{h/2} - \mathbf{v}_h)$, Richardson extrapolation on the course mesh.
2. Let $\widetilde{\mathbf{w}}_{h/2}$ be its expansion on the mid mesh: $\widetilde{\mathbf{w}}_{h/2}[\texttt{0 : 2 : end}] = \mathbf{w}_h$.
3. Then, you should determine values $\widetilde{\mathbf{w}}_{h/2}[\texttt{1 : 2 : end}]$ to complete the expansion. Using (6.109), if $i \in [\texttt{1 : 2 : 2} * \texttt{n} - \texttt{1}]$, we have

$$\widetilde{\mathbf{w}}_{h/2}(i) = \frac{\widetilde{\mathbf{w}}_{h/2}(i-1) + \widetilde{\mathbf{w}}_{h/2}(i+1)}{2} + \frac{(h/2)^2}{24}(f_{i-1} + 10f_i + f_{i+1}), \quad \textbf{(6.110)}$$

   where $f_i = f(i \cdot (h/2))$.
4. Repeat the above, starting with $\mathbf{w}_{h/2}$, to get its expansion $\widetilde{\mathbf{w}}_{h/4}$.
5. Then **a sixth-order solution on the mid mesh** is

$$\frac{1}{15}\big(16 * \widetilde{\mathbf{w}}_{h/4}[\texttt{0 : 2 : end}] - \widetilde{\mathbf{w}}_{h/2}\big). \quad \textbf{(6.111)}$$

   Check the error.

How to find the accuracy order of a numerical scheme:

- For given h, you can measure the error: that is,

$$||\mathbf{u}_h - \mathbf{v}_h||_\infty = E_h. \tag{6.112}$$

- If the accuracy order is α, we may write

$$E_h = \mathcal{O}(h^\alpha) = C \cdot h^\alpha, \tag{6.113}$$

for some constant $C > 0$.

- If you compute the numerical solution with $h/2$, then you can measure the error to be $E_{h/2}$, which can be written as

$$E_{h/2} = \mathcal{O}\big((h/2)^\alpha\big) = C \cdot (h/2)^\alpha, \tag{6.114}$$

- Dividing (6.113) by (6.114) reads

$$\frac{h^\alpha}{(h/2)^\alpha} = \frac{E_h}{E_{h/2}} \quad\Longrightarrow\quad 2^\alpha = \frac{E_h}{E_{h/2}}.$$

- Now, applying the natural log to the equation results in

$$\boxed{\alpha = \frac{\ln(E_h/E_{h/2})}{\ln 2}} \tag{6.115}$$

6.6. A Nonoscillatory High-Order Time-Stepping Procedure

Note: In this section, we consider a **nonoscillatory second-order time-stepping** procedure, for which details can be found in [14].

6.6.1. Heat transfer

Model-Problem 6.40. For simplicity, consider a diffusion equation defined in the 1D space.

$$\begin{array}{ll} \dfrac{\partial u}{\partial t} - u_{xx} = f, & (x,t) \in (0,1) \times [0,T], \\ u(0,t) = u(1,t) = 0, & t \in [0,T], \\ u(x,0) = u_0(x), & x \in (0,1). \end{array} \tag{6.116}$$

Discretization

- Let the domain be partitioned into n_x subintervals:

$$x_i = i \cdot \Delta x, \ \ i = 0, 1, \cdots, n_x, \quad \Delta x = \frac{1}{n_x}.$$

- Let $\mathcal{A}$ be the second-order approximation of $-\partial_{xx}$:

$$\mathcal{A}u = -u_{xx} + \mathcal{O}(\Delta x^2). \tag{6.117}$$

  Then the **semi-discrete problem** of (6.116) reads

$$\frac{\partial u}{\partial t} + \mathcal{A}u = f. \tag{6.118}$$

- Define $\Delta t = T/n_t$, for some positive integer n_t, $t^n = n\Delta t$, and $u^n = u(\cdot, t^n)$.
- Assume u^n is known.
- **To find u^{n+1}**: the θ**-method** is formulated as

$$\frac{u^{n+1} - u^n}{\Delta t} + \mathcal{A}[\theta u^{n+1} + (1-\theta)u^n] = f^{n+\theta}, \quad \theta \in [0,1], \tag{6.119}$$

where $f^{n+\theta}$ is either $f(x, t^{n+\theta})$ or $\theta f^{n+1} + (1-\theta) f^n$. A simple algebraic rearrangement of (6.119) gives

$$(I + \theta\Delta t\mathcal{A})u^{n+1} \;=\; [I - (1-\theta)\Delta t\mathcal{A}]u^n + \Delta t f^{n+\theta}. \tag{6.120}$$

Note: Popular choices of $\theta \in [0,1]$ are 0, 1, and 1/2.

- **Forward Euler method**: $\theta = 0$.

$$u^{n+1} \;=\; (I - \Delta t\mathcal{A})u^n + \Delta t f^n, \tag{6.121}$$

which is **stable** only for $\mu = \dfrac{\Delta t}{\Delta x^2} \le \dfrac{1}{2}$.
- **Crank-Nicolson method**: $\theta = 1/2$.

$$\Big(I + \frac{\Delta t}{2}\mathcal{A}\Big)u^{n+1} \;=\; \Big(I - \frac{\Delta t}{2}\mathcal{A}\Big)u^n + \Delta t f^{n+1/2}, \tag{6.122}$$

which is **unconditionally stable** and of a **second-order accuracy** in the temporal approximation.
- **Backward Euler method**: $\theta = 1$.

$$(I + \Delta t\mathcal{A})u^{n+1} \;=\; u^n + \Delta t f^{n+1}, \tag{6.123}$$

which is **unconditionally stable**.

Remark 6.41. Although the Crank-Nicolson (CN) method is **unconditionally stable** and of **second-order accuracy** in both spatial and temporal directions, it may introduce **spurious oscillations** into the numerical solution for nonsmooth data.

6.6.2. The variable-θ method

Strategy 6.42. Variable-θ method. In each time level,

- Default is the CN method ($\theta = 1/2$).
- The method detects points of potential oscillations, to implicitly resolve the solution there ($\theta = 1$).

How to detect points of potential oscillations

Definition 6.43. The **wobble set** is a collection of the grid points where the solution has high fluctuations so that the implicit method ($\theta = 1$) should be applied for the numerical solution not to develop oscillations.

Observation 6.44. Numerical oscillations of the CN method occur when **its explicit half step produces spurious oscillations**.

- Such non-physical oscillations may happen particularly when the time step size Δt is larger than the stability limit of the explicit scheme.
- Thus the wobble set may be formed to include points where the explicit half step of the CN method introduces undesired local extrema.
- It follows from (6.122) that the explicit half step of the CN method reads
$$u^{n,*} := \Big(I - \frac{\Delta t}{2}\mathcal{A}\Big)u^n, \tag{6.124}$$
for which we set $f \equiv 0$.

Algorithm 6.45. Determining the wobble set.

- Define an index function for local extrema (`idxt`) as

$$\texttt{idxt}(a,b,c) = \begin{cases} 0, & \text{if } \min(a,c) < b < \max(a,c), \\ 1, & \text{if } b = \max(a,c), \\ -1, & \text{if } b = \min(a,c), \\ 2, & \text{if } \max(a,c) < b, \\ -2, & \text{if } b < \min(a,c). \end{cases} \quad (6.125)$$

- Let P, Q, and R be point indices and define

$$\texttt{iswb}(P,Q,R,n) = \begin{cases} 1, & \text{if } \texttt{idxt}(u_P^{n,*}, u_Q^{n,*}, u_R^{n,*}) \neq 0 \text{ and} \\ & \left|\texttt{idxt}(u_P^{n,*}, u_Q^{n,*}, u_R^{n,*}) + \texttt{idxt}(u_P^{n}, u_Q^{n}, u_R^{n})\right| < 4, \\ 0, & \text{otherwise.} \end{cases} \quad (6.126)$$

- Then, the **wobble set** (for the computation of u^{n+1}) is defined as

$$\mathcal{W}^n = \left\{x_i \mid \texttt{iswb}(i-1,\, i,\, i+1,\, n) = 1\right\}. \quad (6.127)$$

Remark 6.46. The function `iswb` selects candidates for the wobble set from local extrema satisfying $\texttt{idxt}(u_P^{n,*}, u_Q^{n,*}, u_R^{n,*}) \neq 0$; however, the condition

$$\left|\texttt{idxt}(u_P^{n,*}, u_Q^{n,*}, u_R^{n,*}) + \texttt{idxt}(u_P^{n}, u_Q^{n}, u_R^{n})\right| < 4$$

excludes cases where a *strict* extremum in u^n becomes a *strict* extremum in the same sense in $u^{n,*}$. Thus, the wobble set (6.127) is the set of interior grid points x_i where $u_i^{n,*}$ becomes a local extremum while u_i^n is either a non-extreme or an extreme in the opposite sense.

Algorithm 6.47. The variable-θ method.

- Having the wobble set, the parameter θ for the computation of u^{n+1} can be assigned pointwisely

$$\theta_i^{n+1} := \theta(x_i, t^{n+1}) = \begin{cases} 1, & \text{if } x_i \in \mathcal{W}^n, \\ 1/2, & \text{otherwise.} \end{cases} \tag{6.128}$$

- Thus, the **variable-θ method** for (6.119) can be formulated as

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + \mathcal{A}[\theta_i^{n+1} u_i^{n+1} + (1-\theta_i^{n+1}) u_i^n] = f_i^{n+1/2}. \tag{6.129}$$

Example 6.48. When $f \equiv 0$ and

$$u_0(x) = \begin{cases} 0 & \text{if } 0 < x < \frac{1}{4}, \\ 1 & \text{if } \frac{1}{4} \le x < \frac{3}{4}, \\ 0 & \text{if } \frac{3}{4} \le x < 1, \end{cases}$$

the exact solution of (6.116) is given by

$$u(x,t) = \sum_{k=1}^{\infty} \frac{4}{k\pi} \sin\left(\frac{k\pi}{2}\right) \sin\left(\frac{k\pi}{4}\right) \sin(k\pi x)\, e^{-k^2\pi^2 t}, \tag{6.130}$$

for $(x,t) \in (0,1) \times [0,T]$.

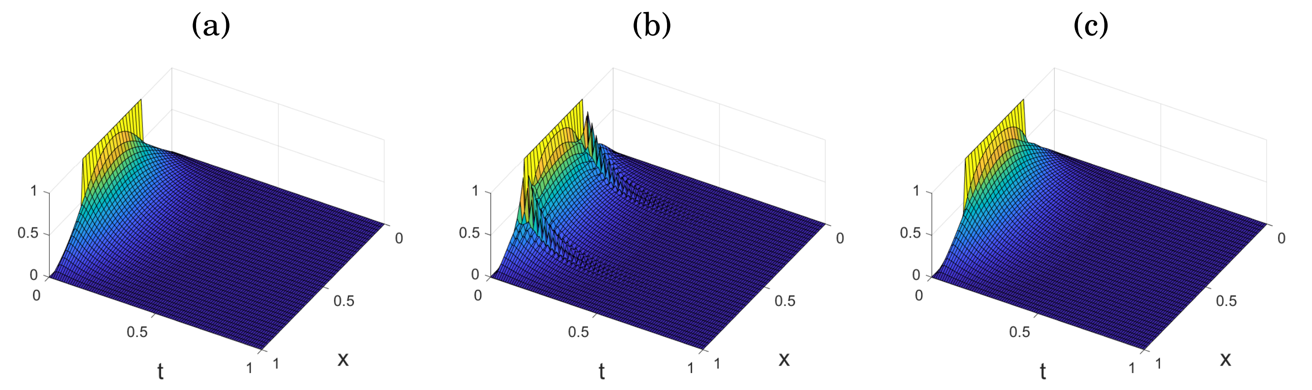


Figure 6.5: Propagation of the exact and numerical solutions for (6.116): (a) the exact solution and the numerical solution of (b) the CN method and (c) the variable-θ method (6.129) for $0 \le t \le T = 1.0$, when $\Delta t = 0.01$ and $\Delta x = 0.025$.

Accuracy Comparison

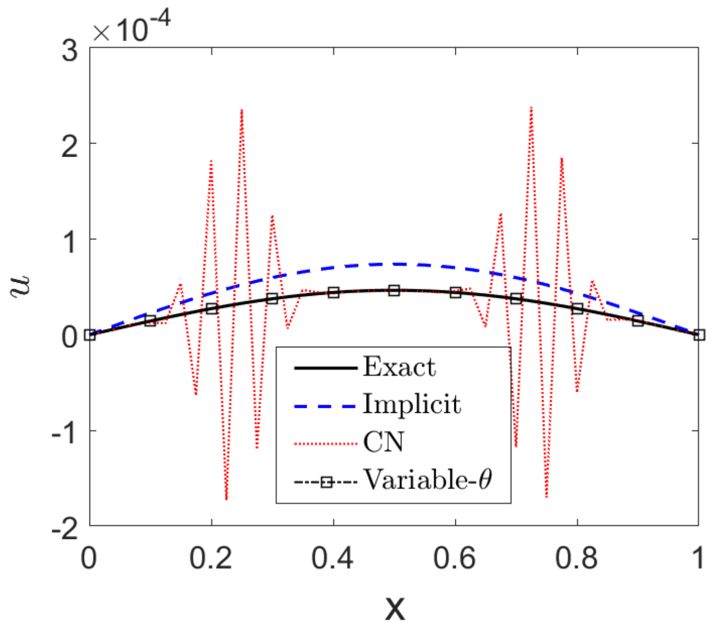


Figure 6.6: The numerical solutions at $T = 1.0$, compared with the exact solution.

Remark 6.49. The variable-θ method

- Its numerical solutions are **nonoscillatory**.
- It shows **second-order accuracy**, in practice.
  - Often, it is more accurate than the CN method.
- It is easily applicable for 2D and 3D problems [14, 15].
  - computational cost: only 3-5% increase.

Exercises for Chapter 6

6.1. Verify that the overall truncation error for the FD scheme (6.14) is second-order in h_x. *Hint*: Define

$$K(x) = a(x)\frac{u_{xxx}(x)}{3!}\left(\frac{h_x}{2}\right)^2 + \cdots,$$

for the truncation errors appeared in (6.13). Then the truncation error for the approximation of $(au_x)_{i+1/2} - (au_x)_{i-1/2}$ becomes $K(x_{i+1/2}) - K(x_{i-1/2}) = h_x K'(x_i) + \cdots$.

6.2. When the **boundary-value problem**

$$\begin{cases} -u_{xx} = -2, & 0 < x < 4 \\ u_x(0) = 0, & u(4) = 16 \end{cases} \tag{6.131}$$

is discretized by the second-order finite difference method with $h = 1$, the algebraic system reads $A\mathrm{x} = \mathbf{b}$, where

$$A = \begin{bmatrix} 2 & -2 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -2 \\ -2 \\ -2 \\ 14 \end{bmatrix} \tag{6.132}$$

and the exact solution is $\mathbf{x} = [0,\ 1,\ 4,\ 9]^T$.

(a) Is A irreducibly diagonally dominant?

(b) Perform 10 iterations of the Jacobi and Gauss-Seidel methods, starting from $\mathbf{x}_0 = [0,\ 0,\ 0,\ 0]^T$.

(c) Try to find the best ω with which the SOR method converges fastest during the first 10 iterations.

(d) Find the spectral radii of the iteration matrices of the Jacobi, the Gauss-Seidel, and the SOR with the best ω.

6.3. Implement a code for Example 6.36 and Algorithm 6.39, with $h = 1/10$, $1/20$, and $1/40$, to get a sixth-order solution on the mid mesh. If you use Matlab, you may start with

get_A_b.m
```
function [A,b] = get_A_b(interval,n,f,u)

A=zeros(n+1,n+1);
b=zeros(n+1,1);
h=(interval(2)-interval(1))/n;
%%---------
for i=2:n
   A(i,i-1) = -1;
   A(i,i) = 2;
   A(i,i+1) = -1;
   b(i) = h^2*f( interval(1)+(i-1)*h );
```

```
end

A(1,1)=1; A(n+1,n+1)=1;
b(1)=u(interval(1)); b(n+1)=u(interval(2));
```

Part of Richardson_extrapolation.m

```
u = @(x) sin(pi*x)+2*x;
f = @(x) pi^2*sin(pi*x);

interval=[0,1]; n0 = 10;
U = cell(3,1); V  = cell(3,1);
W = cell(2,2); WT = cell(3,1);

%%-- U & V
for i=1:3
    n = n0*2^(i-1); X =linspace(0,1,n+1)';
    U{i} = u(X);
    [A,b] = get_A_b(interval,n,f,u);
    V{i}= A\b;
end

%%-- Richardson
for i=1:2
    W{i,1} = (1/3)*(4*V{i+1}(1:2:end)-V{i});
end
W{1,2} = (1/15)*(16*W{2,1}(1:2:end)-W{1,1});

%%-- Expansion: WT{i}, i=2,3, from W{i-1,1}
```

CHAPTER 7

Matrix Analysis in Data Mining

Contents of Chapter 7

7.1. Introduction to Data Mining

Why Mine Data?

Commercial Viewpoint

- Lots of data is being collected and warehoused.
  - Web data, e-commerce
  - Purchases at department/grocery stores
  - Bank/Credit Card transactions
- Computers have become cheaper and more powerful.
- Competitive pressure is strong.
  - Provide better, customized services for an edge (e.g. in Customer Relationship Management)

Scientific Viewpoint

- Data collected and stored at enormous speeds (GB/hour)
  - Remote sensors on a satellite
  - Telescopes scanning the skies
  - Microarrays generating gene expression data
  - Scientific simulations generating terabytes of data
- Traditional techniques infeasible for raw data
- Data mining may help scientists
  - in classifying and segmenting data
  - in **Hypothesis Formation**

Mining Large Data Sets - Motivation

- There is often information "hidden" in the data that is not readily evident.
- Human analysts may take weeks to discover useful information.
- Much of the data is never analyzed at all.
  - **Data gap** becomes larger and larger.

What is Data Mining?

- **Data mining** is a process to turn raw data into useful information/patterns.
  - Non-trivial extraction of implicit, previously unknown and potentially useful information from data.
  - Exploration & analysis, by automatic or semi-automatic means, of large quantities of data in order to discover meaningful patterns.
  - **Patterns must be: valid, novel, understandable, and potentially useful**.

Note: Data mining is also called **Knowledge Discovery in Data** (KDD).

Origins of Data Mining

- Draws ideas from machine learning/AI, pattern recognition, statistics, and database systems.
- Traditional Techniques may be unsuitable, due to
  - Enormity of data
  - High dimensionality of data
  - Variety: Heterogeneous, distributed nature of data

Data Mining Methods

- **Prediction Methods**
  - Use some variables to predict unknown or future values of other variables.
- **Description Methods**
  - Find human-interpretable patterns that describe the data.

Data Mining Tasks

- **Classification** [Predictive]
  - Given a collection of records (training set), find a **model** for class attribute as a function of the values of other attributes.
- **Regression** [Predictive]
  - Predict a value of a given continuous valued variable based on the values of other variables, assuming a linear or nonlinear model of dependency.
- **Clustering** [Descriptive]
  - Given a set of data points and a similarity measure among them, find clusters such that data points in one cluster are more similar to one another than points in other clusters.
- **Association Rule Discovery** [Descriptive]
  - Given a set of records each of which contain some number of items from a given collection, produce dependency rules which will predict occurrence of an item based on occurrences of other items.
- **Sequential Pattern Discovery** [Descriptive]
  - Given is a set of objects, with each object associated with its own timeline of events, find rules that predict strong sequential dependencies among different events.
- **Deviation/Anomaly Detection** [Predictive]
  - Detect significant deviations from normal behavior.

Challenges in Data Mining

- Scalability
- Dimensionality
- Complex and Heterogeneous Data
  - Spatial and temporal data
  - Point and interval data
  - Categorical data
  - Graph data
  - semi/un-structured Data
- Data Quality
- Data Ownership and Distribution
- Privacy Preservation
- Streaming Data

Related Fields

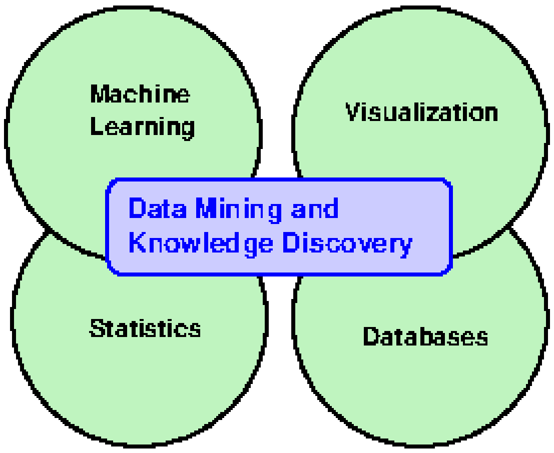


Figure 7.1: Related fields.

7.2. Vectors and Matrices in Data Mining

Note: Often the data are numerical, and the data points can be thought of as belonging to a high-dimensional vector space. Ensembles of data points can then be organized as matrices. In such cases it is natural to use concepts and techniques from linear algebra. Here, we present *Numerical Linear Algebra in Data Mining*, following and modifying (Eldén, 2006) [8].

7.2.1. Examples

Example 7.1. Term-document matrices are used in information retrieval. Consider the following set of five documents. Key words, referred to as **terms**, are marked in boldface.

Document 1:	The **Google matrix** P is a model of the **Internet**.
Document 2:	P_{ij} is nonzero if there is a **link** from **web page** j to i.
Document 3:	The **Google matrix** is used to **rank** all **web pages**.
Document 4:	The **ranking** is done by solving a **matrix eigenvalue** problem.
Document 5:	**England** dropped out of the top 10 in the **FIFA ranking**.

- Counting the frequency of terms in each document we get the following result.

Term	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5
eigenvalue	0	0	0	1	0
England	0	0	0	0	1
FIFA	0	0	0	0	1
Google	1	0	1	0	0
Internet	1	0	0	0	0
link	0	1	0	0	0
matrix	1	0	1	1	0
page	0	1	1	0	0
rank	0	0	1	1	1
web	0	1	1	0	0

The set of terms is called the **dictionary**.

- Each document is represented by a vector in $\mathbb{R}^{10}$, and we can organize the data as a **term-document matrix**,

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{10\times 5}. \tag{7.1}$$

- Assume that we want to find all documents that are relevant with respect to the query "ranking of web pages". This is represented by a query vector, constructed in an analogous way as the term-document matrix, using the same dictionary.

$$\mathbf{q} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \in \mathbb{R}^{10}. \tag{7.2}$$

  Thus the query itself is considered as a document.

- Now, the information retrieval task can be formulated as a mathematical problem: *find the columns of A that are close to the vector* $\mathbf{q}$.

  - To solve this problem we use some distance measure in $\mathbb{R}^{10}$.

Remark 7.2. Information Retrieval, with term-document matrix $A \in \mathbb{R}^{m\times n}$.

- It is common that m **is large**, of the order 10^6, say.
- **The matrix A is sparse**, because most of the documents only contain a small fraction of the terms in the dictionary.
- In some methods for information retrieval, linear algebra techniques, such as **singular value decomposition (SVD)**, are used for **data compression** and **retrieval enhancement**.

Note: **The very idea of data mining is to extract useful information from large and often unstructured datasets**. Therefore

- **the methods must be efficient** and often specially designed for large problems.

Example 7.3. Google Pagerank algorithm. The task of extracting information from all the web pages available on the Internet, is performed by **search engines**.

- The core of the **Google search engine** is a matrix computation, probably the largest that is performed routinely.
- The **Google matrix** P is assumed to be of dimension of **the order billions (2005)**, and it is used as a model of (all) the web pages on the Internet.
- In the **Google Pagerank algorithm**, the problem of assigning ranks to all the web pages is formulated as a **matrix eigenvalue problem**.

The Google Pagerank algorithm

- Let all web pages be ordered from 1 to n, and let i be a particular web page.
- Then O_i will denote the set of pages that i is linked to, the **outlinks**. The number of outlinks is denoted $N_i = |O_i|$.
- The set of inlinks, denoted I_i, are the pages that have an outlink to i.
- Now define Q to be a square matrix of dimension n, and let

$$Q_{ij} = \begin{cases} 1/N_j, & \text{if there is a link from } j \text{ to } i, \\ 0, & \text{otherwise.} \end{cases} \tag{7.3}$$

  - This definition means that row i has nonzero elements in those positions that correspond to inlinks of i.
  - Similarly, column j has nonzero elements equal to $1/N_j$ in those positions that correspond to the outlinks of j.
  - Thus, the sum of each column is either 0 or 1.

- The following **link graph** illustrates a set of web pages with outlinks and inlinks.

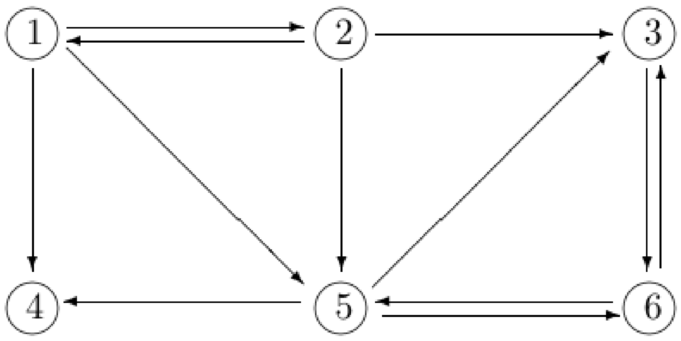


Figure 7.2: A link graph, for six web pages.

The corresponding **link matrix** becomes

$$Q = \begin{bmatrix} 0 & 1/3 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 & 1/3 & 1/2 \\ 1/3 & 0 & 0 & 0 & 1/3 & 0 \\ 1/3 & 1/3 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 1 & 0 & 1/3 & 0 \end{bmatrix} \tag{7.4}$$

- Define a **pagerank vector** r, which holds the ranks of all pages.
- Then, the vector r can be found as the eigenvector corresponding to the eigenvalue $\lambda = 1$ of Q:
$$Q\mathbf{r} = \lambda\mathbf{r}. \tag{7.5}$$
We discuss numerical aspects of the Pagerank computation in Section 7.4.

7.2.2. Data compression: Low rank approximation

Note: **Rank Reduction**.

- One way of measuring the information contents in a data matrix is to compute its rank.
- Obviously, linearly dependent column or row vectors are redundant.
- Therefore, one natural procedure for extracting information from a data matrix is **to systematically determine a sequence of linearly independent vectors**, and deflate the matrix by subtracting rank one matrices, one at a time.
- It turns out that this **rank reduction procedure** is closely related to **matrix factorization**, **data compression**, **dimensionality reduction**, and **feature selection/extraction**.
- The key link between the concepts is the **Wedderburn rank reduction theorem**.

Theorem 7.4. *(Wedderburn, 1934) [27]. Suppose $A \in \mathbb{R}^{m\times n}$, $f \in \mathbb{R}^n$, and $g \in \mathbb{R}^m$. Then*
$$\textit{rank}\Big(A - \frac{Afg^TA}{\omega}\Big) = \textit{rank}(A) - 1 \iff \omega = g^TAf \neq 0. \tag{7.6}$$

Algorithm 7.5. Wedderburn rank-reduction process.
Based on Wedderburn rank reduction theorem, a stepwise rank reduction procedure can be defined.

- Let $A^{(0)} = A$.
- Define a sequence of matrices $\{A^{(i)}\}$:

$$A^{(i+1)} = A^{(i)} - \frac{A^{(i)} f^{(i)} {g^{(i)}}^T A^{(i)}}{\omega_i}, \tag{7.7}$$

  where $f^{(i)} \in \mathbb{R}^n$ and $g^{(i)} \in \mathbb{R}^m$ such that

$$\omega_i = {g^{(i)}}^T A^{(i)} f^{(i)} \neq 0. \tag{7.8}$$

- The sequence defined in (7.7) terminates when $r = \text{rank}(A^{(i+1)})$, since each time the rank of the matrix decreases by one. The matrices $A^{(i)}$ are called **Wedderburn matrices**.

Remark 7.6. The Wedderburn rank-reduction process gives a matrix decomposition called the **rank-reduction decomposition**.

$$A = \widehat{F} \Omega^{-1} \widehat{G}, \tag{7.9}$$

where

$$\begin{array}{rcl} \widehat{F} & = & (\mathbf{f}_1, \cdots, \mathbf{f}_r) \in \mathbb{R}^{m\times r}, \quad \mathbf{f}_i = A^{(i)} f^{(i)}, \\ \Omega & = & \text{diag}(\omega_1, \cdots, \omega_r) \in \mathbb{R}^{r\times r}, \\ \widehat{G} & = & (\mathbf{g}_1, \cdots, \mathbf{g}_r) \in \mathbb{R}^{n\times r}, \quad \mathbf{g}_i = {A^{(i)}}^T g^{(i)}. \end{array} \tag{7.10}$$

Theorem 7.4 can be generalized to the case where the reduction of rank is larger than one, as shown in the next theorem.

Theorem **7.7.** *(Guttman, 1957) [11]. Suppose $A \in \mathbb{R}^{m\times n}$, $F \in \mathbb{R}^{n\times k}$, and $G \in \mathbb{R}^{m\times k}$. Then*

$$\begin{aligned} \textit{rank}(A - AFR^{-1}G^TA) &= \textit{rank}(A) - \textit{rank}(AFR^{-1}G^TA) \\ &\iff R = G^TAF \text{ is nonsingular}. \end{aligned} \tag{7.11}$$

Note: There are many choices of F and G that satisfy the condition (7.11).

- Therefore, various rank-reduction decompositions are possible.
- It is known that several standard matrix factorizations in numerical linear algebra are instances of the Wedderburn formula:
  - Gram-Schmidt orthogonalization,
  - singular value decomposition,
  - QR and Cholesky decomposition, and
  - the Lanczos procedure.

Relation between the truncated SVD and the Wedderburn rank reduction process

- Recall the truncated SVD (4.14), page 135.
- In the rank reduction formula (7.11), define the error matrix E as

$$E = A - AFR^{-1}G^TA = A - AF(G^TAF)^{-1}G^TA, \tag{7.12}$$

  where $F \in \mathbb{R}^{n\times k}$ and $G \in \mathbb{R}^{m\times k}$.
- Assume that $k \le \text{rank}(A) = r$, and consider the problem

$$\min ||E|| = \min_{F\in\mathbb{R}^{n\times k}, G\in\mathbb{R}^{m\times k}} ||A - AF(G^TAF)^{-1}G^TA||, \tag{7.13}$$

  where the norm is **orthogonally invariant** such as the L^2-norm and the Frobenius norm.
- According to Theorem 4.12, p.135, the minimum error is obtained when

$$(AF)(G^TAF)^{-1}(G^TA) = U\Sigma_kV^T = U_k\Sigma_kV_k^T, \tag{7.14}$$

  which is equivalent to choosing

$$F = V_k \quad G = U_k. \tag{7.15}$$

7.3. Text Mining

Definition 7.8. Text mining is methods that extract useful information from large and often unstructured collections of texts.

- A related term is **information retrieval**.
- A typical application is search in data bases of abstract of scientific papers.
  - For instance, in medical applications one may want to find all the abstracts in the data base that deal with a particular syndrome.
  - So one puts together a search phrase, a query, with key words that are relevant for the syndrome.
  - Then the retrieval system is used to match the query to the documents in the data base, and present to the user all the documents that are relevant, preferably ranked according to relevance.

Example 7.9. The following is a typical query (Eldén, 2006) [8].

9. the use of induced hypothermia in heart surgery, neurosurgery, head injuries and infectious diseases. (7.16)

We will refer to this query as **Q9** in the sequel.

Note: Another well-known area of text mining is **web search engines**.

- There the search phrase is usually very short.
- Often there are so many relevant documents that it is out of the question to present them all to the user.
- In that application the **ranking of the search result** is critical for the efficiency of the search engine.
- We will come back to this problem in Section 7.4.

Public Domain Text Mining Software

A number of public domain software are available.

- **R**
  - **textmineR**
- **Python**
  - **nltk** (natural language toolkit)
  - **spaCy** (written in Cython)

In this section

We will review one of the most common methods for text mining, namely the **vector space model** (Salton *et al.*, 1975) [19].

- In **Example 7.1**, we demonstrated the basic ideas of the construction of a **term-document matrix** in the vector space model.
- Below we first give a very brief overview of the **preprocessing** that is usually done before the actual term-document matrix is set up.
- Then we describe a variant of the vector space model: **Latent Semantic Indexing** (LSI) (Deerwester *et al.*, 1990) [6], which is based on the SVD of the term-document matrix.

7.3.1. Vector space model: Preprocessing and query matching

Note: In information retrieval, **keywords** that carry information about the contents of a document are called **terms**.

- A basic task is to create a list of all the terms in alphabetic order, a so called **index**.
- But before the index is made, two preprocessing steps should be done:

  (a) removal of stop words
  (b) stemming

Removal of Stop Words

- **Stop words** are words that one can find in virtually any document.
- The occurrence of such a word in a document does not distinguish this document from other documents.
- The following is the beginning of one stop list

  a, able, about, above, according, accordingly, across, actually, after, afterwards, again, against, ain't, all, allow, allows, almost, alone, along, already, also, although, always, am, among, amongst, an, and, ...

- Various sets of stop words are available on the Internet, e.g. https://countwordsfree.com/stopwords.

Stemming

- **Stemming** is the process of reducing each word that is conjugated or has a suffix to its stem.
- Clearly, from the point of view of information retrieval, no information is lost in the following reduction.

$$\left.\begin{array}{l}\textbf{comput}\text{able}\\ \textbf{comput}\text{ation}\\ \textbf{comput}\text{ing}\\ \textbf{comput}\text{ed}\\ \textbf{comput}\text{ational}\end{array}\right\}\Longrightarrow \textbf{comput}$$

- Public domain stemming algorithms are available on the Internet, e.g. the **Porter Stemming Algorithm** https://tartarus.org/martin/PorterStemmer/.

The Term-Document Matrix

- The **term-document matrix** $A \in \mathbb{R}^{m\times n}$, where

  m = the number of terms in the dictionary
  n = the number of documents

- It is common not only **to count the occurrence of terms in documents** but also **to apply a term weighting scheme**.
- Similarly, **document weighting** is usually done.

Example 7.10. For example, one can define the elements in A by

$$a_{ij} = f_{ij}\log(n/n_i), \tag{7.17}$$

where

- f_{ij} is **term frequency**,
  the number of times term i appears in document j,
- n_i is the number of documents that contain term i
  (*inverse document frequency*).

If a term occurs frequently in only a few documents, then both factors are large. In this case the term discriminates well between different groups of documents, and it gets a large weight in the documents where it appears.

Normally, the term-document matrix is *sparse*: most of the matrix elements are equal to zero.

Example 7.11. For the stemmed Medline collection in Example 7.9, p.309, the matrix is 4163×1063, with 48263 non-zero elements, i.e. approximately 1%. (It includes 30 query columns.) The first 500 rows and columns of the matrix are illustrated in Figure 7.3.

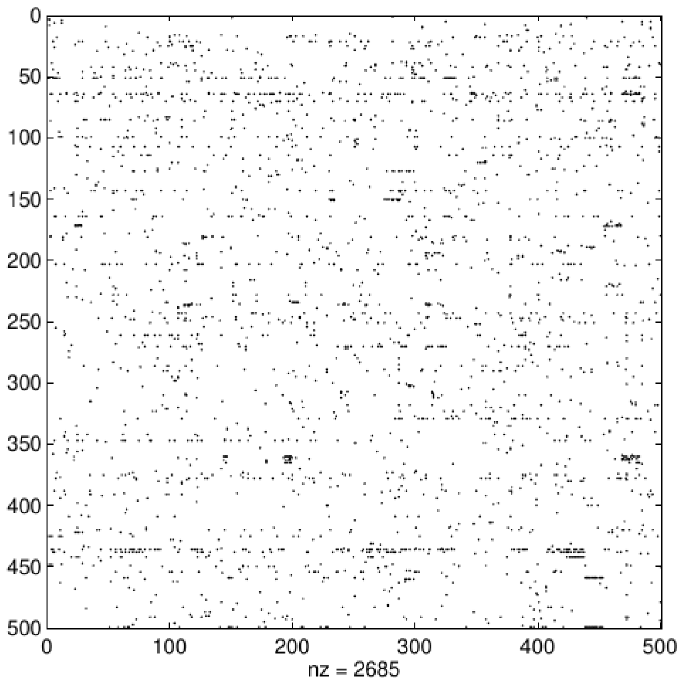


Figure 7.3: The first 500 rows and columns of the Medline matrix. Each dot represents a non-zero element.

Query Matching

- The query is parsed using the same dictionary as the documents, giving a vector $\mathbf{q} \in \mathbb{R}^m$.
- **Query matching** is the process of finding all documents that are considered relevant to a particular query q.
- This is often done using the **cosine distance measure**: All documents $\{\mathbf{a}_j\}$ are returned for which

$$\frac{\mathbf{q} \cdot \mathbf{a}_j}{||\mathbf{q}||\,||\mathbf{a}_j||} \geq \texttt{tol}, \tag{7.18}$$

where `tol` is user-defined tolerance.

Example 7.12. Query matching is performed for query **Q9** in the stemmed Medline collection. With $\texttt{tol} = 0.19$ only a single document is considered relevant. When the tolerance was lowered to 0.17, then three documents are retrieved.

- Irrelevant documents may be returned.
  - For a high value of the tolerance, the retrieved documents are likely to be relevant.
  - When the cosine tolerance is lowered, irrelevant documents may be returned *relatively more*.

Definition 7.13. In performance modelling for information retrieval, we define the following measures:

$$P = \frac{T_r}{T_t} \ (\textbf{precision}) \qquad R = \frac{T_r}{B_r} \ (\textbf{recall}), \tag{7.19}$$

where

T_r = the number of relevant documents retrieved

T_t = the total number of documents retrieved

B_r = the total number of relevant documents in the database

Note: With the cosine measure:

- We see that with a large value of `tol`, we have high precision, but low recall.
- For a small value of `tol`, we have high recall, but low precision.

Example 7.14. Query matching is performed for query **Q9** in the Medline collection using the cosine measure, in order to obtain recall and precision as illustrated in Figure 7.4.

- In the comparison of different methods, it is more illustrative to draw the **recall versus precision diagram**.
- Ideally a method has high recall at the same time as the precision is high. Thus, the closer the curve is to the upper right corner, the better the method is.

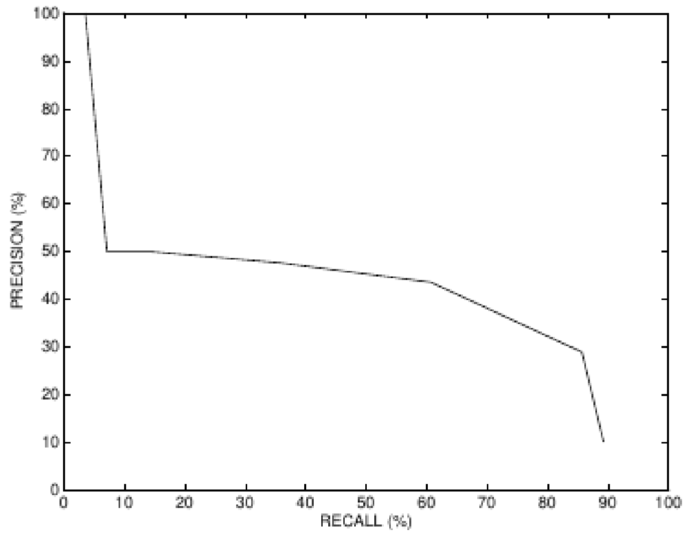


Figure 7.4: Recall versus precision diagram for query matching for Q9, using the vector space method.

7.3.2. Latent Semantic Indexing

Latent Semantic Indexing (**LSI**) is based on the assumption

- that there is **some underlying latent semantic structure** in the data that is corrupted by the wide variety of words used
- and that this semantic structure can be enhanced by projecting the data onto a lower-dimensional space using the **singular value decomposition**.

Algorithm 7.15. Latent Semantic Indexing (LSI)

- Let $A = U\Sigma V^T$ be the SVD of the term-document matrix.
- Let A_k be its approximation of rank k:

$$A_k = U_k\Sigma_k V_k^T = U_k(\Sigma_k V_k^T) =: U_k D_k, \tag{7.20}$$

  where $V_k \in \mathbb{R}^{n\times k}$ so that $D_k \in \mathbb{R}^{k\times n}$.

  - The columns of U_k live in the document space and are an orthogonal basis that we use to approximate all the documents.
  - Column j of D_k holds the coordinates of document j in terms of the orthogonal basis.

- Note that

$$\mathbf{q}^T A_k = \mathbf{q}^T U_k D_k = (U_k^T\mathbf{q})^T D_k \in \mathbb{R}^{1\times n}. \tag{7.21}$$

- Thus, in query matching, we compute the coordinates of the query in terms of the new document basis and compute the cosines from

$$\cos\theta_j = \frac{\widehat{\mathbf{q}}_k \cdot (D_k\mathbf{e}_j)}{||\widehat{\mathbf{q}}_k||\,||D_k\mathbf{e}_j||}, \quad \widehat{\mathbf{q}}_k = U_k^T\mathbf{q}. \tag{7.22}$$

- This means that the query-matching is performed in a k-dimensional space.

Example 7.16. Query matching is carried out for **Q9** in the Medline collection, approximating the matrix using the truncated SVD with of rank 100 ($k = 100$). The recall-precision curve is given in Figure 7.5. It is seen that for this query, the LSI improves the retrieval performance.

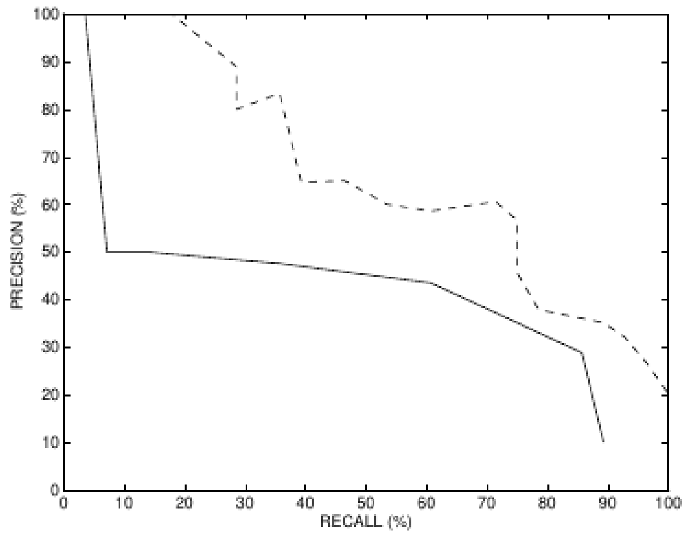


Figure 7.5: Recall versus precision diagram for query matching for Q9, using the full vector space method (solid curve) and the rank 100 approximation (dashed).

Example 7.17. Recall Example 7.1. Consider the term-document matrix $A \in \mathbb{R}^{10\times 5}$ and the query vector $\mathbf{q} \in \mathbb{R}^{10}$, of which the query is "**ranking** of **web pages**". See pages 302–303 for details.

Document 1:	The **Google matrix** P is a model of the **Internet**.
Document 2:	P_{ij} is nonzero if there is a **link** from **web page** j to i.
Document 3:	The **Google matrix** is used to **rank** all **web pages**.
Document 4:	The **ranking** is done by solving a **matrix eigenvalue** problem.
Document 5:	**England** dropped out of the top 10 in the **FIFA ranking**.

$$A = \begin{bmatrix} 0&0&0&1&0\\ 0&0&0&0&1\\ 0&0&0&0&1\\ 1&0&1&0&0\\ 1&0&0&0&0\\ 0&1&0&0&0\\ 1&0&1&1&0\\ 0&1&1&0&0\\ 0&0&1&1&1\\ 0&1&1&0&0 \end{bmatrix} \in \mathbb{R}^{10\times 5}, \qquad \mathbf{q} = \begin{bmatrix} 0\\0\\0\\0\\0\\0\\0\\1\\1\\1 \end{bmatrix} \in \mathbb{R}^{10}.$$

- (Eldén, 2006) [8]
  "**Obviously, Documents 1-4 are relevant with respect to the query, while Document 5 is totally irrelevant.** However, we obtain the following cosines for query and the original data

  $$(0 \;\; 0.6667 \;\; 0.7746 \;\; 0.3333 \;\; 0.3333)$$

  We then compute the SVD of the term-document matrix, and use a rank 2 approximation. After projection to the two-dimensional subspace the cosines, computed according to (7.22), are

  $$(0.7857 \quad 0.8332 \quad 0.9670 \quad 0.4873 \quad 0.1819)$$

  It turns out that Document 1, which was deemed totally irrelevant for the query in the original representation, is now highly relevant. In addition, the scores for the relevant Documents 2-4 have been reinforced. At the same time, the score for Document 5 has been significantly reduced."
- **However, I view it as a warning.**

7.4. Eigenvalue Methods in Data Mining

Note: An **Internet search** performs two major operations, using a search engine.

(a) **Traditional text processing**. The aim is to find all the web pages containing the words of the query.

(b) **Sorting out**.

- Due to the massive size of the Web, the number of hits is likely to be much too large to be handled by the user.
- Therefore, some measure of quality is needed to sort out the pages that are likely to be most relevant to the particular query.

When one uses a web search engine, then typically the search phrase is under-specified.

Example 7.18. A **Google search** conducted on October 21, 2022, using the search phrase "**university**":

- The result: links to universities, including *Mississippi State University, University of Arizona, University of Washington - Seattle, University of Wisconsin–Madison, The University of Texas at Austin*, and *University of Southern California - Los Angeles*.
- The total number of web pages relevant to the search phrase was more than 7 billions.

Remark 7.19. Google uses an algorithm (**Pagerank**) for ranking all the web pages that agrees rather well with a common-sense quality measure.

- Google assigns a high rank to a web page, if it has **inlinks** from other pages that have a high rank.
- We will see that this "**self-referencing**" statement can be formulated mathematically as an eigenvalue problem.

7.4.1. Pagerank

Note: Google uses the concept of **Pagerank** as a quality measure of web pages. It is based on the assumption that

> ***the number of links to and from a page give information about the importance of a page.***

- Let all web pages be ordered from 1 to n, and let i be a particular web page.
- Then O_i will denote the set of pages that i is linked to, the **outlinks**. The number of outlinks is denoted $N_i = |O_i|$.
- The set of **inlinks**, denoted I_i, are the pages that have an outlink to i.

Note: In general, a page i can be considered as ***more important the more inlinks it has***.

- However, a ranking system based ***only*** *on the number of inlinks* is easy to manipulate.
  - When you design a web page i that you would like to be seen by as many as possible, you could simply create a large number of (information-less and unimportant) pages that have outlinks to i.
- In order to discourage this, one may define **the rank of i** in such a way that if **a highly ranked page j**, has an outlink to i, this should add to the importance of i.
- Here the manner is:

  > ***the rank of page i is a weighted sum of the ranks of the pages that have outlinks to i.***

Definition 7.20. The preliminary definition of **Pagerank** is

$$r_i = \sum_{j \in I_i} \frac{r_j}{N_j}, \quad i = 1, 2, \cdots, n. \tag{7.23}$$

That is, the weighting is such that the rank of a page j is **divided evenly** among its outlinks.

Remark 7.21. Pagerank may not be solvable.

- As in (7.3)-(7.4), p. 305, let Q be a square matrix of dimension n, and let

$$Q_{ij} = \begin{cases} 1/N_j, & \text{if there is a link from } j \text{ to } i, \\ 0, & \text{otherwise,} \end{cases} \tag{7.24}$$

  where Q is sometimes called the **normalized web matrix**.
- Then, (7.23) can be written as

$$\lambda \mathbf{r} = Q\mathbf{r}, \quad \lambda = 1, \tag{7.25}$$

  i.e., r is an eigenvector of Q with eigenvalue $\lambda = 1$.
- However, it is not clear that Pagerank is well-defined, because we do not know if there exists an eigenvalue equal to 1.

Reformulation of (7.23)

Modify the matrix Q to have an eigenvalue $\lambda = 1$.

- Assume that a surfer visiting a web page, always chooses the next page among the outlinks with equal probability.
- Assume that the random surfer never get stuck.
  - In other words, there should be no web pages without outlinks (such a page corresponds to a zero column in Q).
- Therefore the model is modified so that zero columns are replaced by a constant value in each position.

- Define the vectors

$$\mathbf{e} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^n, \qquad \mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix}, \; d_j = \begin{cases} 1, & \text{if } N_j = 0, \\ 0, & \text{otherwise}. \end{cases} \tag{7.26}$$

- Then the modified matrix is defined

$$P = Q + \frac{1}{n}\mathbf{e}\mathbf{d}^T. \tag{7.27}$$

- Then P is a **column-stochastic matrix**, of which columns are probability vectors. That is, it has non-negative elements ($P \geq 0$) and the sum of each column is 1.
- Furthermore,

$$\mathbf{e}^T P = \mathbf{e}^T Q + \frac{1}{n}\mathbf{e}^T\mathbf{e}\mathbf{d}^T = \mathbf{e}^T Q + d^T = \mathbf{e}^T, \tag{7.28}$$

  which implies that $\lambda = 1$ is a **left eigenvalue** and therefore a **right eigenvalue**. Note that

$$\begin{aligned} &\mathbf{e}^T P = \mathbf{e}^T \iff P^T\mathbf{e} = \mathbf{e}, \\ &\det(A - \lambda I) = \det(A^T - \lambda I). \end{aligned} \tag{7.29}$$

- Now, we define the **Pagerank vector** r as *a unique eigenvector of* P *with eigenvalue* $\lambda = 1$,

$$P\mathbf{r} = \mathbf{r}. \tag{7.30}$$

- However, uniqueness is still not guaranteed.
  - To ensure this, the directed graph corresponding to the matrix must be **strongly connected**
  - Equivalently, in matrix terms, P must be **irreducible**.
  - Equivalently, there must not exist any subgraph, which has no outlinks.

The uniqueness of the eigenvalue is guaranteed by the **Perron-Frobenius theorem**.

Theorem 7.22. (Perron-Frobenius) *If $A \in \mathbb{R}^{n\times n}$ is nonnegative, then*

- *$\rho(A)$ is an eigenvalue of A.*
- *There is a nonnegative eigenvector $\mathbf{x}$ such that $A\mathbf{x} = \rho(A)\mathbf{x}$.*

Theorem 7.23. (Perron-Frobenius) *If $A \in \mathbb{R}^{n\times n}$ is nonnegative and irreducible, then*

- *$\rho(A)$ is an eigenvalue of A.*
- *$\rho(A) > 0$.*
- *There is a positive eigenvector $\mathbf{x}$ such that $A\mathbf{x} = \rho(A)\mathbf{x}$.*
- *$\rho(A)$ is a simple eigenvalue.*

Theorem 7.24. (Perron) *If $A \in \mathbb{R}^{n\times n}$ is positive, then*

- *Theorem 7.23 holds, and in addition,*
- *$|\lambda| < \rho(A)$ for any eigenvalue λ with $\lambda \neq \rho(A)$.*

Corollary 7.25. *Let A be an irreducible column-stochastic matrix. Then*

- *The largest eigenvalue in magnitude is equal to 1.*
- *There is a unique corresponding eigenvector $\mathbf{r}$ satisfying $\mathbf{r} > 0$ and $||\mathbf{r}||_1 = 1$; this is the only eigenvector that is non-negative.*
- *If $A > 0$, then $|\lambda_i| < 1$, $i = 2, 3, \cdots, n$.*

Remark 7.26. Given the size of the Internet and reasonable assumptions about its structure,

- it is highly probable that the **link graph** is not strongly connected,
- which means that **the Pagerank eigenvector of P may not be well-defined.**

7.4.2. The Google matrix

To ensure connectedness, i.e., to make it impossible for the random walker to get trapped in a subgraph, **one can add, artificially, a link from every web page to all the other**. In matrix terms, this can be made by taking a convex combination of P and a rank one matrix.

One billion dollar idea, by **Sergey Brin** and **Lawrence Page** in 1996

- The **Google matrix** is the matrix

$$G = \alpha P + (1-\alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T, \tag{7.31}$$

  for some α satisfying $0 < \alpha < 1$, called the **damping factor**.
- Obviously G is irreducible (since $G > 0$) and column-stochastic.[a]
- Furthermore,

$$\mathbf{e}^T G = \alpha \mathbf{e}^T P + (1-\alpha)\mathbf{e}^T \frac{1}{n}\mathbf{e}\mathbf{e}^T = \alpha \mathbf{e}^T + (1-\alpha)\mathbf{e}^T = \mathbf{e}^T. \tag{7.32}$$

- The **pagerank equation** reads

$$G\mathbf{r} \;=\; \left[\alpha P + (1-\alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T\right]\mathbf{r} \;=\; \mathbf{r}. \tag{7.33}$$

[a] A $n \times n$ matrix is called a **Markov matrix** if all entries are nonnegative and the sum of each column vector is equal to 1. A Markov matrix are also called a **stochastic matrix**.

Note: The random walk interpretation of the additional rank one term is that each time step a page is visited, the surfer will jump to any page in the whole web with probability $1-\alpha$ (sometimes referred to as **teleportation**).

- Recall (7.27): $P = Q + \dfrac{1}{n}\mathbf{e}\mathbf{d}^T$, which can be interpreted as follows.

  When a random surfer visits a web page of no outlinks, the surfer will jump to any page with an equal probability $1/n$.

- The convex combination in (7.31): $G = \alpha P + (1-\alpha)\dfrac{1}{n}\mathbf{e}\mathbf{e}^T$.

  Although there are outlinks, the surfer will jump to any page with an equal probability $(1-\alpha)/n$.

Proposition 7.27. Let the eigenvalues of the column-stochastic matrix P be $\{1, \lambda_2, \lambda_3, \cdots, \lambda_n\}$. Then, the eigenvalues of $G = \alpha P + (1-\alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T$ are $\{1, \alpha\lambda_2, \alpha\lambda_3, \cdots, \alpha\lambda_n\}$.

- This means that even if P has a multiple eigenvalue equal to 1, the second largest eigenvalue in magnitude of G is equal to α.

Remark 7.28. The vector $\dfrac{1}{n}\mathrm{e}$ in (7.31) can be replaced by a non-negative vector v with $||\mathrm{v}||_1 = 1$, which can be chosen in order to make the search biased towards certain kinds of web pages. Therefore, it is referred to as a **personalization vector**.

7.4.3. Solving the Pagerank equation

Now, we should solve the Pagerank equation, an eigenvalue problem

$$G\mathbf{r} = \mathbf{r}, \tag{7.34}$$

where $\mathbf{r} \geq 0$ with $||\mathbf{r}||_1 = 1$.

Observation 7.29. The Google matrix $G \in \mathbb{R}^{n\times n}$

- G is a full matrix, although it is not necessary to construct it explicitly.
- n represents the number of all web pages, which is order of billions.
- It is **impossible** to use **sparse eigenvalue algorithms** that require the storage of more than very few vectors.

The only viable method so far for Pagerank computations on the whole web seems to be the **power method**. See (5.13), p.188.

- The rate of convergence of the power method depends on the ratio of the second largest and the largest eigenvalue in magnitude.
- Here, we have
$$|1 - \lambda^{(k)}| = \mathcal{O}(\alpha^k), \tag{7.35}$$
due to Proposition 7.27.
- In view of the huge dimension of the Google matrix, it is non-trivial to compute the matrix-vector product. We will consider some details.

The power method: matrix-vector product

Recall: It follows from (7.24), (7.27), and (7.31) that the **Google matrix** is formulated as

$$G = \alpha P + (1-\alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T, \tag{7.36}$$

where

$$P = Q + \frac{1}{n}\mathbf{e}\mathbf{d}^T.$$

Here Q is the **link matrix** and e and d are defined as in (7.26).

$\boxed{\textbf{Derivation}}$ **7.30.** Let $\mathbf{z} = G\mathbf{y}$.

- **Normalization-free**: Since G is column-stochastic ($\mathbf{e}^T G = \mathbf{e}^T$),

$$||\mathbf{z}||_1 = \mathbf{e}^T\mathbf{z} = \mathbf{e}^T G\mathbf{y} = \mathbf{e}^T\mathbf{y} = ||\mathbf{y}||_1. \tag{7.37}$$

  Thus, when the power method begins with $\mathbf{y}^{(0)}$ with $||\mathbf{y}^{(0)}||_1 = 1$, the normalization step in the power method is unnecessary.

- Let us look at the multiplication in some detail:

$$\mathbf{z} \;=\; \Big[\alpha P + (1-\alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T\Big]\mathbf{y} \;=\; \alpha Q\mathbf{y} + \beta\frac{\mathbf{e}}{n}, \tag{7.38}$$

  where

$$\beta \;=\; \alpha\mathbf{d}^T\mathbf{y} + (1-\alpha)\mathbf{e}^T\mathbf{y}. \tag{7.39}$$

- Apparently we need to know which pages lack outlinks (d), in order to find β. However, in reality, we do not need to define d. It follows from (7.37) and (7.38) that

$$\beta = 1 - \alpha\mathbf{e}^T Q\mathbf{y} = 1 - ||\alpha Q\mathbf{y}||_1. \tag{7.40}$$

Algorithm 7.31. The following Matlab code implements the matrix vector multiplication: $\mathbf{z} = G\mathbf{y}$.

```
zhat = alpha*Q*y;
beta = 1-norm(zhat,1);
z = zhat + beta*v;
residual = norm(y-z,1);
```

Here v is $(1/n)$e or a **personalization vector**; see Remark 7.28.

Note:

- From Proposition 7.27, we know that the second eigenvalue of the Google matrix is $\alpha\lambda_2$.
- A typical value of $\alpha = 0.85$.
- Approximately $k = 57$ iterations are needed to reach $0.85^k < 10^{-4}$.
- This is reported to be close the number of iterations used by Google.

Exercises for Chapter 7

7.1. Consider Example 7.17, p.318. Compute vectors of cosines, for each subspace approximations, i.e., with A_k where $k = 1, 2, \cdots, 5$.

7.2. Verify equations in Derivation 7.30, p.327, particularly (7.38), (7.39), and (7.40).

7.3. Consider the link matrix Q in (7.4) and its corresponding link graph in Figure 7.2. Find the pagerank vector r by solving the Google pagerank equation.

- You may initialize the power method with any vector $\mathbf{r}^{(0)}$ satisfying $||\mathbf{r}^{(0)}||_1 = 1$.
- Set $\alpha = 0.85$.
- Let the iteration stop, when `residual` $< 10^{-4}$.

7.4. Now, consider a **modified** link matrix $\widetilde{Q}$, by adding an outlink from page ④ to ⑤ in Figure 7.2. Find the pagerank vector $\widetilde{\mathrm{r}}$, by setting parameters and initialization the same way as for the previous problem.

- Compare r with $\widetilde{\mathrm{r}}$.
- Compare the number of iterations for convergence.

CHAPTER P

Projects

Finally we add projects.

Contents of Projects

P.1. mCLESS

Note: Some **machine learning** algorithms are considered as **black boxes**, because

- the models are sufficiently complex and
- they are not straightforwardly interpretable to humans.

Lack of interpretability in predictive models can undermine trust in those models, especially in health care, in which so many decisions are – literally – life and death issues [17].

Project Objectives

- Develop a family of **interpretable** machine learning algorithms.
  - We will develop algorithms involving least-squares formulation.
  - The family is called the *Multi-Class Least Error Square Sum* (**mCLESS**).
- Compare with traditional methods, using public domain datasets.

P.1.1. What is machine learning?

Definition P.1. Machine learning (ML)

- **ML algorithms** are algorithms that can learn from **data** (input) and produce **functions/models** (output).
- **Machine learning** is the science of getting machines to act, without functions/models being explicitly programmed to do so.

Example P.2. There are three different types of ML:

- **Supervised learning**: e.g., classification, regression
  - Labeled data
  - Direct feedback
  - Predict outcome/future
- **Unsupervised learning**: e.g., clustering
  - No labels
  - No feedback
  - Find hidden structure in data
- **Reinforcement learning**: e.g., chess engine
  - Decision process
  - Reward system
  - Learn series of actions

Note: The most popular type is **supervised learning**.

Supervised Learning

Assumption. Given a data set $\{(\mathbf{x}_i, y_i)\}$, where y_i are labels, there exists a relation $f : X \to Y$.

Supervised learning:

$$\begin{cases} \text{Given}: & \text{A } \textbf{training data } \{(\mathbf{x}_i, y_i) \mid i = 1, \cdots, N\} \\ \text{Find}: & \widehat{f} : X \to Y, \quad \text{a good approximation to } f \end{cases} \tag{P.1.1}$$

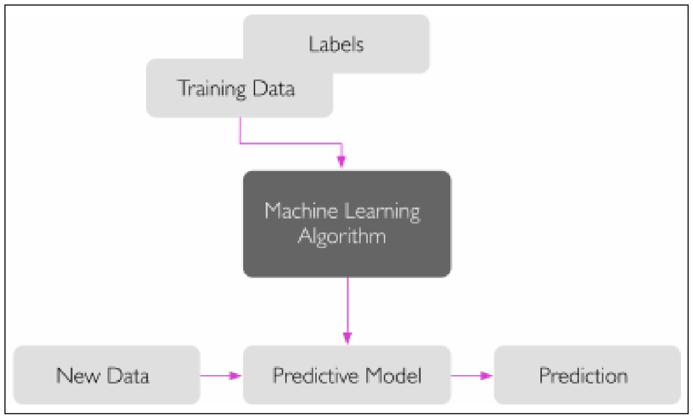


Figure P.1: Supervised learning and prediction.

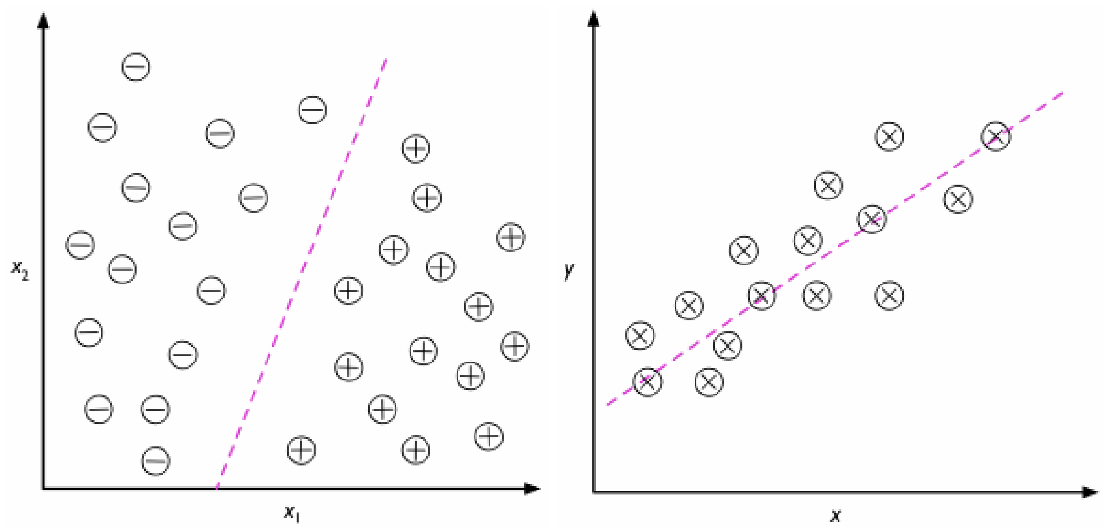


Figure P.2: Classification and regression.

P.1.2. Simple classifiers

The **Perceptron** [18] (or Adaline) is the simplest artificial neuron that makes decisions for datasets of two classes by *weighting up evidence*.

- Inputs: feature values $\mathbf{x} = [x_1, x_2, \cdots, x_d]$
- Weight vector and bias: $\mathbf{w} = [w_1,\, w_2,\, \cdots\,,\, w_d]^T$, w_0
- Net input:
$$z = w_0 + w_1\, x_1 + w_2\, x_2 + \cdots + w_d\, x_d \tag{P.1.2}$$
- *Activation*:
$$\phi(z) = \begin{cases} 1, & \text{if } z \ge \theta \\ 0, & \text{otherwise,} \end{cases} \tag{P.1.3}$$
where θ is a threshold. When the logistic sigmoid function is chosen for the **activation function**, i.e., $\phi(z) = 1/(1 + e^{-z})$, the resulting classifier is called the **Logistic Regression**.

Remark P.3. Note that the net input in (P.1.2) represents a **hyperplane** in $\mathbb{R}^d$.

- More complex neural networks can be built, stacking the simple artificial neurons as building blocks.
- Machine learning (ML) is to train weights from datasets of an arbitrary number of classes.
  - The weights must be trained in such a way that *data points in a class are heavily weighted by the corresponding part of weights.*
- The **activation function** is incorporated in order
  (a) **to keep the net input restricted to a certain limit** as per our requirement and, more importantly,
  (b) **to add nonlinearity** to the network.

P.1.3. The mCLESS classifier

Here we present a new classifier which is based on a least-squares formulation and able to classify datasets having arbitrary numbers of classes. Its nonlinear expansion will also be suggested.

Two-layer Neural Networks

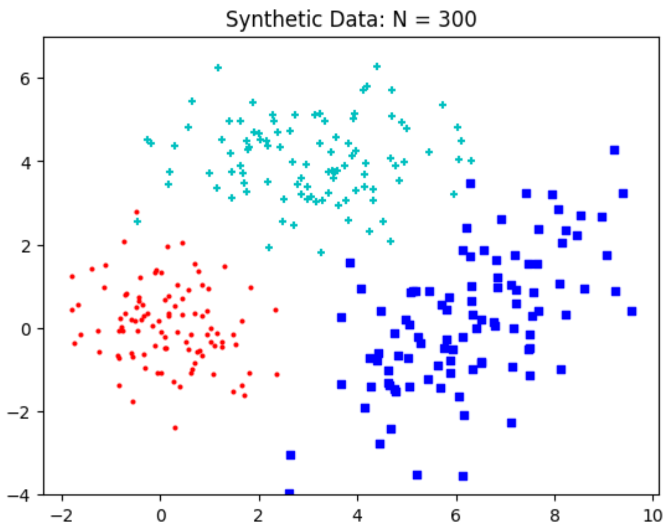


Figure P.3: A synthetic data of three classes.

- In order to describe the proposed algorithm effectively, we exemplify a synthetic data of three classes, as shown in Figure P.3, in which each class has 100 points.
- A point in the c-th class is expressed as

$$\mathbf{x}^{(c)} = [x_1^{(c)}, x_2^{(c)}] = [x_1, x_2, c] \quad c = 0, 1, 2,$$

  where the number in () in the superscript denotes the class that the point belongs.
- Let's consider an artificial neural network of the identity activation and no hidden layer, for simplicity.

A set of weights can be trained in a way that *points in a class are **heavily** weighted by the corresponding part of weights*, i.e.,

$$w_0^{(j)} + w_1^{(j)} x_1^{(i)} + w_2^{(j)} x_2^{(i)} = \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{P.1.4}$$

where δ_{ij} is called the Kronecker delta and $w_0^{(j)}$ is a bias for the class j.

- Thus, for neural networks which classify a dataset of C classes with points in $\mathbb{R}^d$, the weights to be trained must have dimensions $(d+1)\times C$.
- The weights can be computed by the least-squares method.
- We will call the algorithm the *Multi-Class Least Error Square Sum* (**mCLESS**).

Training in the mCLESS

- **Dataset**: We express the dataset $\{X, \mathbf{y}\}$ used for Figure P.3 by

$$X = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ \vdots & \vdots \\ x_{N1} & x_{N2} \end{bmatrix} \in \mathbb{R}^{N\times 2}, \quad \mathbf{y} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix}, \tag{P.1.5}$$

  where $c_i \in \{0, 1, 2\}$, the class number.

- **The algebraic system**: It can be formulated using (P.1.4).
  - Define the **information matrix**:

$$A = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ & \vdots & \\ 1 & x_{N1} & x_{N2} \end{bmatrix} \in \mathbb{R}^{N\times 3}. \tag{P.1.6}$$

    Note. The information matrix can be made using

```
A = np.column_stack((np.ones([N,]),X))
```

  - The **weight matrix** to be learned is:

$$W = [\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \mathbf{w}^{(2)}] = \begin{bmatrix} w_0^{(0)} & w_0^{(1)} & w_0^{(2)} \\ w_1^{(0)} & w_1^{(1)} & w_1^{(2)} \\ w_2^{(0)} & w_2^{(1)} & w_2^{(2)} \end{bmatrix}, \tag{P.1.7}$$

where the j-th column weights heavily points in the j-th class.

- Define the **source matrix**:

$$B = [\delta_{c_i,j}] \in \mathbb{R}^{N\times 3}. \tag{P.1.8}$$

For example, if the i-th point is in Class 0, then the i-th row of B is $[1, 0, 0]$.

- Then the **multi-column least-squares** (MC-LS) problem reads

$$\widehat{W} = \arg\min_{W} ||AW - B||^2, \tag{P.1.9}$$

which can be solved by the **method of normal equations**:

$$(A^T A)\,\widehat{W} = A^T B, \qquad A^T A \in \mathbb{R}^{3\times 3}. \tag{P.1.10}$$

- **The output of training**: The weight matrix $\widehat{W}$.

Note: The normal matrix $A^T A$ is occasionally singular, particularly for small datasets. In the case, the MC-LS problem can be solved using the **singular value decomposition (SVD)**.

Prediction in the mCLESS

The prediction step in the mCLESS is quite simple:

(a) Let $[x_1, x_2]$ be a new point.

(b) Compute

$$[1, x_1, x_2]\,\widehat{W} \;=\; [p_0, p_1, p_2], \qquad \widehat{W} \in \mathbb{R}^{3\times 3}. \tag{P.1.11}$$

Note. Ideally, if the point $[x_1, x_2]$ is in class j, then p_j is near 1, while others would be near 0. Thus p_j is the largest.

(c) Decide the class c:

$$\texttt{c} = \texttt{np.argmax}([p_0, p_1, p_2], \texttt{axis} = 1). \tag{P.1.12}$$

Experiment **P.4. mCLESS, with a Synthetic Dataset**

- As a preprocessing, the dataset X is scaled column-wisely so that the maximum value in each column is 1 in modulus.
- The training is carried out with randomly selected 70% the dataset.
- The output of training, $\widehat{W}$, represents three sets of parallel lines.
  - Let $[w_0^{(j)}, w_1^{(j)}, w_2^{(j)}]^T$ be the j-th column of $\widehat{W}$. Define $L_j(x_1, x_2)$ as
  $$L_j(x_1, x_2) = w_0^{(j)} + w_1^{(j)} x_1 + w_2^{(j)} x_2, \quad j = 0, 1, 2. \qquad \text{(P.1.13)}$$
  - Figure P.4 depicts $L_j(x_1, x_2) = 0$ and $L_j(x_1, x_2) = 1$ superposed on the training set.
- It follows from (P.1.12) that the mCLESS can be viewed as an **one-versus-rest** (**OVR**) classifier.

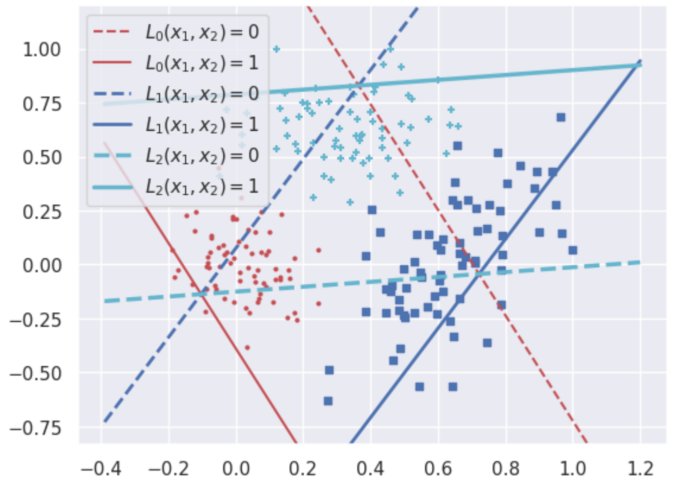


Figure P.4: Lines represented by the weight vectors. mCLESS is interpretable!

The whole algorithm (training-prediction) is run 100 times, with randomly splitting the dataset into 70:30 parts respectively for training and prediction; which results in 98.37% and 0.00171 sec for the average accuracy and e-time. The used is a laptop of an Intel Core i7-10750H CPU at 2.60GHz.

P.1.4. Feature expansion

Remark P.5. Nonlinear mCLESS

- The mCLESS so far is a **linear classifier**.
- As for other classifiers, its nonlinear expansion begins with a data transformation, more precisely, **feature expansion**.
- For example, the **Support Vector Machine (SVM)** replaces the dot product of feature vectors (point) with the result of a kernel function applied to the feature vectors, in the construction of the Gram matrix:

  $$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) \approx \sigma(\mathbf{x}_i) \cdot \sigma(\mathbf{x}_j),$$

  where σ is a function for feature expansion.

  – Thus, without an explicit expansion of feature vectors, the SVM can incorporate the effect of data transformation effectively. Such a technique is called the **kernel trick**.
- However, the **mCLESS** does not incorporate dot products between points.

  – As a result, we must *perform feature expansion without a kernel trick*, which results in an augmented normal matrix, expanded in both column and row directions.

Feature Expansion for mCLESS

- A feature expansion is expressed as

$$\left\{\begin{array}{l}\mathbf{x} = [x_1, x_2, \cdots, x_d]\\ \mathbf{w} = [w_0, w_1, \cdots, w_d]^T\end{array}\right. \quad\Rightarrow\quad \left\{\begin{array}{l}\widetilde{\mathbf{x}} = [x_1, x_2, \cdots, x_d, \sigma(\mathbf{x})]\\ \widetilde{\mathbf{w}} = [w_0, w_1, \cdots, w_d, w_{d+1}]^T\end{array}\right. \tag{P.1.14}$$

  where $\sigma()$ is a **feature function** of x.

- Then, the expanded weights must be trained to satisfy

$$[1, \widetilde{\mathbf{x}}^{(i)}]\,\widetilde{\mathbf{w}}^{(j)} = w_0^{(j)} + w_1^{(j)}x_1^{(i)} + \cdots + w_d^{(j)}x_d^{(i)} + w_{d+1}^{(j)}\sigma(\mathbf{x}^{(i)}) = \delta_{ij}, \tag{P.1.15}$$

  for all points in the dataset. Compare the equation with (P.1.4).

- The corresponding expanded information and weight matrices read

$$\widetilde{A} = \left[\begin{array}{ccccc:c} 1 & x_{11} & x_{12} & \cdots & x_{1d} & \sigma(\mathbf{x}_1)\\ 1 & x_{21} & x_{22} & \cdots & x_{2d} & \sigma(\mathbf{x}_2)\\ & \vdots & & \ddots & & \vdots\\ 1 & x_{N1} & x_{N2} & \cdots & x_{Nd} & \sigma(\mathbf{x}_N)\end{array}\right], \quad \widetilde{W} = \left[\begin{array}{cccc} w_0^{(0)} & w_0^{(1)} & \cdots & w_0^{(C-1)}\\ w_1^{(0)} & w_1^{(1)} & \cdots & w_1^{(C-1)}\\ \vdots & & \ddots & \vdots\\ w_d^{(0)} & w_d^{(1)} & \cdots & w_d^{(C-1)}\\ \hdashline w_{d+1}^{(0)} & w_{d+1}^{(1)} & \cdots & w_{d+1}^{(C-1)}\end{array}\right], \tag{P.1.16}$$

  where $\widetilde{A} \in \mathbb{R}^{N\times(d+2)}$, $\widetilde{W} \in \mathbb{R}^{(d+2)\times C}$, and C is the number of classes.

- Feature expansion can be performed multiple times. When α features are added, the optimal weight matrix $\widehat{W} \in \mathbb{R}^{(d+1+\alpha)\times C}$ is the least-squares solution of

$$(\widetilde{A}^T\widetilde{A})\,\widehat{W} = \widetilde{A}^T B, \tag{P.1.17}$$

  where $\widetilde{A}^T\widetilde{A} \in \mathbb{R}^{(d+1+\alpha)\times(d+1+\alpha)}$ and B is the same as in (P.1.8).

Remark P.6. Various feature functions $\sigma()$ can be considered. Here we will focus on the **feature function** of the form

$$\sigma(\mathbf{x}) = \|\mathbf{x} - \mathbf{p}\|, \tag{P.1.18}$$

the Euclidean distance between x and a prescribed point p.
Now, the question is: "*How can we find* p?"

What to do

1. Implement mCLESS.
   - **Training**. You should implement modules for each of (P.1.6) and (P.1.8). Then use `Xtrain` and `ytrain` to get A and B.
   - **Test**. Use the same module (implemented for A) to get `Atest` from `Xtest`. Then perform `P = (Atest)*`$\widehat{W}$ as in (P.1.11). Now, you can get the prediction using

     ```
     prediction = np.argmax(P,axis=1);
     ```

     which may be compared with `ytest` to obtain accuracy.
2. Also, add modules for feature expansion, as described on page 341.
   - For this, try to an **interpretable strategy** to find an effective point p such that the feature expansion with (P.1.18) improves accuracy.
3. Use datasets such as `iris` and `wine`. To get them:

   ```
   from sklearn import datasets
   data_read1 = datasets.load_iris()
   data_read2 = datasets.load_wine()
   ```
4. Report your experiments with the code and results.

You may start with the following **machine learning modelcode**; add your own modules.

Machine_Learning_Model.py

```
import numpy as np;    import pandas as pd
import seaborn as sbn; import matplotlib.pyplot as plt
import time
from sklearn.model_selection import train_test_split
from sklearn import datasets; #print(dir(datasets))
np.set_printoptions(suppress=True)

#=====================================================================
# DATA: Read & Preprocessing
# load_iris, load_wine, load_breast_cancer, ...
#=====================================================================
data_read = datasets.load_iris();  #print(data_read.keys())

X = data_read.data
y = data_read.target
```

```
datafile = data_read.filename
targets  = data_read.target_names
features = data_read.feature_names

print('X.shape=',X.shape, 'y.shape=',y.shape)

#-----------------------------------------------------------------------
# SETTING
#-----------------------------------------------------------------------
N,d = X.shape; labelset=set(y)
nclass=len(labelset);
print('N,d,nclass=',N,d,nclass)

rtrain = 0.7e0; run = 100
rtest  = 1-rtrain

#=======================================================================
# CLASSIFICATION
#=======================================================================
btime = time.time()
Acc = np.zeros([run,1])
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(5)

for it in range(run):
    Xtrain, Xtest, ytrain, ytest = train_test_split(
        X, y, test_size=rtest, random_state=it, stratify = y)
    clf.fit(Xtrain, ytrain);
    Acc[it] = clf.score(Xtest, ytest)

#-----------------------------------------------
# Print: Accuracy && E-time
#-----------------------------------------------
etime = time.time()-btime
print('   %s: Acc.(mean,std) = (%.2f,%.2f)%%; Average E-time= %.5f'
          %(datafile,np.mean(Acc)*100,np.std(Acc)*100,etime/run))

#=======================================================================
# Scikit-learn Classifiers, for Comparisions
#=======================================================================
#exec(open("sklearn_classifiers.py").read())
```

sklearn_classifiers.py

```
#=====================================================================
# Required: X, y, datafile
print('========= Scikit-learn Classifiers, for Comparisions =========')
#=====================================================================
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.inspection import DecisionBoundaryDisplay

#-----------------------------------------------
names = [
    "Logistic Regr",
    "KNeighbors-7 ",
    "Linear SVM   ",
    "RBF SVM      ",
    "Random Forest",
    "Deep-NN      ",
    "AdaBoost     ",
    "Naive Bayes  ",
    "QDA          ",
    "Gaussian Proc",
]

#-----------------------------------------------
classifiers = [
    LogisticRegression(max_iter = 1000),
    KNeighborsClassifier(7),
    SVC(kernel="linear", C=0.5),
    SVC(gamma=2, C=1),
    RandomForestClassifier(max_depth=5, n_estimators=50, max_features=1),
    MLPClassifier(alpha=1, max_iter=1000),
    AdaBoostClassifier(),
    GaussianNB(),
    QuadraticDiscriminantAnalysis(),
    GaussianProcessClassifier(),
]

#-----------------------------------------------
acc_max=0
for name, clf in zip(names, classifiers):
    Acc = np.zeros([run,1])
    btime = time.time()

    for it in range(run):
        Xtrain, Xtest, ytrain, ytest = train_test_split(
            X, y, test_size=rtest, random_state=it, stratify = y)

        clf.fit(Xtrain, ytrain);
        Acc[it] = clf.score(Xtest, ytest)

    etime = time.time()-btime
    accmean = np.mean(Acc)*100
    print('%s: %s: Acc.(mean,std) = (%.2f,%.2f)%%; E-time= %.5f'
           %(datafile,name,accmean,np.std(Acc)*100,etime/run))
    if accmean>acc_max:
        acc_max= accmean; algname = name
print('sklearn classifiers max: %s= %.2f' %(algname,acc_max))
```

Bibliography

[1] O. Alter, P. O. Brown, and D. Botstein, *Singular value decomposition for genome-wide expression data processing and modeling*, PNAS, 97 (2000), pp. 10101–10106.

[2] O. Alter and G. H. Golub, *Integrative analysis of genome-scale data by using pseudoinverse projection predicts novel correlation between DNA replication and RNA transcription*, PNAS, 101 (2004), pp. 16577–16582.

[3] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the solution of linear systems: Building blocks for iterative methods*, SIAM, Philadelphia, 1994. The postscript file is free to download from `http://www.netlib.org/templates/` along with source codes.

[4] N. M. Bertagnolli, J. A. Drake, J. M. Tennessen, and O. Alter, *SVD identifies transcript length distribution functions from DNA microarray data and reveals evolutionary forces globally affecting GBM metabolism*, PLoS One, 8 (2013), p. e78913.

[5] R. Dai, *Richardson extrapolation-based high accuracy high efficiency computation for partial differential equations*, Theses and Dissertations – Computer Science 20, University of Kentucky, Lexington, KY 40506, 2014.

[6] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman, *Indexing by latent semantic analysis.*, Journal of the American Society for Information Science, (1990), pp. 391–407.

[7] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[8] L. Eldén, *Numerical linear algebra in data mining*, Acta Numerica, 15 (2006), pp. 327 – 384.

[9] S. Gerschgorin, *Über die abgrenzung der eigenwerte einer matrix*, Izv. Akad. Nauk SSSR Ser. Mat., 7 (1931), pp. 746–754.

[10] F. Golub and C. V. Loan, *Matrix Computations, 3rd Ed.*, The Johns Hopkins University Press, Baltimore, 1996.

[11] L. Guttman, *A necessary and sufficient formula for matric factoring*, Psychometrika, 22 (1957), pp. 79–81.

[12] C. JOHNSON, *Numerical Solutions of Partial Differential Equations by the Finite Element Method*, Cambridge University Press, New York, New Rochelle, Melbourne, Sydney, 1987.

[13] C. KELLY, *Iterative methods for linear and nonlinear equations*, SIAM, Philadelphia, 1995.

[14] P. LEE, G. V. POPESCU, AND S. KIM, *A nonoscillatory second-order time-stepping procedure for reaction-diffusion equations for biological pattern formation*, Complexity, Special Issue: Dynamical Analysis of Biological Systems, 2020, Article ID 5163704 (2020).

[15] ——, *Stable rotational symmetric schemes for nonlinear reaction-diffusion equations*, Computers & Mathematics with Applications, 109 (2022), pp. 191–203.

[16] A. OSTROWSKI, *On the linear iteration procedures for symmetric matrices*, Rend. Mat. e Appl., 14 (1954), pp. 140–163.

[17] J. PETCH, S. DI, AND W. NELSON, *Opening the black box: The promise and limitations of explainable machine learning in cardiology*, Canadian Journal of Cardiology, 38 (2022), pp. 204–213.

[18] F. ROSENBLATT, *The Perceptron, a Perceiving and Recognizing Automaton Project Para*, Report: Cornell Aeronautical Laboratory, Cornell Aeronautical Laboratory, 1957.

[19] G. SALTON, A. WONG, AND C.-S. YANG, *A vector space model for automatic indexing*, Communications of the ACM, 18 (1975), pp. 613–620.

[20] M. G. SHIRANGI, *History matching production data and uncertainty assessment with an efficient TSVD parameterization algorithm*, Journal of Petroleum Science and Engineering, 113 (2014), pp. 54–71.

[21] P. STEIN AND R. L. ROSENBERG, *On the solution of linear simultaneous equations by iteration*, Journal of the London Mathematical Society, s1-23 (1948), pp. 111–118.

[22] O. TAUSSKY, *Bounds for characteristic roots of matrices*, Duke Math. J., 15 (1948), pp. 1043–1044.

[23] L. N. TREFETHEN AND D. BAU, *Numerical Linear Algebra*, SIAM: Society for Industrial and Applied Mathematics, 1997.

[24] R. VARGA, *Matrix Iterative Analysis, 2nd Ed.*, Springer-Verlag, Berlin, Heidelberg, 2000.

[25] S. WALTON, O. HASSAN, AND K. MORGAN, *Reduced order modelling for unsteady fluid flow using proper orthogonal decomposition and radial basis functions*, Applied Mathematical Modelling, 37 (2013), pp. 8930–8945.

[26] D. WATKINS, *Foundations of Matrix Computations, 3rd Ed.*, John Wiley and Sons, New York, Chichester, Brisbane, Toronto, Singapore, 2010.

[27] J. H. M. WEDDERBURN, *Lectures on Matrices*, Amer. Math. Soc., New York, 1934.

[28] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford University Press, 1965. (reprint 1988).

Index