Contents

Chapter 1

Matrix Computation

1.1 Linear Equation Solver

1.1.1 Linear equations

After the application of scientific computing has been modeled and numerically discretized (such as interpolation and fitting of data, differentiation of differential equations, etc.), it all comes down to solving one or several linear equations. In matrix-vector notation, a system of linear algebraic equations has the form

$$A\mathbf{x} = \mathbf{b}. \tag{1.1}$$

where A is a known $m \times n$ matrix

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}.$$

$b = [b_1, b_2, \dots, b_m]^T$ is an m-vector, and $\mathbf{x} = [x_1, x_2, \dots, x_n]^T$ an n-vector.

定义1.1.1 (Singularity)**.** An $n \times n$ matrix A is said to be **nonsingular** if it satisfies any one of the following equivalent conditions:

1. $rank(A) = n$.
2. $|A| \neq 0$ (i.e., the determinant of A is nonzero).
3. A has an inverseA^{-1}.C
4. for any vector $\mathbf{z} \neq 0$,$A\mathbf{z} \neq 0$.

Otherwise, the matrix A is **singular**.

The existence and uniqueness of a solution to a system of linear equations $A\mathbf{x} = \mathbf{b}$ depend on whether the matrix A is singular or nonsingular. In fact, if the matrix A is nonsingular, then its inverse, A^{-1}, exists, and the system $A\mathbf{x} = \mathbf{b}$ always has the unique solution

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

regardless of the value for $\mathbf{b}$.

For a given square matrix A and right-hand-side vector $\mathbf{b}$, the possibilities are summarized as follows:

1. Unique solution: A nonsingular, $\mathbf{b}$ arbitrary
2. Infinitely many solutions: A singular, $\mathbf{b} \in$span(A) (consistent)
3. No solution: A singular, $\mathbf{b} \notin$ span(A) (inconsistent)

1.1.2 Matrix Norms and Conditioning

The measurement of the coefficient matrix is an important indicator of the difficulty of solving linear equations.We need some way to measure the size or magnitude of matrices. A more general definition is possible, but all of the matrix norms we will use are defined in terms of an underlying vector norm.

We Commonly use following definations of matrix norms :

1. 1-norm : $\|A\|_1 = \max\limits_{1\le j\le n} \sum\limits_{i=1}^{m} |a_{ij}|$,
2. ∞-norm : $\|A\|_\infty = \max\limits_{1\le i\le n} \sum\limits_{j=1}^{m} |a_{ij}|$,
3. 2-norm: $\|A\|_2 = \sqrt{\lambda_{max}(A^T A)}$ 。

The matrix norms we have defined satisfy the following important properties, where A and B are any matrices:

1. $\|A\| > 0$, if $A \neq O$.
2. $\|\gamma A\| = |\gamma| \cdot \|A\|$ for any scalar γ.
3. $\|A + B\| \le \|A\| + \|B\|$.
4. $\|AB\| \le \|A\| \cdot \|B\|$.

5. $\|A\mathbf{x}\| \leq \|A\| \cdot \|\mathbf{x}\|$ for any vector $\mathbf{x}$.

What is worth mentioning is that any matrix norms satisfy $\|A\| \geq \rho(A)$, where $\rho(A) = \max|\lambda(A)|$ was called spectral radius, i.e. the eigenvalue with the largest absolute value.

定义1.1.2 (Matrix Condition Number)**.** The *condition number* of a nonsingular square matrix A with respect to a given matrix norm is defined to be

$$cond(A) = \|A\| \cdot \|A^{-1}\|$$

.

The following important properties of the condition number are easily derived from the definition and hold for any norm:

1. For any nonsingular matrix A, $cond(A) \geq 1$
2. For the identity matrix I, $cond(I) = 1$
3. For any nonsingular matrix A and nonzero scalar $\gamma \in \mathbb{R}$, $cond(\gamma A) = cond(A)$
4. For any diagonal matrix$D = diag(d_i), cond(D) = \dfrac{\max\limits_{1\leq i\leq n}|d_i|}{\min\limits_{1\leq i\leq n}|d_i|}$.

1.1.3 Gaussian Elimination

The most classic method to solve this problem is Gauss elimination method, readers may have already studied in linear algebra or advanced algebra courses. The elimination method is a direct method for solving linear equations, and the iterative method is another method that is more commonly used in practical calculations and will be described in the next chapter. Here is just one example, the purpose is to show how to complete the algorithm implementation.The implementation of the numerical algorithm in this book uses MATLAB, but the related algorithm flow is also given, so the reader can implement it in any other computer language according to his own situation.

例1.1.1 (Gauss Elimination)**.** Let the matrix A and the vector $\mathbf{b}$ be the coefficient matrix and the right-side vector of a certain linear equations. Please write the Gauss elimination method for solving the linear equations $A\mathbf{x} = \mathbf{b}$.

```
function x = gauss(A, b);
n = size(A,1);
for k = 1:n−1
    A(k,:) = A(k,:)/A(k,k);
    for j = k+1:n
        factor = −A(j,k)/A(k,k);
```

```
        A(j,k:end) = A(j,k:end) + factor*A(k,k:end);
        b(j) = b(j) + factor*b(k);
    end
end
b(n) = b(n)/A(n,n);
for k = n:-1:2
  b(1:k-1) = b(1:k-1) - A(1:k-1,k)*b(k);
end
x = b;
```

- Time complexity: Consider the number of multiplications and divisions made during algorithm execution.

  Number of multiplications and divisions in one step elimination:

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

  The number of multiplications made in the back-up process is:

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

  In summary, the total number of multiplications and divisions required for Gaussian elimination is:

$$\sum_{k=1}^{n}(n-k+1)^2 + \sum_{k=1}^{n}(n-k) = \frac{1}{3}n(n^2+3n-1) \approx \frac{1}{3}n^3. \tag{1.2}$$

- Space complexity: No additional storage space required, **Original** storage of decomposition results.

- Stability: $|a_{kk}| \approx 0$ can cause floating point overflow, and srability can be measured by *condition number* cond(A).

1.1.4 Extension of Elimination method: LU Factorization

If there are a series of linear equations with the same coefficient matrix (different right-end terms) that require solutions, such as the discrete discretization of the time-dependent partial differential equations under a fixed grid, it is obvious that the elimination process is repeated. using L (lower triangle) U (upper triangle) matrix factorization of coefficient matrix

$$A = LU$$

can eliminate the repeat work. Specifically, let us illustrate with an example: if the Gaussian elimination process for the matrix A is stable, there is

$$A = \begin{bmatrix} 2 & 4 & 4 & 2 \\ 3 & 3 & 12 & 6 \\ 2 & 4 & -1 & 2 \\ 4 & 2 & 1 & 1 \end{bmatrix} \overset{k=1}{\rightarrow} \begin{bmatrix} 2 & 4 & 4 & 2 \\ \frac{3}{2} & -3 & 6 & 3 \\ 1 & 0 & -5 & 0 \\ 2 & -6 & -7 & -3 \end{bmatrix}$$

$$\overset{k=2}{\rightarrow} \begin{bmatrix} 2 & 4 & 4 & 2 \\ \frac{3}{2} & -3 & 6 & 3 \\ 1 & 0 & -5 & 0 \\ 2 & 2 & -19 & -9 \end{bmatrix} \overset{k=3}{\rightarrow} \begin{bmatrix} 2 & 4 & 4 & 2 \\ \frac{3}{2} & -3 & 6 & 3 \\ 1 & 0 & -5 & 0 \\ 2 & 2 & \frac{19}{5} & -9 \end{bmatrix}$$

From the results of the last step, L and U can be separated as:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{3}{2} & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 2 & 2 & \frac{19}{5} & 1 \end{bmatrix} \quad U = \begin{bmatrix} 2 & 4 & 4 & 2 \\ 0 & -3 & 6 & 3 \\ 0 & 0 & -5 & 0 \\ 0 & 0 & 0 & -9 \end{bmatrix}$$

Generally, for the n order non-singular matrix A, you can use the following script to achieve LU decomposition:

```
function [L, U] = lu_primer(A);
n = size(A,1);
for k = 1:n−1
    A(k+1:n,k) = A(k+1:n,k)/A(k,k);
    for i = k+1:n
        A(i,k+1:end) = A(i,k+1:end) + A(i,k)∗A(k,k+1:end);
    end
end
L = tril(A); U = triu(A) + diag(A);
```

When the matrix A is symmetric, obviously the two triangular matrices obtained from the decomposition are also symmetric, i.e. $U = L^T$. This is also called the Doolittle decomposition of a symmetric matrix.

1.1.5 Classical iterative methods

Perhaps the simplest type of iterative method for solving a linear system $A\mathbf{x} = \mathbf{b}$ has the form

$$\mathbf{x}^{(k)} = G\mathbf{x}^{(k-1)} + \mathbf{c}$$

where the matrix G and vector $\mathbf{c}$ are chosen so that a fixed point of the function $g(\mathbf{x}) = G\mathbf{x} + \mathbf{c}$ is a solution to $A\mathbf{x} = \mathbf{b}$. Such a method is said to be *stationary* if G and $\mathbf{c}$ are constant over all iterations.

When $x^{(0)}$ is given, iterative format

$$\mathbf{x}^{(k)} = G\mathbf{x}^{(k-1)} + \mathbf{b}, \quad \forall k = 1, 2, \cdots, n.$$

can be used to get the sequence$\{x^{(k)}\}_{k=0}^{n}$.

The convergence of the sequence is only related to the matrix G, and has nothing to do with the initial value of$\mathbf{x}^{(0)}$!

定理1.1.1 (convergence of iterative method)**.** The iteration scheme is convergent if the spectral radius $\rho(G) < 1$.

Jacobi Method

Writing out this scheme for each individual solution component, we see that, beginning with an initial guess $\mathbf{x}^{(0)}$, the Jacobi method computes the next iterate by solving for each component of $\mathbf{x}$ in terms of the others:

$$x_i^{(k)} = \frac{1}{G_{ii}}\left(b_i - \sum_{j=1}^{i-1} G_{ij}x_j^{(k-1)} - \sum_{j=i+1}^{n} G_{ij}x_j^{(k-1)}\right)$$

Let D be a diagonal matrix with the same diagonal entries as A, and let L and U be the strict lower and upper triangular portions of A, respectively, so that

$$A = L + D + U$$

gives a splitting of A. If A has no zero diagonal entries, so that D is nonsingular, then we obtain the iterative scheme known as the *Jacobi method*:

$$A\mathbf{x} = \mathbf{b}$$

$$(L + D + U)\mathbf{x} = \mathbf{b}$$

$$D\mathbf{x} = -(L + U)\mathbf{x} + \mathbf{b}$$

$$\mathbf{x} = -D^{-1}(L + U)\mathbf{x} + D^{-1}\mathbf{b}$$

```
function [x,n] = jacobi(A,b,x0,TOL,NMAX)
D = diag(diag(A));
L = -tril(A,-1);
U = -triu(A,1);
B = D\(L+U);
f = D\b;
x = B*x0+f;
iter = 1;
while norm(x-x0) >= TOL
    x0 = x;
    x = B*x0 + f
    iter = iter + 1;
    if( iter >= NMAX)
        disp('Do not converge!')
        return;
    end
end
```

The Jacobi method does not always converge, but it is guaranteed to converge under conditions that are often satisfied in practice (e.g., if the matrix is diagonally dominant by rows). Unfortunately, the convergence rate of the Jacobi method is usually unacceptably slow.

Gauss-Seidel Method

One reason for the slow convergence of the Jacobi method is that it does not make use of the latest information available: new component values are used only after the entire sweep has been completed. The *Gauss-Seidel method* remedies this drawback by using each new component of the solution as soon as it has been computed:

$$x_i^{(k)} = \frac{1}{G_{ii}}\Big(b_i - \sum_{j=1}^{i-1} G_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} G_{ij}x_j^{(k-1)}\Big) \tag{1.3}$$

The Gauss-Seidel method does not always converge, but it is guaranteed to converge under conditions that are often satisfied in practice and are somewhat weaker than those for the Jacobi method (e.g., if the matrix is symmetric and positive definite). Although the Gauss-Seidel method converges more rapidly than the Jacobi method, it is often still too slow to be practical.

```
function [x,n] = guaseidel(A,b,x0,TOL,NMAX)
D = diag(diag(A));
L = -tril(A,-1);
U = -triu(A,1);
G = (D-L)\U;
f = (D-L)\b;
x = G*x0 + f;
iter = 1;
while norm(x-x0) >= TOL
    x0 = x;
    x = G*x0 + f;
    iter = iter + 1;
    if( iter >= NMAX)
        disp('Do not converge!');
        return;
    end
end
```

When the matrix G is diagonally dominant, both types of iterations converge. But different examples can be constructed to show that when the Gauss-Seidel method converges, the Jacobi method may not converge; on the other hand, when the Jacobi method converges, the Gauss-Seidel method may not converge! There is a theoretical convergence result when the system matrix is symmetric and positive definite.

定理1.1.2 (Convergence of GS iterative method)**.** In the system $A\mathbf{x} = \mathbf{b}$, if A is a symmetric and positive definite matrix, Gauss-Seidel method converges.

Successive Over-Relaxation

The convergence rate of the Gauss-Seidel method can be accelerated by a technique called *successive over-relaxation (SOR)*, which in effect uses the step to the next Gauss-Seidel iterate as a search direction, but with a fixed search parameter denoted by ω. Starting with $\mathbf{x}^{(\mathbf{k-1})}$:

Step 1

$$\tilde{x}_i^{(k)} = \frac{1}{G_{ii}}\left(b_i - \sum_{j=1}^{i-1} G_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} G_{ij}x_j^{(k-1)}\right)$$

Step 2

$$\begin{aligned} x_i^{(k)} &= x_i^{(k-1)} + \omega(\tilde{x}_i^{(k)} - x_i^{(k-1)}) \\ &= (1-\omega)x_i^{(k-1)} + \omega\tilde{x}_i^{(k)} \end{aligned}$$

SOR method convergence condition judgment:

定理1.1.3. For linear equations $A\mathbf{x} = \mathbf{b}$, the necessary condition for the convergence of the SOR method is $|\omega - 1| < 1$. In particular, if $\omega \in \mathbb{R}$, $0 < \omega < 2$.

定理1.1.4. Known linear equations $A\mathbf{x} = \mathbf{b}$, if

1. A is a symmetric and positive definite matrix,
2. $0 < \omega < 2$,

then, SOR method converges.

定理1.1.5. If A is strictly diagonally dominant and $0 <$ *omega leq*1, the SOR method converges.

A value $\omega > 1$ gives over-relaxation, whereas $\omega < 1$ gives under-relaxation ($\omega = 1$ simply gives the Gauss-Seidel method). We always have $0 < \omega < 2$ (otherwise the method diverges), but choosing a specific value of ω to attain the best possible convergence rate is a difficult problem in general and is the subject of an elaborate theory for special classes of matrices. Empirically, we can use $1.4 < \omega < 1.6$.

```
function [M,x] = SOR(A, b, x0, oemga, NMAX)
D = diag(diag(A));
L = D−tril(A);
U = D−triu(A);
M = (D − omega.∗L)\((1−omega).∗D + omega.∗U);
if vrho(M) >= 1
    error('A is bad for Convergence!');
end
iter = 1;
while iter <= NMAX
    x = M∗x0+(D−w.∗L)\(w.∗b);
    printf("Iter %d : %f", iter,norm(x−x0));
    iter = iter + 1; x0 = x;
end
```

例1.1.2. In the calculation examples listed in the figure above, the accuracy of the calculation requires that the error reaches the seventh position after the decimal point. From the calculation results, we can know that the Gauss-Sediel iteration method calculates 34 steps, while the SOR iteration method only uses 14 steps.

Example

- The linear system $A\mathbf{x} = \mathbf{b}$ given by

$$\begin{aligned} 4x_1 + 3x_2 &= 24 \\ 3x_1 + 4x_2 - x_3 &= 30 \\ - x_2 + 4x_3 &= -24 \end{aligned}$$

  has the solution $(3, 4, -5)^t$.
- Compare the iterations from the Gauss-Seidel method and the SOR method with $\omega = 1.25$ using $\mathbf{x}^{(0)} = (1, 1, 1)^t$ for both methods.

Solution (1/3)

For each $k = 1, 2, \ldots$, the equations for the Gauss-Seidel method are

$$\begin{aligned} x_1^{(k)} &= -0.75x_2^{(k-1)} + 6 \\ x_2^{(k)} &= -0.75x_1^{(k)} + 0.25x_3^{(k-1)} + 7.5 \\ x_3^{(k)} &= 0.25x_2^{(k)} - 6 \end{aligned}$$

and the equations for the SOR method with $\omega = 1.25$ are

$$\begin{aligned} x_1^{(k)} &= -0.25x_1^{(k-1)} - 0.9375x_2^{(k-1)} + 7.5 \\ x_2^{(k)} &= -0.9375x_1^{(k)} - 0.25x_2^{(k-1)} + 0.3125x_3^{(k-1)} + 9.375 \\ x_3^{(k)} &= 0.3125x_2^{(k)} - 0.25x_3^{(k-1)} - 7.5 \end{aligned}$$

Gauss-Seidel Iterations

k	0	1	2	3	$\cdots$	7
$x_1^{(k)}$	1	5.250000	3.1406250	3.0878906		3.0134110
$x_2^{(k)}$	1	3.812500	3.8828125	3.9267578		3.9888241
$x_3^{(k)}$	1	−5.046875	−5.0292969	−5.0183105		−5.0027940

SOR Iterations ($\omega = 1.25$)

k	0	1	2	3	$\cdots$	7
$x_1^{(k)}$	1	6.312500	2.6223145	3.1333027		3.0000498
$x_2^{(k)}$	1	3.5195313	3.9585266	4.0102646		4.0002586
$x_3^{(k)}$	1	−6.6501465	−4.6004238	−5.0966863		−5.0003486

Using the same parameter settings, find the optimal ω value in the SOR iterative solution of the following example:

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

1.2 Linear Least Squares

Writing the linear system in matrix-vector notation, we have

$$A\mathbf{x} = \mathbf{b}$$

where A is an $m \times n$ matrix with $m > n$, $\mathbf{b}$ is an m-vector, and $\mathbf{x}$ an n-vector. In general, with only n parameters in the vector $\mathbf{x}$, we would not expect to be able to reproduce the m-vector $\mathbf{b}$ as a linear combination of the n columns of A. In other words, for an overdetermined system there is usually no solution in the usual sense. We call it *overdetermined system* with more equations than unknowns.

例1.2.1 (Overdetermined System)**.**

$$A\mathbf{x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \mathbf{x_3} \end{bmatrix} \approx \begin{bmatrix} 1237 \\ 1941 \\ 2417 \\ 711 \\ 1177 \\ 475 \end{bmatrix} = \mathbf{b}$$

Existence and Uniqueness:

- If *span*(A) is convex set, $\mathbf{r} = \mathbf{b} - A\mathbf{x}$ is convex mapping, then there exists unique solution;
- The solution to an $m \times n$ least squares problem $Ax \approx b$ is unique if, and only if, A has full column rank, i.e., rank(A) = n;
- If rank(A) < n, then A is said to be *rank-deficient*, and though a solution of the corresponding least squares problem must still exist, it cannot be unique in this case.

1.2.1 Normal Equations

We wish to minimize the squared Euclidean norm of the residual vector$r = b - Ax$. Denoting this objective function by ϕ: $\mathbb{R}^n \to \mathbb{R}$, we have

$$\phi(x) = \|r\|_2^2 = r^T r = (b - Ax)^T (b - Ax) = b^T b - 2x^T A^T b + x^T A^T A x$$

Let gradient vector $\nabla\phi(x) = 0$, we have

$$0 = \nabla\phi(x) = 2A^T Ax - 2A^T b$$

so any minimizer x for ϕ must satisfy the $n \times n$ symmetric linear system

$$(A^T A)\mathbf{x} = A^T \mathbf{b}$$

which is commonly known as the system of *normal equations*.Hessian matrix $A^T A$ is symmetric and positive definite.

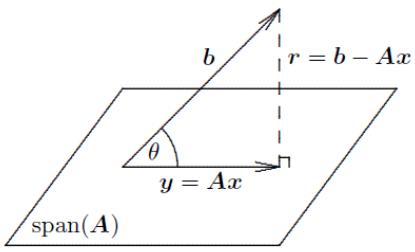
例1.2.2 (Normal Equations)**.** The system of normal equations for the linear least squares problem in last example is the symmetric positive definite system

$$A^T Ax = \begin{bmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -651 \\ 2177 \\ 4069 \end{bmatrix} = A^T b.$$

whose solution $x = [-651, 2177, 4069]^T$, achieves the minimum possible sum of squares, $\|r\|_2^2 = 35$.

1.2.2 Orthogonality and Orthogonal Projectors

For a least squares problem $Ax = b$, with $m > n$, the m-vector b generally does not lie in span(A), a subspace of dimension at most n. The vector $y = Ax \in$span(A) closest to b in the Euclidean norm occurs when the residual vector $r = b` Ax$ is orthogonal to span(A), see in following figure.



Thus, for the least squares solution x, the residual vector $r = b - Ax$ must be orthogonal to each column of A, and hence we must have

$$\mathbf{0} = A^T r = A^T (b - Ax) \qquad \to \qquad A^T Ax = A^T b,$$

which is the same system of normal equations we derived earlier using calculus.

1.2.3 Sensitivity and Conditioning

We turn now to the sensitivity and conditioning of linear least squares problems. First, we must extend the notion of matrix condition number to include rectangular matrices. The definition of condition number for a square matrix makes use of the matrix inverse. A nonsquare matrix A does not have an inverse in the conventional sense, but it is possible to define a pseudoinverse, denoted by A^+, that behaves like an inverse in many respects. We will later see a more general definition that applies to any matrix, but for now we consider only matrices A with full column rank, in which case A^TA is nonsingular and we define the pseudoinverse of A to be

$$A^+ = \left(A^TA\right)^{-1} A^T.$$

Trivially, we see that $A^+A = I$, and $\mathrm{P} = AA^+$ is an orthogonal projector onto $\mathrm{span}(A)$, so that the solution to the least squares problem $Ax = b$ is given by

$$x = A^+b.$$

We now define the condition number of an $m \times n$ matrix with $\mathrm{rank}(A) = n$ to be

$$\mathbf{cond}(A) = \|A\|_2 \cdot \left\|A^+\right\|_2.$$

By convention, $\mathrm{cond}(A) = \inf$ if $\mathrm{rank}(A) < n$.

To solve $A^TA\mathbf{x} = A^T\mathbf{b}$, for simplicity, we will consider perturbations in b and A separately. For a perturbed right-hand-side vector $b + \delta b$, the perturbed solution is given by the normal equations

$$A^TA(\mathbf{x} + \delta\mathbf{x}) = A^T(\mathbf{b} + \delta\mathbf{b})$$

Because $A^TAx = A^Tb$, we then have

$$A^TA\delta x = A^T\delta b$$

so that

$$\delta x = \left(A^TA\right)^{-1} A^T\delta b = A^+\delta b$$

Taking norms, we obtain

$$\|\delta x\|_2 \leq \left\|A^+\right\|_2 \cdot \|\delta b\|_2.$$

Dividing both sides by$\|x\|_2$, we obtain the bound

$$\frac{\|\delta\mathbf{x}\|_2}{\|\mathbf{x}\|_2} \leq \cdots \leq cond(A) \left(\frac{\|\mathbf{b}\|_2}{\|A\mathbf{x}\|_2}\right) \left(\frac{\|\delta\mathbf{b}\|_2}{\|\mathbf{b}\|_2}\right)$$

Thus, the condition number for the least squares solution x with respect to perturbations in b depends on $\mathrm{cond}(A)$ and also on the angle θ between b and Ax.

例1.2.3 (Sensitivity and Conditioning)**.** We again illustrate these concepts by continuing with previous example. The pseudoinverse is given by

$$A^{+}=\left(A^{T}A\right)^{-1}A^{T}=\frac{1}{4}\begin{bmatrix}2&1&1&-1&-1&0\\1&2&1&1&0&-1\\1&1&2&0&1&1\end{bmatrix}.$$

The matrix norms can be computed to obtain

$$\|A\|_2=2,\left\|A^{+}\right\|_2=1.$$

so that

$$\mathbf{cond}(A)=\|A\|_2\cdot\left\|A^{+}\right\|_2=2.$$

From the ratio

$$\cos(\theta)=\frac{\|\mathbf{b}\|_2}{\|A\mathbf{x}\|_2}\approx 0.99999868$$

we see that the angle θ between b and y is about 0.001625, which is very tiny, as expected for a problem with a very close fit to the data. From the small condition number and small angle θ, we conclude that this particular least squares problem is well-conditioned.

For a perturbed matrix $A+E$, the perturbed solution is given by the normal equations

$$(A+E)^{T}(A+E)(x+\delta x)=(A+E)^{T}b.$$

Noting that $A^{T}Ax=A^{T}b$, dropping second-order terms (i.e., products of small perturbations), and rearranging, we then have

$$\begin{aligned}A^{T}A\delta x&\approx E^{T}b-E^{T}Ax-A^{T}Ex\\&=E^{T}(b-Ax)-A^{T}Ex\\&=E^{T}r-A^{T}Ex,\end{aligned}$$

so that

$$\delta x\approx\left(A^{T}A\right)^{-1}E^{T}r-A^{+}Ex.$$

Taking norms, we obtain

$$\|\delta x\|_2\leq\left\|\left(A^{T}A\right)^{-1}\right\|_2\cdot\|E\|_2\cdot\|r\|_2+\left\|A^{+}\right\|_2\cdot\|E\|_2\cdot\|x\|_2.$$

Dividing both sides by $\|x\|_2$ and using the fact that $\|A\|_2^2\cdot\left\|(A^{T}A)^{-1}\right\|_2=[cond\,(A)]^2$, we obtain the bound

$$\begin{aligned}\frac{\|\delta x\|_2}{\|x\|_2} &\leq \left\|\left(A^TA\right)^{-1}\right\|_2 \cdot \|E\|_2 \cdot \frac{\|r\|_2}{\|x\|_2} + \left\|A^+\right\|_2 \cdot \|E\|_2 \\ &= \left[cond\,(A)\right]^2 \frac{\|E\|_2}{\|A\|_2} \frac{\|r\|_2}{\|A\|_2 \cdot \|x\|_2} + cond(A)\frac{\|E\|_2}{\|A\|_2} \\ &\leq \left(\left[cond\,(A)\right]^2 \frac{\|r\|_2}{\|Ax\|_2} + cond(A)\right)\frac{\|E\|_2}{\|A\|_2} \\ &= \left(\left[cond\,(A)\right]^2 \tan(\theta) + cond(A)\right)\frac{\|E\|_2}{\|A\|_2}.\end{aligned}$$

Thus, the condition number for the least squares solution x with respect to perturbations in A depends on cond(A) and also on the angle θ between b and Ax.

例1.2.4 (Condition-Squaring Effect)**.** Consider the matrix and perturbation

$$A = \begin{bmatrix} 1 & 1 \\ \epsilon & -\epsilon \\ 0 & 0 \end{bmatrix}, E = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ -\epsilon & \epsilon \end{bmatrix},$$

where $\epsilon \ll 1$, say around $\sqrt{\epsilon_{mach}}$, for which we have

$$cond(A) = 1/\epsilon, \|E\|_2 / \|A\|_2 = \epsilon.$$

For the right-hand-side vector $b = [1, 0, \epsilon]^T$, we have $\|\delta x\|_2 / \|x\|_2 = 0.5$, so the relative perturbation in the solution is about equal to cond(A) times the relative perturbation in A. There is no condition-squaring effect for this right-hand side because the residual is small and $\tan(\theta) \approx \epsilon$, effectively suppressing the condition-squared term in the perturbation bound.

For the right-hand-side vector $b = [1, 0, 1]^T$, on the other hand, we have $\|\delta x\|_2 / \|x\|_2 = 0.5/\epsilon$, so the relative perturbation in the solution is about equal to $[\text{cond}(A)]^2$ times the relative perturbation in A. For this right-hand side, the norm of the residual is about 1 and $\tan(\theta) \approx 1$, so that the condition-squared term in the perturbation bound is not suppressed, and the solution is highly sensitive.

1.2.4 Problem Transformations

Considering the numerical stability, the least squares problem can also be transformed into *Augmented System* or the *Triangular Least Squares Problems*.

Augmented System

Another way to transform a least squares problem into a square linear system is to embed it in a larger system. The definition of the residual vector r, together with the requirement

that the residual be orthogonal to the columns of A, gives the system of two equations

$$\begin{aligned} \mathbf{r} + A\mathbf{x} &= \mathbf{b} \\ A^T\mathbf{r} &= \mathbf{0} \end{aligned}$$

which can be written in matrix form as the $(m+n) \times (m+n)$ *augmented system*

$$\begin{bmatrix} I & A \\ A^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}$$

whose solution yields both the desired solution x and the residual r at that solution.The relative scales of r and x are arbitrary, so we introduce a scaling parameter α for the residual, giving the new system

$$\begin{bmatrix} \alpha I & A \\ A^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{r}/\alpha \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}$$

The parameter α controls the relative weights of the entries in the two subsystems in choosing pivots from either. A reasonable rule of thumb is to take

$$\alpha = \max_{i,j} |a_{i,j}|/1000.$$

but some experimentation may be required to determine the best value.

A straightforward implementation of this method can be prohibitive in cost (proportional to $(m+n)^3$), so the special structure of the augmented matrix must be carefully exploited. For example, the augmented system method is used effectively in MATLAB for large sparse linear least squares problems.

Triangular Least Squares Problems

Now that we have a family of transformations that preserve the least squares solution, we next need a suitable target for simplifying a least squares problem so that it becomes easy to solve. As we did with square linear systems, let us consider least squares problems having an upper triangular matrix. In the overdetermined case, $m > n$, such a problem has the form

$$\begin{bmatrix} R \\ O \end{bmatrix} x \approx \begin{bmatrix} c_1 \\ c_2 \end{bmatrix},$$

where R is an nn upper triangular matrix, and we have partitioned the right-hand-side vector c similarly. The least squares residual is then given by

$$\|r\|_2^2 = \|c_1 - Rx\|_2^2 + \|c_2\|_2^2 .$$

Because it is independent of x, we have no control over the second term, $\|c_2\|_2^2$, in the foregoing sum, but the first term can be forced to be zero by choosing x to satisfy the triangular system

$$Rx = c_1,$$

which can be solved for x by back-substitution. We have therefore found the least squares solution x and can also conclude that the minimum sum of squares is

$$\|r\|_2^2 = \|c_2\|_2^2 .$$

QR Factorization

Orthogonal transformation to triangular form is accomplished by the QR *factorization*, which, for an $m \times n$ matrix A with $m > n$, has the form

$$A = Q \begin{bmatrix} R \\ O \end{bmatrix},$$

If we partition Q as $Q = [Q_1, Q_2]$, where Q_1 contains the first n columns and Q_2 contains the remaining $m - n$ columns of Q, then we have

$$A = Q \begin{bmatrix} R \\ O \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ O \end{bmatrix} = Q_1 R.$$

In the next section, we will see how to compute the QR factorization.

A factorization of the form $A = Q_1 R$, with Q_1 having orthonormal columns and the same dimensions as A, and R square and upper triangular, is sometimes called the *reduced*, or "economy size" QR factorization of A. If A has full column rank, so that R is nonsingular, then the columns of Q_1 form an orthonormal basis for $\text{span}(A)$ and the columns of Q_2 form an orthonormal basis for its orthogonal complement, $\text{span}(A)^\perp$, which is the same as the *null space* of A^T (i.e., $z \in \mathbb{R}^m : A^T z = 0$). Such orthonormal bases are useful not only in least squares computations, but also in eigenvalue computations, optimization, and many other problems .

1.2.5 Orthogonalization Methods

Our approach to computing the QR factorization of a matrix will be similar to LU factorization using Gaussian elimination in that we will introduce zeros successively into the matrix A, eventually reaching upper triangular form, but we will use orthogonal transformations rather than elementary elimination matrices so that the Euclidean norm will be preserved. A number of such orthogonalization methods are commonly used, including

1. Householder transformations (elementary reflectors)
2. Givens transformations (plane rotations)
3. Gram-Schmidt orthogonalization

Householder Transformations

We seek an orthogonal transformation that annihilates desired components of a given vector. One way to accomplish this is a *Householder transformation*, or *elementary reflector*, which is a matrix of the form

$$H = I - \frac{2}{\mathbf{v}^T\mathbf{v}}\mathbf{v}\mathbf{v}^T$$

where v is a nonzero vector. From the definition, we see that $H = H^T = H^{`1}$, so that H is both orthogonal and symmetric. Given a vector a, we wish to choose the vector v so that all the components of a except the first are annihilated, i.e.,

$$Ha = \begin{bmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \alpha \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \alpha e_1.$$

Using the formula for H, we have

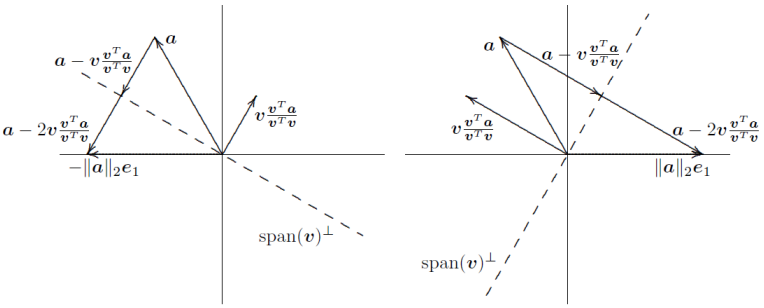
$$\alpha e_1 = Ha \left(I - 2\frac{vv^T}{v^Tv} \right) a = a - 2v\frac{v^Ta}{v^Tv},$$

and hence

$$v = (a - \alpha e_1)\,\frac{v^Tv}{2v^Ta}.$$

But the scalar factor is irrelevant in determining v, since it divides out in the formula for H anyway, so we can take

$$v = a - \alpha e_1.$$



To preserve the norm, we must have $\alpha = \pm \|a\|_2$, and the sign should be chosen to avoid cancellation (i.e.,$\alpha = -\ \mathrm{sign}(a)_2$).

$$\mathbf{v} = \mathbf{a} - \|\mathbf{a}\|_2\mathbf{e}_1$$

例1.2.5 (Householder Transformation.)**.** To illustrate the construction just described, we determine a Householder transformation that annihilates all but the first component of the vector

$$a = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix}.$$

Following the foregoing recipe, we choose the vector

$$\mathbf{v} = \mathbf{a} - \alpha\mathbf{e}_1 = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} - (-3)\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \\ 2 \end{bmatrix}$$

To confirm that the Householder transformation performs as expected, we compute

$$Ha = a - 2\frac{v^T a}{v^T v}v = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} - 2\frac{15}{30}\begin{bmatrix} 5 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix},$$

which shows that the zero pattern of the result is correct and that the 2-norm is preserved. Note that there is no need to form the matrix H explicitly, as the vector v is all we need to apply H to any vector.

思考：一次H变换完成一个列向量单位化，那么，如何利用这个算法实现矩阵的QR分解?

Givens Rotations

Householder transformations introduce many zeros in a column at once. Although generally good for effciency, this approach can be too heavy-handed when greater selectivity is needed in introducing zeros. For this reason, in some situations it is better to use Givens rotations, which introduce zeros one at a time.

We seek an orthogonal matrix that annihilates a single component of a given vector. One way to accomplish this is a *plane rotation*, often called a *Givens rotation* in the context of QR factorization, which has the form

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

Given a 2-vector $a = \begin{bmatrix} a_1 & a_2 \end{bmatrix}^T$, G can rotate a like

$$Ga = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sqrt{a_1^2 + a_2^2} \\ 0 \end{bmatrix}.$$

where

$$\cos(\theta) = \frac{a_1}{\sqrt{a_1^2 + a_2^2}}, \quad \sin(\theta) = \frac{a_2}{\sqrt{a_1^2 + a_2^2}}$$

The angle of rotation need not be determined explicitly, as only its sine and cosine are needed.

例1.2.6 (Givens Rotation.)**.** To illustrate the construction just described, we determine a Givens rotation that annihilates the second component of the vector

$$a = \begin{bmatrix} 4 \\ 3 \end{bmatrix}.$$

For this problem, we can safely compute the cosine and sine directly, obtaining

$$\cos(\theta) = \frac{a_1}{\sqrt{a_1^2 + a_2^2}} = \frac{4}{5} = 0.8, \sin(\theta) = \frac{a_2}{\sqrt{a_1^2 + a_2^2}} = \frac{3}{5} = 0.6.$$

Thus, the rotation is given by

$$G = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} = \begin{bmatrix} 0.8 & 0.6 \\ -0.6 & 0.8 \end{bmatrix}.$$

To confirm that the rotation performs as expected, we compute

$$Ga = \begin{bmatrix} 0.8 & 0.6 \\ -0.6 & 0.8 \end{bmatrix} \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix},$$

which shows that the zero pattern of the result is correct and that the 2-norm is preserved.

Gram-Schmidt Orthogonalization

Gram-Schmidt orthogonalization is a general technique, if there are $\mathbf{a_1}, \mathbf{a_2}, \cdots \mathbf{a_n}$,, we can use following algorithms.

```
for k = 1 → n do
    q_k = a_k
    for j = 1 → k − 1 do
        r_jk = q_j^T a_k
        q_k = q_k − r_jk q_j
    end
    r_kk = ‖q_k‖_2
    if r_kk == 0 then
        quit
    else
        q_k = q_k / r_kk
    end
end
```

```
// Gram-Schmidt
for k = 1 → n do
    r_kk = ‖a_k‖_2
    if r_kk == 0 then
        quit
    else
        q_k = a_k / r_kk
    end
    for j = k + 1 → n do
        r_kj = q_k^T a_j
        a_j = a_j − r_kj q_k
    end
end
```

```
for i → n do
    q_i = a_i
    for j = 1 → i do
        r_ji = q_j^T a_i
        q_i = q_i − r_ji q_j
    end
    r_ii = ‖q_i‖_2
    if r_ii == 0 then
        quit
    else
        q_i = q_i / r_ii
    end
end
```

The above two implementations are equivalent, with only a few differences in form: passive reduction and active cancellation. In addition, the improved version can choose the pivot to increase stability! We can use Gram-Schmidt implements QR decomposition as the third one in the above chart.

例1.2.7 (Gram-Schmidt QR Factorization)**.** We illustrate modified Gram-Schmidt orthogonalization by using it to solve the least squares problem metended before. Normalizing the first column of A, we compute

$$r_{11} = \|\mathbf{a}_1\|_2 = 1.7321, \mathbf{q}_1 = \mathbf{a}_1/r_{11} = \begin{bmatrix} 0.5774 \\ 0 \\ 0 \\ -0.5774 \\ -0.5774 \\ 0 \end{bmatrix}.$$

Orthogonalizing the first column against the subsequent columns, we obtain

$$r_{12} = \mathbf{q}_1^T \mathbf{a}_2 = -0.5774, r_{123} = \mathbf{q}_1^T \mathbf{a}_3 = -0.5774.$$

Subtracting these multiples of $\mathbf{q}_1$ from the second and third columns, respectively, and replacing the first column with $\mathbf{q}_1$, we obtain the transformed matrix

$$\begin{bmatrix} 0.5774 & 0.3333 & 0.3333 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -0.5774 & 0.6667 & -0.3333 \\ -0.5774 & -0.3333 & 0.6667 \\ 0 & -1 & 1 \end{bmatrix}.$$

Normalizing the second column, we compute

$$r_{22} = \|\mathbf{a}_2\|_2 = 1.6330, \mathbf{q}_2 = \mathbf{a}_2/r_{22} = \begin{bmatrix} 0.2041 \\ 0.6124 \\ 0 \\ 0.4082 \\ -0.2041 \\ -0.6124 \end{bmatrix}.$$

Orthogonalizing the second column against the third column, we obtain

$$r_{23} = \mathbf{q}_2^T \mathbf{a}_3 = -0.8165.$$

Subtracting this multiple of $\mathbf{q}_2$ from the third column and replacing the second column with $\mathbf{q}_2$, we obtain the transformed matrix

$$\begin{bmatrix} 0.5774 & 0.2041 & 0.5 \\ 0 & 0.6124 & 0.5 \\ 0 & 0 & 1 \\ -0.5774 & -0.4082 & 0 \\ -0.5774 & -0.2041 & 0.5 \\ 0 & -0.6124 & 0.5 \end{bmatrix}.$$

Finally, we normalize the third column

$$r_{33} = \|\mathbf{a}_3\|_2 = 1.4142, \mathbf{q}_3 = \mathbf{a}_3/r_{33} = \begin{bmatrix} 0.3536 \\ 0.3536 \\ 0.7071 \\ 0 \\ 0.3536 \\ 0.3536 \end{bmatrix}.$$

Replacing the third column with $\mathbf{q}_3$, we have obtained the reduced QR factorization

$$A = \begin{bmatrix} 0.5774 & 0.2041 & 0.3536 \\ 0 & 0.6124 & 0.3536 \\ 0 & 0 & 0.7071 \\ -0.5774 & 0.4082 & 0 \\ -0.5774 & -0.2041 & 0.3536 \\ 0 & -0.6124 & 0.3536 \end{bmatrix} \begin{bmatrix} 1.7321 & -0.5774 & -0.5774 \\ 0 & 1.6330 & -0.8165 \\ 0 & 0 & 1.4142 \end{bmatrix} = Q_1 R.$$

For this well-conditioned problem, we can safely compute the transformed right-hand side explicitly, obtaining

$$Q_1^T \mathbf{b} = \begin{bmatrix} -376 \\ 1200 \\ 3417 \end{bmatrix} = \mathbf{c}_1.$$

We can now solve the upper triangular system $R\mathbf{x} = \mathbf{c}_1$ by back-substitution to obtain $\mathbf{x} = [1236, 1943, 2416]^T$.

例1.2.8 (QR decomposition example)**.** It is not difficult to verify that

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{3}} \\ 0 & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} \frac{2}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{3}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ 0 & 0 & \frac{2}{\sqrt{3}} \end{bmatrix} := QR$$

1.2.6 Singular Value Decomposition

定理1.2.1. The *singular value decomposition (SVD)* of an $m \times n$ matrix A has the form

$$A = U\Sigma V^T$$

where U is an $m \times m$ orthogonal matrix($U^H U = I$), V is an $n \times n$ orthogonal matrix($V^H V = I$), and $\Sigma = diag(\sigma_1, \sigma_2, \cdots, \sigma_n)$ is an $m \times n$ diagonal matrix, with

$$\sigma_{ij} = \begin{cases} 0, & \text{for} i \neq j \\ \sigma \leq 0 & \text{i=j} \end{cases}.$$

The diagonal entries σ_i are called the *singular values* of A and are usually ordered so that $\sigma_{i-1} \leq \sigma_i, i = 2, \cdots,$ minm, n. The columns $\mathbf{u}_i$ of U and $\mathbf{v}_i$ of V are the corresponding left and right singular vectors.

例1.2.9 (Singular Value Decomposition)**.**

$$A = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 2^{-\frac{1}{2}} & -2^{-\frac{1}{2}} \\ 2^{-\frac{1}{2}} & 2^{-\frac{1}{2}} \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 2^{-\frac{1}{2}} & -2^{-\frac{1}{2}} \\ 2^{-\frac{1}{2}} & 2^{-\frac{1}{2}} \end{bmatrix} := U\Sigma V^T$$

Wen minimizing $\|A\mathbf{x} - \mathbf{b}\|_2$, if the A rank deficit or "close to" the rank deficit, it is often referred to as the least squares problem of rank deficit. SVD can calculate its low-dimensional approximate solution.

SVD has a wide range of applications in modern scientific applications. In solving ill-conditioned problems such as digital image restoration, the solution of some integral equations, or extracting signals from noised data, regularization is commonly used to solve the rank deficit problem.

Computing the Singular Value Decomposition

Singular values and vectors are intimately related to eigenvalues and eigenvectors: the singular values of $\boldsymbol{A}$ are the nonnegative square roots of the eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$, and the columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal eigenvectors of $\boldsymbol{A}\boldsymbol{A}^T$ and $\boldsymbol{A}^T\boldsymbol{A}$, respectively. Stable algorithms for computing the SVD work directly with $\boldsymbol{A}$, however, without forming $\boldsymbol{A}\boldsymbol{A}^T$ or $\boldsymbol{A}^T\boldsymbol{A}$, thereby avoiding any loss of information associated with forming these matrix products explicitly.

例1.2.10 (Computing the Singular Value Decomposition)**.** Given a matrix:

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

First step: computing $\boldsymbol{U}$.

Computing $AA^T = \begin{bmatrix} 2 & 2 & 0 \\ 2 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, we obtain the eigenvalues 4, 0, 0 and the corresponding eigenvectors $\left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right]^T$, $\left[-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right]^T$, $[0, 0, 1]^T$, so $U = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

Second step: computing $\boldsymbol{V}$.

Computing $A^TA = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$, we obtain the eigenvalues 4, 0, and the corresponding eigenvectors $\left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]^T$, $\left[-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]^T$,so $V = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$.

Third step:

Computing $\Sigma_{mn} = \begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix}$,where $\Sigma_1 = \operatorname{diag}(\sigma_1, \sigma_1, \ldots, \sigma_r)$ is the value of the non-zero eigenvalues obtained in the first or second step from the largest to the smallest, and here $\Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$. Finally, we can get the singular value decomposition of $\boldsymbol{A}$:

$$A = U\Sigma V^T = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}^T = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}.$$

Image Compression

Singular value decomposition has important applications in image processing. Assuming that an image has $n \times n$ pixels, if these n^2 data are transmitted together, it will often appear to have a large amount of data. Therefore, we hope to be able to transfer some other relatively small data and use the data at the receiving end to reconstruct the original image.

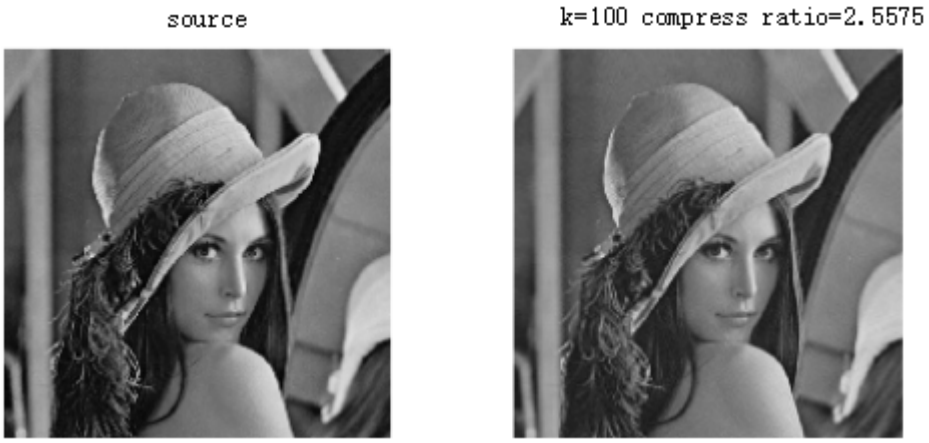
Suppose singular value decomposition is performed on matrix $\boldsymbol{A}$, we get $A = U\Sigma V^T$, where the singular values are arranged in order from large to small. If k large singular values are selected from these and the corresponding left and right singular vectors are used to approximate the original image, $k(2n + 1)$ values can be used in total to replace the original $n \times n$ image data.

Image compression ratio:

$$\rho = \frac{n^2}{k(2n + 1)}.$$

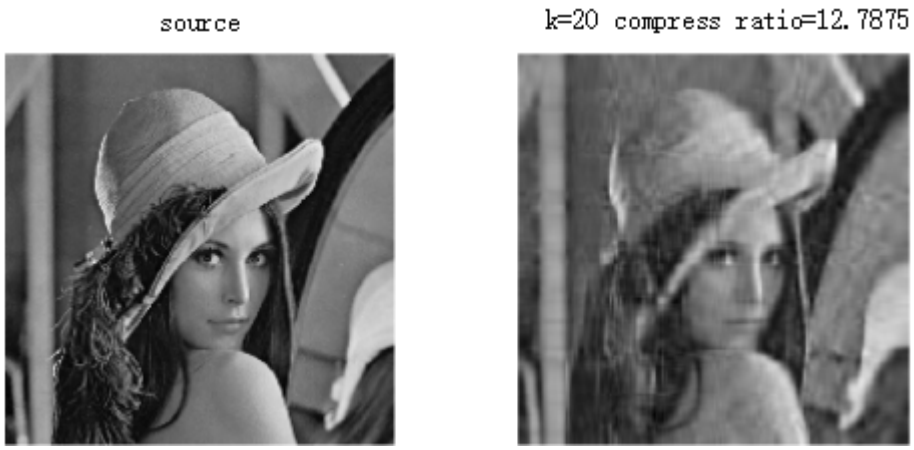
例1.2.11 (Image Compression with SVD)**.** We use image "lina" as an example:

k = 100: It can be seen that the singular values of the first 100 images are retained,



and most of the information of the image can be basically retained. The compression rate at this time is 2.5575. If you want to increase the compression ratio, keep less.

k = 25:



Code show as below:

```
clear all;close all; clc
f = imread('lena.jpg');
[m,n] = size(f);

k = 100;
f = double(f);
[u,s,v] = svd (f);
```

```
s = diag(s);
smax = max(s);smin = min(s);
s1=s;s1(k:end) = 0;
s1 = diag(s1);
g = u*s1*v';
g = uint8(g);
compressratio = n^2/(k*(2*n+1));

subplot(1,2,1),imshow(mat2gray(f));title('source')
subplot(1,2,2),imshow(g); title(['compress_ratio',num2str(compressratio)])
figure,plot(s,'.','Color','k')
```

1.3 Eigenvalue Problem

The numerical method of calculating one or more eigenvalues of a matrix is an important concern in scientific computing.

定义1.3.1 (eigenvector)**.** Given an $n \times n$ matrix A representing a linear transformation on an n-dimensional vector space, we wish to find a nonzero vector $\mathbf{x}$ and a scalar λ such that

$$A\mathbf{x} = \lambda\mathbf{x}$$

Such a scalar λ is called an eigenvalue, and $\mathbf{x}$ is a corresponding *eigenvector*.In addition to the *right* eigenvector just defined, we could also define a nonzero left eigenvector $\mathbf{y}$ such that $\mathbf{y}^T A = \lambda\mathbf{y}^T$. A *left* eigenvector of A is a right eigenvector of A^T , however, so for computational purposes we will consider only right eigenvectors

The set of all the eigenvalues of a matrix A, denoted by $\lambda(A)$, is called the *spectrum* of A. The maximum modulus of the eigenvalues, $\max|\lambda| : \lambda \in \lambda(A)$, is called the *spectral radius* of A, denoted by $\rho(A)$.

定理1.3.1 (Gershgorin's Theorem)**.** The eigenvalues of an $n \times n$ matrix A are all contained within the union of n disks, with the kth disk centered at $a_k k$ and having radius $\sum_{j\neq k}|a_{kj}|$.

proof: let λ be any eigenvalue, with corresponding eigenvector $\mathbf{x}$, normalized so that $\|\mathbf{x}\|_{\text{inf}} = 1$. Let $\mathbf{x}_k$ be an entry of $\mathbf{x}$ such that $|\mathbf{x}_k| = 1$ (at least one component has magnitude 1, by definition of the inf-norm). Because $A\mathbf{x} = \lambda\mathbf{x}$, we have

$$(\lambda - a_{kk})\, x_k = \sum_{j\neq k} a_{kj} x_j$$

so that

$$|\lambda - a_{kk}| \leq \sum_{j\neq k} |a_{kj}| \cdot |x_j| \leq \sum_{j\neq k} |a_{kj}|\,.$$

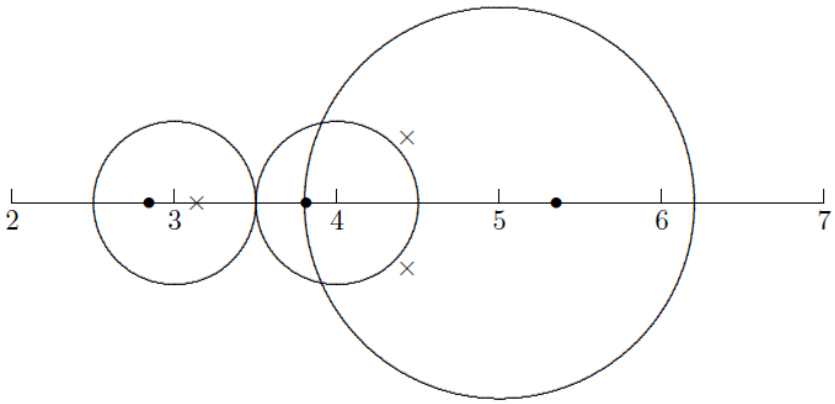
例1.3.1 (Gershgorin Disks.)**.** The Gershgorin disks for the real matrix

$$A_1 = \begin{bmatrix} 4.0 & -0.5 & 0.0 \\ 0.6 & 5.0 & -0.6 \\ 0.0 & 0.5 & 3.0 \end{bmatrix}$$

are plotted in the complex plane. The three eigenvalues of this matrix, indicated by $\times$ in the figure, lie within the union of the disks. Note that two of the eigenvalues are complex conjugates. The matrix

$$A_2 = \begin{bmatrix} 4.0 & 0.5 & 0.0 \\ 0.6 & 5.0 & 0.6 \\ 0.0 & 0.5 & 3.0 \end{bmatrix}$$

has the same Gershgorin disks, but all three of its eigenvalues, indicated by $\cdot$ in the figure, are real and hence lie on the real axis of the complex plane.



1.3.1 Sensitivity and Conditioning

We now consider the sensitivity of an individual eigenvalue of a (possibly defective) matrix A. Let $\mathbf{x}$ and $\mathbf{y}$ be right and left eigenvectors, respectively, corresponding to a simple eigenvalue λ of A, and consider the perturbed eigenvalue problem

$$(A + E)(\mathbf{x} + \delta\mathbf{x}) = (\lambda + \delta\lambda)(\mathbf{x} + \delta\mathbf{x}).$$

忽略二阶项，再化简、左乘左特征向量$\mathbf{y}^H$,可得：Expanding both sides, dropping second-order terms (i.e., products of small perturbations, such as $E\delta\mathbf{x}$), and using the fact that $A\mathbf{x} = \lambda\mathbf{x}$, we obtain the approximation

$$\boldsymbol{A}\delta\boldsymbol{x} + \boldsymbol{E}\boldsymbol{x} \approx \delta\lambda\boldsymbol{x} + \lambda\delta\boldsymbol{x}.$$

Premultiplying both sides by $\boldsymbol{y}^H$, we obtain

$$\boldsymbol{y}^H \boldsymbol{A}\Delta\boldsymbol{x} + \boldsymbol{y}^H \boldsymbol{E}\boldsymbol{x} \approx \Delta\lambda \boldsymbol{y}^H \boldsymbol{x} + \lambda \boldsymbol{y}^H \Delta\boldsymbol{x}.$$

Because $\boldsymbol{y}$ is a left eigenvector, $\boldsymbol{y}^H \boldsymbol{A} = \lambda \boldsymbol{y}^H$, and using this fact yields

$$\boldsymbol{y}^H \boldsymbol{E}\boldsymbol{x} = \delta\lambda \boldsymbol{y}^H \boldsymbol{x}.$$

By assumption λ is a simple eigenvalue, so$\boldsymbol{y}^H \boldsymbol{x} \neq 0$ and hence we can divide by$\boldsymbol{y}^H \boldsymbol{x}$ to obtain

$$\delta\lambda \approx \frac{\mathbf{y}^H E\mathbf{x}}{\mathbf{y}^H \mathbf{x}},$$

which, upon taking norms yields the bound

$$|\delta\lambda| \leq \frac{\|\mathbf{y}\|_2 \|\mathbf{x}\|_2}{|\mathbf{y}^H \mathbf{x}|} \|E\|_2 = \frac{1}{\cos\theta} \|E\|_2$$

where θ is the angle between $\boldsymbol{x}$ and $\boldsymbol{y}$. Thus, the absolute condition number of a simple eigenvalue is given by the reciprocal of the cosine of the angle between its corresponding right and left eigenvectors. We can conclude that a simple eigenvalue is sensitive if its right and left eigenvectors are nearly orthogonal, so that $\cos(\theta) \approx 0$, but is insensitive if the angle between its right and left eigenvectors is small, so that $\cos(\theta) \approx 1$. In particular, the eigenvalues of real symmetric and complex Hermitian matrices are always well-conditioned, since the right and left eigenvectors are the same, so that $\cos(\theta) = 1$.

1.3.2 Computing Eigenvalues and Eigenvectors

Solving eigenvalues can be roughly divided into direct method and iterative method.

Characteristic Polynomial

The direct method of calculating eigenvalues can start from its definition and transform the problem into an equation to find the root problem through its relationship with the zero point of the characteristic polynomial:

Eigenvalue problem $A\mathbf{x} = \lambda\mathbf{x}$ is equivalent to solving an equation

$$(A - \lambda\mathbf{I})\mathbf{x} = \mathbf{0}$$

The necessary and sufficient condition for $\mathbf{x}$ to have a non-zero solution is that the coefficient matrix is singular, that is, the zero points of the characteristic polynomial

$$det(A - \lambda I) = 0$$

are the eigenvalue of A.

例1.3.2 (Characteristic Polynomial)**.**

$$\boldsymbol{A} = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}.$$

The characteristic polynomial of matrix $\boldsymbol{A}$ is

$$\begin{aligned} \det\left(\begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) &= \det\left(\begin{bmatrix} 3-\lambda & 1 \\ 1 & 3-\lambda \end{bmatrix}\right) \\ &= (3-\lambda)(3-\lambda) - 1 \times 1 \\ &= \lambda^2 - 6\lambda + 8 \\ &= (\lambda - 4)(\lambda - 2) = 0 \end{aligned}$$

so eigenvalue are $\lambda_1 = 4 and \lambda_2 = 2$.

Iterative methods are important methods for solving matrix eigenvalues. Iterative methods are usually used in the case of sparse matrices or implicit operators that can easily perform matrix-vector multiplication. The power method and inverse power method are the simplest iterative methods for solving matrix eigenvalues.

Power Iteration

A simple method for computing a single eigenvalue and corresponding eigenvector of an $n \times n$ matrix $\boldsymbol{A}$ is , which multiplies an arbitrary nonzero vector repeatedly by the matrix, in effect multiplying the initial starting vector by successively higher powers of the matrix. The power iteration can only find the eigenvalues with the largest absolute value of the matrix A.

```
1 x_0 = arbitrary nonzero vector
2 for k = 0, 1, ··· until convergence do
3 |   y_k = A x_{k-1}
4 |   x_k = y_k / ||y_k||_2
5 |   λ_k = x_k^T A x_k
6 end
```

Algorithm 1: Normalized Power Iteration

例1.3.3 (Normalized Power Iteration.)**.** We apply power iteration to matrix

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

with starting vector

$$\boldsymbol{x}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

As is shown in the right table, the approximate eigenvector is normalized at each iteration, thereby avoiding geometric growth or decay of its components.

k	$\boldsymbol{x}_k^T$		$\|\boldsymbol{y}_k\|_\infty$
0	0.000	1.0	
1	0.333	1.0	3.000
2	0.333	1.0	3.000
3	0.600	1.0	3.333
4	0.778	1.0	3.600
5	0.882	1.0	3.778
6	0.969	1.0	3.939
7	0.984	1.0	3.969
8	0.992	1.0	3.984
9	0.996	1.0	3.992

Inverse Iteration

For some applications, the smallest eigenvalue of a matrix in magnitude is required rather than the largest. We can make use of the fact that the eigenvalues of $\boldsymbol{A}^{-1}$ are the reciprocals of those of $\boldsymbol{A}$, and hence the smallest eigenvalue of $\boldsymbol{A}$ is the reciprocal of the largest eigenvalue of $\boldsymbol{A}^{-1}$. This suggests applying power iteration to $\boldsymbol{A}^{-1}$, but as usual the inverse of $\boldsymbol{A}$ need not be computed explicitly. Inverse iteration converges to the eigenvector corresponding to the smallest eigenvalue of $\boldsymbol{A}$. The eigenvalue obtained is the dominant eigenvalue of $\boldsymbol{A}^{-1}$, and hence its reciprocal is the smallest eigenvalue of $\boldsymbol{A}$ in modulus.

```
1 x_0 = arbitrary nonzero vector
2 for k = 0, 1, ··· until convergence do
3 |   solve A y_k = x_{k-1} for y_k
4 |   x_k = y_k / ||y_ik||_2
5 end
```

Algorithm 2: Inverse Iteration

例1.3.4 (Inverse Iteration.)**.** To illustrate inverse iteration, we use it to compute the smallest eigenvalue of the matrix

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

obtaining the sequence

k	$\boldsymbol{x}_k^T$		$\|\boldsymbol{y}_k\|_\infty$
0	0.000	1.0	
1	−0.333	1.0	
2	−0.333	1.0	0.375
3	−0.600	1.0	0.417
4	−0.778	1.0	0.450
5	−0.882	1.0	0.485
6	−0.969	1.0	0.492
7	−0.984	1.0	0.496
8	−0.992	1.0	0.498
9	−0.996	1.0	0.499

which is converging to an eigenvector $[`1, 1]^T$ corresponding to the dominant eigenvalue of $\boldsymbol{A}^{-1}$, which is 0.5. This same vector is an eigenvector corresponding to the smallest eigenvalue of $\boldsymbol{A}$, $\lambda_2 = 2$, which is the reciprocal of the largest eigenvalue of $\boldsymbol{A}^{-1}$.

Rayleigh Quotient Iteration

If $\boldsymbol{x}$ is an approximate eigenvector for a real matrix $\boldsymbol{A}$, then determining the best estimate for the corresponding eigenvalue λ can be considered as an $n \times 1$ linear least squares approximation problem

$$\boldsymbol{x}\lambda \approx \boldsymbol{A}\boldsymbol{x}.$$

From the normal equation $\boldsymbol{x}^T\boldsymbol{x}\lambda = \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}$, we see that the least squares solution is given by

$$\lambda = \frac{\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}}.$$

The latter quantity, known as the *Rayleigh quotient*, has many useful properties. In particular, it can be used to accelerate the convergence of a method such as power iteration, since at iteration k the Rayleigh quotient $\boldsymbol{x}_k^T\boldsymbol{A}\boldsymbol{x}_k/\boldsymbol{x}_k^T\boldsymbol{x}_k$ gives a better approximation to an eigenvalue than that provided by the basic method alone.

例1.3.5 (Rayleigh Quotient.)**.** Using normalized power iteration in

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

the value of the Rayleigh quotient at each iteration is shown in the following table.

k	$\boldsymbol{x}_k^T$		$\|\boldsymbol{y}_k\|_\infty$	$\boldsymbol{x}_k^T\boldsymbol{A}\boldsymbol{x}_k/\boldsymbol{x}_k^T\boldsymbol{x}_k$
0	0.000	1.0		3.000
1	0.333	1.0	3.000	3.600
2	0.600	1.0	3.333	3.882
3	0.778	1.0	3.600	3.969
4	0.882	1.0	3.778	3.992
5	5.939	1.0	3.882	3.998
6	0.969	1.0	3.939	4.000

Note that the Rayleigh quotient converges to the dominant eigenvalue, $\lambda_1 = 4$, much faster than the successive approximations produced by power iteration alone.

Given an approximate eigenvector, the Rayleigh quotient provides a good estimate for the corresponding eigenvalue. Conversely, inverse iteration converges very rapidly to an eigenvector if an approximate eigenvalue is used as shift, with a single iteration often sufficing. It is natural, therefore, to combine these two ideas together, known as *Rayleigh quotient iteration*.

```
1 x_0 = arbitrary nonzero vector
2 for k = 1, ··· until ‖Ax_{i+1} − ρ_{i+1}x_{i+1}‖_2 < TOL do
3 |   σ_k = x_{k−1}^T A x_{k−1} / x_{k−1}^T x_{k−1}
4 |   Solve (A − σ_k I) y_k = x_{k−1} for y_k
5 |   x_k = y_k / ‖y_k‖_∞
6 end
```

Algorithm 3: Rayleigh Quotient Iteration

As one might expect, Rayleigh quotient iteration converges very rapidly.

例1.3.6 (Rayleigh Quotient Iteration.)**.** Using the matrix

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

and a randomly chosen starting vector $\boldsymbol{x}_0$, Rayleigh quotient iteration converges to the accuracy shown in only two iterations:

k	$\boldsymbol{x}_k^T$		σ_k
0	0.807	0.397	3.792
1	0.924	1.000	3.997
2	1.000	1.000	4.000

A对称情形可用。该算法等同于逆迭代算法中位移取为瑞利商$\rho(\mathbf{x}, A) := \frac{\mathbf{x}^T A\mathbf{x}}{\mathbf{x}^T\mathbf{x}}$。采用初值$\mathbf{x}_0 = [0, \cdots, 0, 1]^T$时与QR迭代得到的序列相同，是分析QR迭代的基础。Releigh商迭代法对单重特征值的计算是局部立方收敛的。

此外，三对角QR迭代是目前求对称三对角矩阵所有特征值最快($O(n)^2$)的方法($n=25$以内);Matlab指令eig,LAPACK程序ssyev(稠密)和sstev(三对角阵)。 二分法和逆迭代(Bisection and inverse iteration)：二分法只求对称三对角阵特征值的一个子集（给定区间）；逆迭代可求相应的特征向量。最坏的情形（许多特征值很接近）不能保证精度；LAPACK程序ssyevx。分而治之(Divide-and-conquer)：求解$n>25$对称三对角矩阵所有特征值和特征向量的最快方法，(平均$O(n)^{2.3}$，最坏$O(n)^3$)；LAPACK程序sstevd。 Jacobi方法是求特征值问题最古老的方法(1846年)；可反复利用Givens变换实现；通常比任何方法都慢($O(n^3)$)变换实现；但结果可以更精确。

QR Iteration

Orthogonalization is a very efficient calculation technique. Iterative methods that use orthogonal techniques to calculate matrix eigenvalues also have very good results. Based on QR orthogonal decomposition, people have proposed the following QR algorithm:

```
1 A_0
2 for k = 1, 2, ··· until convergence do
3 |   Compute QR factorization
4 |   Q_k R_k = A_{k-1}
5 |   A_k = R_k Q_k
6 end
```

Algorithm 4: QR Iteration

Using this method, we will be able to get all the eigenvalues and eigenvectors of the matrix A at the same time. Finally $\{A_k\}_{k=1}^{\infty}$ will converge to an upper triangular matrix whose diagonal elements are all the eigenvalues.This is because:

$$A_k = R_k Q_k = (Q_k^T Q_k) R_k Q_k = Q_k^T (Q_k R_k) Q_k = Q_k^T A_{k-1} Q_k$$

在这里，A_i等同于用正交迭代隐式计算矩阵$Z_i^T Z_i$，且数值稳定。此外, 带位移的QR迭代可加快收敛（当选择的位移接近特征值时二次收敛）

例1.3.7 (QR Iteration.)**.** To illustrate QR iteration, we will apply it to the real symmetric matrix

$$A = \begin{bmatrix} 2.9766 & 0.3945 & 0.4198 & 1.1159 \\ 0.3945 & 2.7328 & -0.3097 & 0.1129 \\ 0.4198 & -0.3097 & 2.5675 & 0.6079 \\ 1.1159 & 0.1129 & 0.6079 & 1.7231 \end{bmatrix}$$

which has eigenvalues $\lambda_1 = 4, \lambda_2 = 3, \lambda_3 = 2, \lambda_4 = 1$. Computing its QR factorization and then forming the reverse product, we obtain

$$A_1 = \begin{bmatrix} 3.7703 & 0.1745 & 0.5126 & -0.3934 \\ 0.1745 & 2.7675 & -0.3872 & 0.0539 \\ 0.5126 & -0.3872 & 2.4019 & -0.1241 \\ -0.3934 & 0.0539 & -0.1241 & 1.0603 \end{bmatrix}$$

Most of the off-diagonal entries are now smaller in magnitude, and the diagonal entries are somewhat closer to the eigenvalues. Continuing for a couple more iterations, we obtain

$$A_2 = \begin{bmatrix} 3.9436 & 0.0143 & 0.3046 & 0.1038 \\ 0.0143 & 2.8737 & -0.3362 & -0.0285 \\ 0.3046 & -0.3362 & 2.1785 & 0.0083 \\ 0.1038 & -0.0285 & 0.0083 & 1.0042 \end{bmatrix}$$

and

$$A_3 = \begin{bmatrix} 3.9832 & -0.0356 & 0.1611 & -0.0262 \\ -0.0356 & 2.9421 & -0.2432 & 0.0098 \\ 0.1611 & -0.2432 & 2.0743 & 0.0047 \\ -0.0262 & 0.0098 & 0.0047 & 1.0003 \end{bmatrix}$$

The off-diagonal entries are now fairly small, and the diagonal entries are quite close to the eigenvalues. Only a few more iterations would be required to compute a he eigenvalues to the full accuracy shown.

Blas Provides software packages for matrix-vector and matrix-matrix operations, and supports types of operations with different precisions (such as float, double) .It is the basis of many other software packages.

Lapack It integrates a large number of basic numerical linear algebra operations, and is developed by many experts in the field of computational mathematics. It is an essential tool for researchers of numerical algebra.

Intel MKL It is not open source, but it can be used for free under Linux system. Compiler instructions optimized for Intel CPU have much faster execution speed than other open source software.

Spooles, mumpus and superLU are also famous software packages for solving direct solutions to linear equations, available in MPI / OpenMP parallel version and serial version. A direct solution for solving linear equations. Open source library installation case: Spooles 2.2

Numerical linear algebra algorithms are an important foundation for high-performance scientific computing. There are many classic textbooks that provide more detailed algorithm details and expanded introductions. Interested readers can get more useful information through these textbooks:

1. Lloyd N. Trefethen & David Bau III: Numerical Linear Algebra
2. Gene H Golub & van Loan: Matrix Computation
3. James W. Demmel: Applied Numerical Linear Algebra

Chapter 2

Numerical Analysis

2.1 方程求根问题

2.1.1 Nonlinear equations

定义2.1.1 (求根问题). 已知$\mathbf{f}$是从$\mathbb{R}^n$到$\mathbb{R}^n$的一个非线性映射。对于任意给定的$\mathbf{y} \in \mathbb{R}^n$，问：$\mathbf{x}$取何值时，$\mathbf{f}$的取值恰好为$\mathbf{y}$，即使得

$$\mathbf{f}(\mathbf{x}) = \mathbf{y}$$

成立？更一般地，将两边同时减去$\mathbf{y}$，使其变为标准形式(为简单起见这里仍用f表示新的映射)：

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{2.1}$$

例2.1.1. 一元非线性方程

$$f(x) = x^2 - 4\sin x = 0$$

的一个近似解为$x = 1.93375$。二元非线性方程组

$$f(x) = \begin{bmatrix} x^2 - y + 0.25 \\ -x + y^2 + 0.25 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

的解为$x = [0.5, 0.5]^T$.

对于一般的$f(x) = 0$，我们无法给出该非线性方程组解的通用表达式。线性方程组只有$0/1/\infty$个解等情况，非线性方程组解的个数可能是任意个。一个经典的做法是以一维情形为突破线索：

例2.1.2 (Some classical 1-dimensional problems). $e^x + 1 = 0$,unsolvable
$e^x - x = 0$,unique solution
$x^2 - 4\sin x = 0$,multi solutions
$\sin x = 0$,infinite solutions

回顾数学分析中的方法，几个存在性定理是研究方程解存在性的几类典型方法：

引理2.1.1 (介值定理). 若f是闭区间上[a,b]的连续函数，c介于f(a)和f(b)，之间，则必存在一个$x^1 \in [a,b]$，满足$f(x^1) = c$，取c为0即可证明f在[a,b]上一定有根。

其次，反函数定理$\mathbf{x} = f^{-1}(\mathbf{0})$。更进一步地，还有压缩映射理论(也是迭代法收敛性的基本定理)也可以获得解的存在性证明.

定义2.1.2 (不动点). f:$\mathbb{R}^n \to \mathbb{R}^n$是集合$S \in \mathbb{R}^n$上的压缩映射，即存在$\gamma \in [0,1]$,对任意集合S内的两个点x，z满足$||f(x) - f(z)|| \leq \gamma||x - z||$。满足$g(x) = x$的$x$称为g的不动点。

定理2.1.1 (压缩映射(不动点)定理). 若g在闭集S上是压缩映射，且$g(S) \in S$,则g在S内存在唯一的不动点。此时如果非线性方程有形如f=x-g，则我们可称，f=0在闭集S上是有唯一解的。

而关于根的进一步研究，需要借助topological degree of function f与重根理论。

2.1.2 Numerical methods for $f(x) = 0$

让我们通过讨论标量非线性方程的求根方法来回顾

二分法

假设我们有一个比较小的区间[a,b]，而函数f在此区间上有符号变化，则连续函数f在这个区间内一定有零点。设初始的函数为f，满足f(a)与f(b)的符号不相同，最终区间长度的误差上线为tol。我们可以得到如下二分法的算法。

例2.1.3 (Dichotomy). Using dichotomy method to solve function $f(x) = x^3 - 4cosx = 0$.

```
function x = bisect(f,a,b,tol)
  xlow = a;  plow  = f(xlow);
  xhigh = b; phigh = f(xhigh);
  while xhigh - xlow > 2*tol,
    xmid = (xlow + xhigh)/2; pmid = f(
        xmid);
    if pmid*plow < 0,
      xhigh = xmid;   phigh = pmid;
    elseif pmid*phigh < 0,
      xlow = xmid;     plow = pmid;
    else
      xlow = xmid;    xhigh = xmid;
    end
  end
  x = [xlow, xhigh];
end
```

x_1	$f(x_1)$	x_2	$f(x_2)$
1	-1.1612	1.5	3.0921
1	-1.1612	1.25	0.6918
1.125	-0.3009	1.25	0.6918
1.125	-0.3009	1.1875	0.1786
1.15625	-0.0653	1.1875	0.1786
1.15625	-0.0653	1.171875	0.05562
1.1640625	-0.00509	1.171875	0.05562
1.1640625	-0.00509	1.16796875	0.025201
1.1640625	-0.00509	1.166015625	0.010037

不动点迭代

不动点迭代的思想为，将原问题变形为：$x = g(x)$的形式，再进行求解，我们可以构造格式$x_{k+1} = gx_k$来对非线性方程进行求解。值得一提的是，这样的构造方式并不唯一，相对应的，不同的构造方式也有不同的稳定性以及收敛速度。

例2.1.4. It is obvious to find out that the roots of nonlinear equation $f(x) = x^2 - x - 2 = 0$ are $x^* = 2$ and $x^* = -1$. There establish various fixed point equation, such as:
1.$g(x) = x^2 - 2$;
2.$g(x) = \sqrt{x+2}$;
3.$g(x) = 1 + 2/x$;
4.$g(x) = (x^2+2)/(2x-1)$;
The corresponding calculation is performed in matlab

```
%f = @(x) x.^2 - 2;
f = @(x) sqrt(x+2); x0 = 0;
err = 1; temp = x0; i = 0;
while err>1e-5
    i = i+ 1;
    x(i) = f(temp);
    err = abs(x(i)-temp);
    temp = x(i);
end
```

k	x_k	$f(x_k)$
0	0	1.414
1	1.414	1.848
2	1.848	1.962
3	1.962	1.990
4	1.990	1.998
5	1.998	1.999
6	1.999	2

牛顿迭代

牛顿法的思想可以理解为：由于零点是其切线与x的交点，非零点则不是，于是在点x_k附近，使用$f(x_k)$处的切线来近似，使用这个切线的零点作为新的近似值，以此来靠近真正的零点。其算法流程可表示为：
x_0 =初始值
for k = 0,1,2,...
$x_{k+1} = x_k - f(x_k)/f'(x_k)$
end

例2.1.5 (Newton Iteration)**.** Using the method of Newton iteration to solve nonlinear equation $f(x) = x^2 - 4cosx = 0$.

It is obvious that $f'(x) = 2x + 4sinx$, so that the iteration scheme is

$$x_{n+1} = x_n - \frac{x_n^2 - 4cosx}{2x + 4sinx}$$

The calculation is performed in matlab, in which the initial guess is $x_0 = 3.0$ and the iterative numerical solution are listed in the following table. It can be found that the convergence is fast as soon as x_n approaching the roots.

```
f = @(x) (x.^2-4*cos(x))./(2*x + 4*sin(
        x));
x0 = 3;  temp = x0; err = 1; i = 1;
while err > 1e-5 && i <= 10
    x(i) = temp - f(temp);
    err = abs(f(temp));
    temp = x(i);  i = i+1;
end
```

k	x_k	$f(x_k)$
1	1.0257	0.18672
2	1.2125	-0.01089
3	1.2016	-3.3033×10^{-5}
4	1.2015	-3.0633×10^{-10}

割线法

牛顿法有一个计算上的缺陷，即在每次迭代时都要计算函数及其导数的值，导数在计算中往往不方便或者计算量很大，因此在步长较小的情况下，我们可以用有限差分来近似代替导数，即用相邻两次迭代的函数值来代替其导数。这种方法叫割线法。

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

其算法流程可以表示为：
x_0 =初始值
for k = 0,1,2,...
$x_{k+1} = x_k - f(x_k)(x_k - x_{k-1}/[f(x_k) - f(x_{k-1})]$
end

例2.1.6 (Secant method). Using the Secant method to solve nonlinear equation $f(x) = x^2 - 4cosx = 0$.

It is obvious that the iteration equation can be written as:

$$x_{n+1} = x_n - (x_n^2 - 4\cos x_n)\frac{x_n - x_{n-1}}{x_n^2 - 4\cos x_n - x_{n-1}^2 + 4\cos x_{n-1}}$$

where the initial guess is $x_0 = 3$, $x_1 = 2$.

```
f = @(x) x.^2-4*cos(x); x0 = 3; x1 = 2;
err = 1; temp1 = x0; temp2 = x1; i = 1;
        x(i) = temp2;
while err > 1e-5
    i = i+1;
    x(i) = temp2 - f(temp2).*(temp2-
        temp1)./(f(temp2)-f(temp1));
    err = abs(f(temp2).*(temp2-temp1)
        ./(f(temp2)-f(temp1)));
    temp1 = temp2;
    temp2 = x(i);
end
```

k	x_{k-1}	x_k	$f(x_k)$
1	3	2	-5.6646
2	2	1.223538	-0.13576
3	1.223538	1.204472	-0.018008
4	1.204472	1.201556	-0.0001094
5	1.201556	1.201538	-8.9895×10^{-8}

如何计算多项式的所有零点？这是一个被人们研究了很久的问题。对于某些特殊情形的n阶多项式p(x)，有时要求求出它的所有n个零点，具体可以使用以下的思想来进行求值：用之前的方法，如牛顿法求出一个根x_1，考虑低一阶的收缩多项式$p(x)/(x-x_1)$，重复此过程，直到求出全部的根。形成给定多项式的友阵，利用之前讲过的计算特征值的方法计算特征值，即计算出多项式的根。一些专门计算多项式零点的方法，如Laguerre法、Traub法等。

2.1.3 Numerical methods for equation system

Considering numerical solution to a general nonlinear equation system

$$\mathbf{F}(\mathbf{x}) = \mathbf{0},$$

which is more competitive than the previous case. It lies in several aspects facts.

- 由于这类问题所涉及的范围更加广泛，所以对解的存在性和个数的分析也更加复杂一些。
- 利用传统的数值方法一般无法既绝对安全又收敛保证地产生有解区间。
- 随着问题维数的增加，计算量也将显著增加。

解决非线性方程组的方法也有很多，如不动点迭代、牛顿法、割线修正法、稳健性牛顿法等，这里主要介绍前三种方法。

Fixed point iteration

给定函数g:$\mathbb{R}^n \to \mathbb{R}^n$，则不动点问题为：寻找$\mathbf{x} \in \mathbb{R}^n$，使

$$\mathbf{x} = g(\mathbf{x}).$$

在一维情形，不动点迭代的收敛与否与收敛速度由g导数的绝对值来决定。而在高维情形下，我们有类似的谱半径条件：

$$\rho(\mathbf{G}(\mathbf{x}^*)) < 1,$$

若满足上述条件，当初始向量充分接近解时，不动点迭代收敛，谱半径越小，收敛速度越快。

Newton's method

定义2.1.3 (Jacobian's Matrix)**.**

$$\{\mathbf{G}(\mathbf{x})_{ij} = \frac{\partial g_i(\mathbf{x})}{\partial x_j}\}.$$

Furthermore, $\mathbf{G}(\mathbf{x})$ means to evaluate G at position $\mathbf{x}$.

对可微函数$\mathbf{f}$:$\mathbb{R}^n \to \mathbb{R}^n$，利用泰勒展开可得

$$\mathbf{f}(\mathbf{x}+\mathbf{s}) \approx \mathbf{f}(\mathbf{x}) + \mathbf{J}(\mathbf{x})\mathbf{s},$$

其中$\mathbf{J}(\mathbf{x})\mathbf{s}$是雅可比矩阵，$\{\mathbf{J}(\mathbf{x})\}_{ij} = \partial f_i(\mathbf{x})/\partial x_j$,即可通过这个迭代过程来求出非线性方程组的零点，其算法为：
$\mathbf{x}_0$ = 初始值
for k=0,1,2,...
解$\mathbf{J}(\mathbf{x}_k)\mathbf{s}_k = -f(\mathbf{x}_k)$求$\mathbf{s}_k$
$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$
end
每一步牛顿法都会解一个线性方程组，因此其运算量是很大的。

定理2.1.2 (local Convergence)**.** F" exists at x^*.

定理2.1.3 (Global Convergence)**.** F" exists and be either non-positive or non-negative.

定理2.1.4 (Semi-local Convergence)**.** Han and Wang's theorem in 1998 paper.

Secant Method

与割线法类似，它是将牛顿法中的雅可比矩阵中的偏导数用每个坐标方向上的有限差分近似代替，从而可以节省不少计算成本，其具体算法可表示为：
$\mathbf{x}_0$ = 初始值
$\mathbf{B}_0$ = 初始雅可比近似
for k=0,1,2,...
解$\mathbf{B}_k\mathbf{s}_k = -f(\mathbf{x}_k)$求$\mathbf{s}_k$
$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$
$\mathbf{y}_k = \mathbf{f}(\mathbf{x}_{k+1}) - \mathbf{f}(\mathbf{x}_k)$
$\mathbf{B}_{k+1} = \mathbf{B}_k + [(\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k)\mathbf{s}_k^T]/(\mathbf{s}_k^T\mathbf{s}_k)$
end
这个方法即可加快迭代进程，提升计算的效率。

例2.1.7. Solving a nonlinear equation system,

$$F(\mathbf{x}) := \begin{bmatrix} 3x_1 - cosx_1 - sinx_2 \\ 4x_2 - sinx_1 - cosx_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

via above numerical methods.

Firstly, let us consider the method of fixed point. It is obvious that the nonlinear equation system can be written as:

$$\mathbf{x} = G(\mathbf{x}) = \begin{bmatrix} (cosx_1 + sinx_2)/3 \\ (sinx_1 + cosx_2)/4 \end{bmatrix}$$

, the calculation is performed in matlab, in which the initial guess is $\mathbf{x}_0 = [1.5, 1]^T$

```
f = @(x1,x2) [(cos(x1) + sin(x2))/3;(sin(x1) + cos(x2))/4];
h = @(x1,x2)[3*x1-cos(x1) - sin(x2);4*x2-sin(x1) - cos(x2)];
x0 = [1.5;1]; err = 1; temp = x0; i = 1;
while err>1e-5
    x(:,i) = f(temp(1),temp(2));
    err = max(abs(h(x(1,i),x(2,i))));
    temp = x(:,i);      i = i + 1;
end
```

k	x_1^k	x_2^k	$f(x_1^k)$	$f(x_2^k)$
1	0.3041	0.3844	-0.4170	0.3114
2	0.4431	0.3066	0.1239	-0.1557
3	0.4018	0.3455	-0.0538	0.0501
4	0.4197	0.3330	0.0190	-0.0206
5	0.4134	0.3381	-0.0074	0.0075
6	0.4158	0.3363	0.0028	-0.0029
7	0.4149	0.3370	0.000396	-0.000410
8	0.4153	0.3367	-0.000150	0.000155
9	0.4152	0.3368	0.000057	-0.000059

Secondly, we consider the Newton method, in which the Jacobi matrix can be written as:

$$J = \begin{bmatrix} 3 + sinx_1 & -cosx_2 \\ -cosx_1 & 4 + sinx_2 \end{bmatrix}$$

. In this sense, the calculation is performed in matlab, in which the initial guess is $\mathbf{x}_0 = [1.5, 1]^T$

```
J = @(x1,x2) [3+sin(x1),-cos(x2);-cos(x1),4+sin(x2)];
h = @(x1,x2)[3*x1-cos(x1) - sin(x2);4*x2-sin(x1) - cos(x2)];
x0 = [1.5;1];   temp = x0; err = 1; i = 1;
while err > 1e-5 && i <= 10
    s = J(temp(1),temp(2))\(-h(temp(1),temp(2)));
    x(:,i) = temp + s;
    err = max(abs(h(x(1,i),x(2,i))));
    temp = x(:,i);   i = i+1;
end
```

k	1	2	3
x_1^k	0.5318	0.4189	0.4152
x_2^k	0.4773	0.3402	0.3368
$f(x_1^k)$	0.2743	0.0095	0.0000084
$f(x_2^k)$	0.5138	-0.0115	0.0000085

At last, let us consider the secant method, in which the initial guess is $\mathbf{x}_0 = [1.5, 1]^T$, and

$$J_0 = \begin{bmatrix} 4 & -0.54 \\ -0.07 & 4.84 \end{bmatrix}$$

```
J = @(x1,x2) [3+sin(x1),-cos(x2);-cos(
        x1),4+sin(x2)];
h = @(x1,x2)[3*x1-cos(x1) - sin(x2);4*
        x2-sin(x1) - cos(x2)];
x0 = [1.5;1]; J0 = J(x0(1),x0(2));
tempx1 = x0; tempJ = J0; err = 1; i = 1;
while err > 1e-5 && i <= 10
    x(:, i) = tempx1;
    s = tempJ\(-h(tempx1(1),tempx1(2)
        ));
    tempx2 = tempx1 + s;
    yk = h(tempx2(1),tempx2(2)) - h(
        tempx1(1),tempx1(2));
    tempJ = tempJ + ((yk - tempJ*s)*s
        ')/(s'*s);
    err = max(abs(h(x(1,i),x(2,i))));
    tempx1 = tempx2; i = i+1;
end
```

k	1	2	3	4
x_1^k	0.5318	0.4382	0.4160	0.4152
x_2^k	0.4773	0.3563	0.3376	0.3358
$f(x_1^k)$	0.2743	0.0601	0.0022	0.00000315
$f(x_2^k)$	0.5138	0.0637	0.0029	-0.00000108

2.2 插值与拟合问题

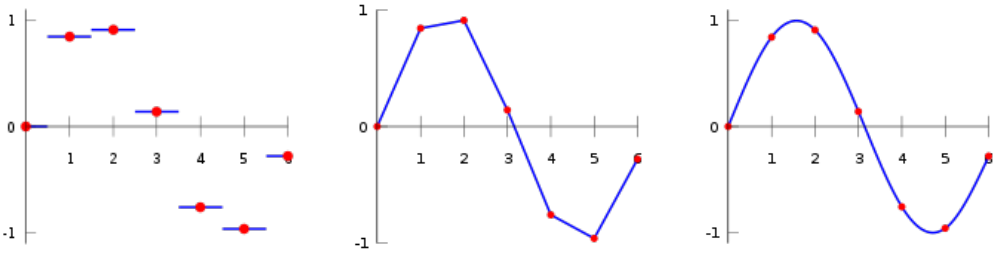
插值问题的来源于各类实际应用，如要求画出一条通过所有离散点的光滑曲线、利用列表函数求中间值、求列表函数的导数或积分、使用简单函数代替复杂函数，从而可以快速方便的求出需要求得的值等等。当然，这些问题又是来自于各类科学与工程计算问题的实际需求中抽象出来的。让用数学的语言进行定义插值问题：

定义2.2.1 (插值问题). 已知数据$\{t_i, y_i\}_{i=0}^n$,寻找函数$f : \mathbb{R} \to \mathbb{R}$,满足：

$$f(t_i) = y_i, \qquad \forall i = 0, 1, ..., n$$

其中f称为插值函数，简称为插值。称$\{t_i\}_{i=0}^n$为插值节点。

如下几个例子描述了如何找到一些“简单”函数来“穿过”给定的数据点： 插值函数



的选择主要考虑因素是多方面的。首先，函数的简单程度是考虑的重要因素，因为越简单的函数越有利于计算机进行求值等各种运算，如多项式是计算机最擅长的数学工

具。其次，也要考虑与拟合数据在性态方面的接近程度（如：光滑性，单调性，周期性等），比如分段多项式、三角函数、指数函数和有理函数等也是数值计算中常用的数学工具。

对于给定的数据点集$(t_i, y_i), i = 1, 2, ..., n$,选择基函数$\phi_1(t), \phi_2(t), ...\phi_n(t)$,在这组基函数所张成的函数空间中选择一个插值函数。将插值函数f写成这些基函数的线性组合：

$$f(t) = \sum_{j=1}^{n} x_j \phi_j(t),$$

其中x_j是待定的参数，则数据点(t_i, y_i)上的插值函数f应满足：

$$f(t_i) = \sum_{j=1}^{n} x_j \phi_j(t_i) = y_i,$$

将其写成$\mathbf{Ax} = \mathbf{y}$的矩阵形式，$\mathbf{A}$的元素为$a_{ij} = \phi_j(t_i)$。

利用上述形式，结合线性代数的知识我们有如下结论:

性质2.2.1 (存在唯一性)**.** 若基函数个数n与数据个数m相等，则得到的是一个方阵线性方程组。若矩阵$\mathbf{A}$非奇异，则一定有且仅有唯一解。而若矩阵$\mathbf{A}$奇异，则可以有许多参数的解，代表着数据点不能被精确拟合。

性质2.2.2 (病态性)**.** 基函数可以有许多选择的方式，相对应的会有许多矩阵$\mathbf{A}$的表达形式。$\mathbf{A}$若是单位阵，下三角矩阵，三对角矩阵等等特殊的矩阵，会大大提升求解参数的效率，降低求解的难度，在之后的具体例子里有所体现。

2.2.1 Interpolation with polynomials

最简单直接的基底是单项式基底，即对于n个数据点，进行选取k=n-1，则有单项式基底：

$$\phi_j(t) = t^{j-1}, \quad j = 1, 2, ..., n,$$

任何n-1次多项式，皆可用这个基底的线性组合来表示。此时，插值多项式的参数可由下述方程组求解：

$$\mathbf{Ax} = \begin{bmatrix} 1 & t_1 & ... & t_1^{n-1} \\ 1 & t_2 & ... & t_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t_n & ... & t_n^{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \mathbf{y}$$

这里，$\mathbf{A}$是一个范德蒙矩阵，当$x_0 < x_1 < ... < x_n$时非奇异。单项式基底具有存在及唯一性，因此解此方程组需要的工作量是$\mathbf{O}(n^3)$, 计算成本高！

拉格朗日插值公式

对给定的数据点$(t_i, y_i), i = 1, 2, ..., n$,n-1次拉格朗日基函数可以表示为：

$$l_j(t) = \frac{\prod_{k=1,k\neq j}^{n}(t - t_k)}{\prod_{k=1,k\neq j}^{n}(t_j - t_k)}, j = 1, 2, ..., n,$$

显然，我们有结论：

$$l_j(t) = \begin{cases} 1, & i = j, \\ 0, & i \neq j \end{cases} i, j = 1, 2, \ldots, n,$$

说明对于拉格朗日插值来说，$\mathbf{A}$ 是单位矩阵$\mathbf{I}$，因此参数$x_1, x_2, \ldots, x_n$可以直接由y_i得到。拉格朗日插值求参数是很容易的，但是同单项式基底表达式相比，它基函数形式更加复杂，并且积分与微分的操作会困难许多。

牛顿插值公式

对给定的数据点$(t_i, y_i), i = 1, 2, ..., n$, n-1次牛顿基函数为：

$$\pi_j(t) = \prod_{k=1}^{j-1}(t - t_k), \qquad \forall j = 1, 2, ..., n,$$

注意到k始终比j小1，也就是说，多项式可以表示为：

$$Q_n(x) = x_1 + x_2(t - t_1) + x_3(t - t_1)(t - t_2) + \cdots + x_n(t - t_1)(t - t_2)\cdots(t - t_{n-1}). \quad (2.2)$$

这里，$\mathbf{A}$是下三角矩阵，解$\mathbf{Ax} = \mathbf{y}$复杂度$O(n^2)$ 同前两种方法相比，牛顿插值既可以节省计算量，又可以节约计算给定点的值的成本。牛顿法的另一大优势在于，它可以逐步进行计算。如果我们需要增加插值点的个数的时候，可以直接计算增加的点即可，这是其他两种方法做不到的。让我们考虑加入点(t_{n+1}, y_{n+1}),利用牛顿法可得新的插值多项式:

$$Q_{n+1}(t) = Q_n(t) + x_{n+1}\pi_{n+1}(t)$$

其中π的定义与之前的基函数相同，且

$$x_{n+1} = \frac{y_{n+1} - Q_n(t_{n+1})}{\pi_{n+1}(t_{n+1})}.$$

而之前两种插值方式只能将所有点再重新计算，再继续求解参数。

这里我们给出一例以体现三种不同插值公式的区别：

例2.2.1 (Base function with three formulas of interpolation)**.** Plot the base function of Lagrange, monominal and Newton interpolation and point out the difference between them. Try to compute the corresponding secondary Interpolation polynomials with datas $(-1, 1), (0, 0), (1, 1)$.

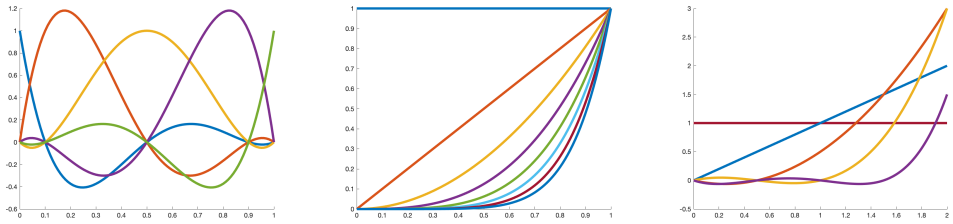
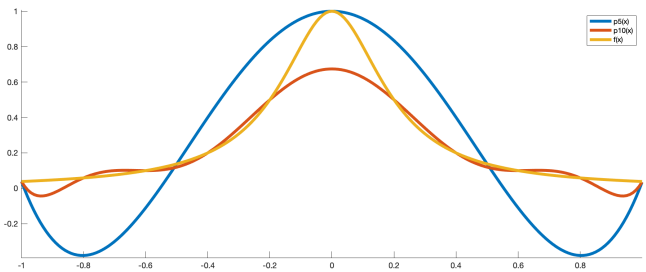


Figure 2.1: (Left)Lagrange base, (Middle) monominal base, (Right) Newton base

2.2.2 Interpolation with piecewise polynomials

基函数和数据点的合适选择可以减轻高次多项式插值的困难，但有时候用一个多项式来拟合大量数据点，会出现令人不悦的振荡现象。



分片思想

分段多项式主要思想是: 对给定的数据点集$(t_i, y_i), i = 1, \cdots, n, \quad t_1 < t_2 < \cdots < t_n$,作分段多项式插值时，在每个子区间$[t_i, t_{i+1}]$上用不同的多项式。典型的分段插值方法包括三次样条插值和三次Hermit插值。

例2.2.2 (Interpolation with cubic splines and cubic Hermitt)**.** Using cubic splines and cubic Hermit method to approximate datas (t_i, y_i), where

t	0	1	3	4	6	7	9	10
y	8	6	5	2	1.5	1.3	1.1	1

```
t = [0 1 3 4 6 7 9 10];
y = [8 6 5 2 1.5 1.3 1.1 1];
x = 0:0.2:10;
yy = spline(t,y,x);
plot(t,y,'o',x,yy,'linewidth',2)
```

```
t = [0 1 3 4 6 7 9 10];
y = [8 6 5 2 1.5 1.3 1.1 1];
x = 0:0.2:10;
yy=interp1(t,y,x,'pchip');
plot(t,y,'o',x,yy,'linewidth',2)
```

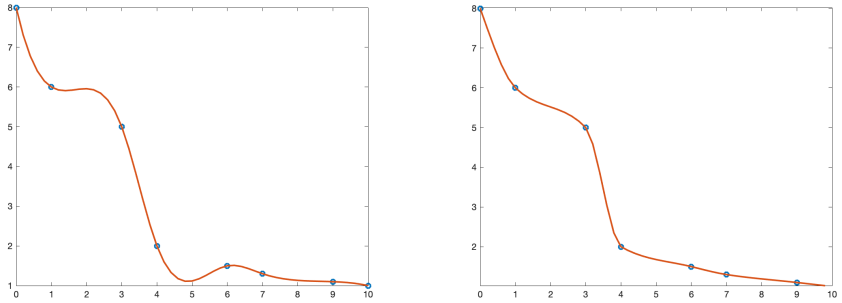


Figure 2.2: (Left)Interpolation with cubic splines;(Right)Interpolation with cubic Hermitt

差商

差商的定义：$f[x_0,\cdots,x_n]=\dfrac{f[x_1,\cdots,x_n]-f[x_0,\cdots,x_{n-1}]}{x_n-x_0}$

差商的性质:

- $f[x_0,\cdots,x_n]=\dfrac{f^{(n)}(\xi)}{n!}$，其中$\min\{x_i\}<\xi<\max\{x_i\}$；
- 交换不变
- $\frac{\partial}{\partial x}f[x_0,\cdots,x_n,x]=f[x_0,x_1,\cdots,x_n,x,x]$

则可以得到插值多项式：

$$f(x)=P_n(x)+\omega_{n+1}(x)f[x_0,x_1,\cdots,x_n,x]$$

并且此时的误差为：

$$E_{n+1}=f(x)-P_n(x)=\omega_{n+1}(x)f[x_0,x_1,\cdots,x_n,x].$$

其中可以计算出二阶的误差为：$E_2=-\frac{h^2}{8}f''(\xi)$

B-样条

B-Spline (which means Basis Spline) was one of earliest splines. It overcame the problems encountered by the Bezier Curve, by providing a set of blending functions that only had effect on a few control points. This gave the local control that was lacking. As most other spline techniques provided the required continuity at the loss of local control, the problem of piecing curves together was avoided by allowing only those curves.

定义2.2.2 (B-spline)**.** The 0_{th}-order B-spline is defined as following

$$B_i^0(t) = \begin{cases} 1 & t_i \leq t < t_{i+1} \\ 0 & others, \end{cases}$$

for any $k > 0$, the $k_{th} - order$ B-spline can be computed as

$$B_i^k(t) = v_i^k(t)B_i^{k-1}(t) + (1 - v_{i+1}^k)B_{i+1}^{k-1}(t).$$

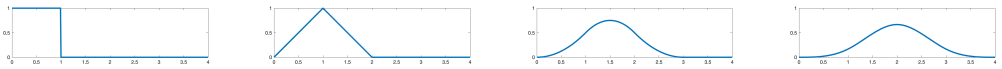


Figure 2.3: Base function of B-spline, which $i = 0, k = 0, 1, 2, 3$.

2.2.3 Best approximation

2.2.4 Least square approximation

Least square approximation is one of the most useful fitting methods, which are represented as:

定义2.2.3 (Least square approximation)**.** With some known data $\{t_i, y_i\}_{i=0}^n$, find out function $f : \mathbb{R} \to \mathbb{R}$, which satisfies:

$$minJ = min \sum_{i=0}^{n} (f(t_i) - y_i)^2.$$

Choosing some base functions $\phi_1(t), \phi_2(t), \ldots, \phi_n(t)$, the fitting function can be represented as $f(t_i) = \sum_{j=1}^{n} x_j \phi_j(t_i)$. In this sense, the least square approximation can be transferred to

$$\mathbf{A}^T A\mathbf{x} = \mathbf{A}^T\mathbf{y},$$

where $A_{ij} = \phi_j(t_i)$. Let us consider three types of useful base functions, including polynomial base, non-polynomial base and B-spline base.

Polynomial base for data fitting

For data set $\{t_i, y_i\}, i = 1, 2, \ldots, n$, choosing $k < n - 1$, so that the polynomial base can be written as:

$$\phi_j(t) = t^{j-1}, j = 1, 2, k.$$

In this sense, the fitting matrix A can be computed as

$$A^T Ax = A^T y, A = \begin{bmatrix} 1 & t_1 & ... & t_1^{k-1} \\ 1 & t_2 & ... & t_2^{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t_n & ... & t_n^{k-1} \end{bmatrix}$$

Non-polynomial base for data fitting

For data set $\{t_i, y_i\}, i = 1, 2, \ldots, n$, choosing some non-polynomial functions $f_j(t), j = 1, 2, \ldots, k$ as base functions, where $k < n-1$. Then the fitting matrix A can be computed as

$$A^T Ax = A^T y, A = \begin{bmatrix} f_1(t_1) & f_2(t_1) & ... & f_k(t_1) \\ f_1(t_2) & f_2(t_2) & ... & f_k(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(t_n) & f_2(t_n) & ... & f_k(t_n) \end{bmatrix}$$

B-spline base for data fitting

For data set $\{t_i, y_i\}, i = 1, 2, \ldots, n$, choosing knots $= \{m_j | t_1 \leq m_1, m_2, \ldots, m_k \leq t_n, k < n - 1\}$, so that the $p-$order fitting matrix of B-spline can be written as:

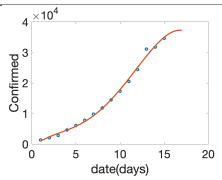
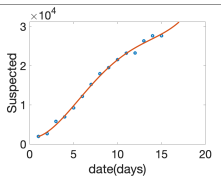
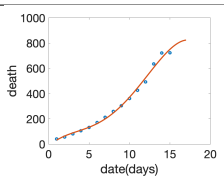
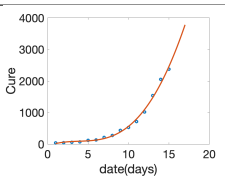
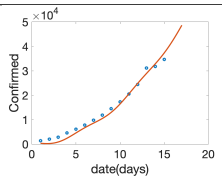
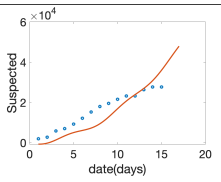
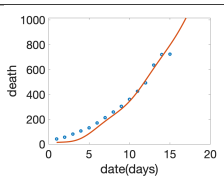
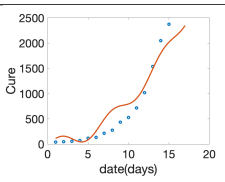
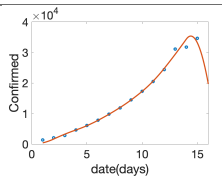
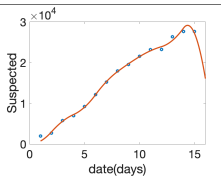
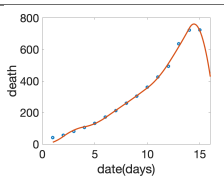
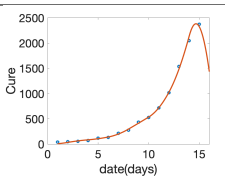
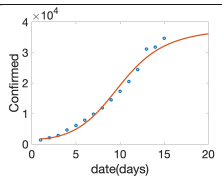
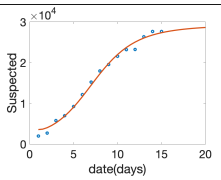
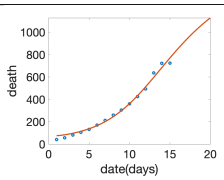
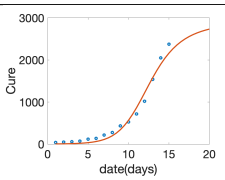
$$A^T Ax = A^T y, A = \begin{bmatrix} B_1^p(t_1) & B_2^t(t_1) & ... & B_k^p(t_1) \\ B_1^p(t_2) & B_2^p(t_2) & ... & B_k^p(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ B_1^p(t_n) & B_2^p(t_n) & ... & B_k^p(t_n) \end{bmatrix}$$

例2.2.3 (Data fitting)**.** Using the above base functions to fit the 2019-NCP datas, including confirmed cases, suspected cases, death and cure cases. The datas are shown as following:

类似问题：多元线性回归、非多项式数据拟合等

关于回归、插值、逼近、拟合的区别：插值：插值曲线要经过型值点;逼近：为复杂函数寻找近似函数，其误差在某种度量意义下最小;拟合：在插值问题中考虑**给定数据点的误差**，其误差在某种度量意义下最小;回归：最小二乘解,在$\|\cdot\|_2$度量意义下最小。

Date	1.25	1.26	1.27	1.28	1.29	1.30	1.31
Confirmed	1377	2071	2846	4630	6086	7830	9811
Suspected	1983	2692	5794	6973	9239	12167	15238
Death	41	56	81	106	132	171	213
Cure	39	49	56	73	119	135	214
2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8
11890	14490	17341	20530	24439	31161	31774	34673
17988	19544	21558	23214	23260	26359	27657	27657
259	304	361	426	493	636	722	724
275	434	527	718	1019	1540	2050	2375

2019-NCP	Confirmed	Suspected	death	Cure
Polynomial k = 4				
Non-polynomial $f_1(x) = lnx$ $f_2(x) = sinx$ $f_3(x) = cosx$				
B-spline p = 3				
Neural network 56 parameters 2 hidden layers				

2.2.5 Neural network approximation

Deep neural networks can be regarded as basis functions to approximate some unknown function. It belongs to the basic approximation problems. Let us introduce the definition of neural networks, which is defined as following:

定义2.2.4 (Neural network)**.** A network$NET(\mathbf{x};\mathbf{w},\mathbf{b})$ consisting of L hidden layers is defined like the figure 2.4. As it can be seen, in layer $l, l = 0, 1, \ldots, L$, let $\mathbf{w}^l, \mathbf{b}^l$ denote

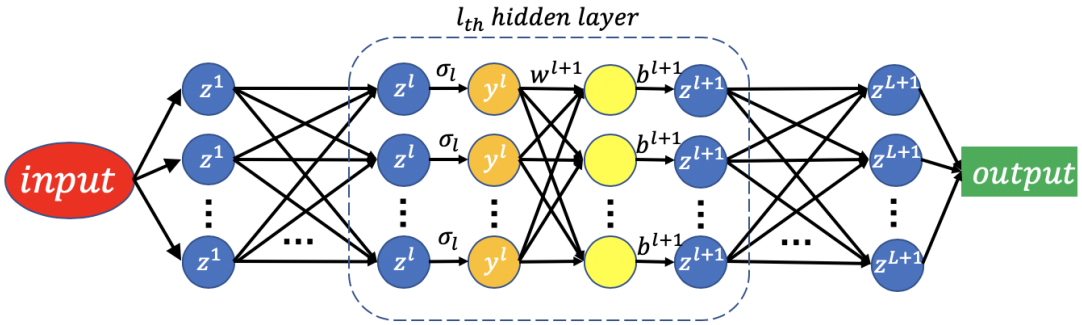


Figure 2.4: Structures of ANN with L hidden layers

the weights and bias and σ_l be the activation functions. Moreover

$$\begin{aligned} \mathbf{z}^{l+1} &= \mathbf{w}^{l+1}\mathbf{y}^l + \mathbf{b}^{l+1}, \\ \mathbf{y}^{l+1} &= \sigma_{l+1}(\mathbf{z}^{l+1}) \end{aligned}$$

With the notation $\mathbf{w} = \{\mathbf{w}^1, \mathbf{w}^2, \ldots, \mathbf{w}^{L+1}\}, \mathbf{b} = \{\mathbf{b}^1, \mathbf{b}^2, \ldots, \mathbf{b}^{L+1}\}$, network $NET(\mathbf{x};\mathbf{w},\mathbf{b}) = \mathbf{y}^{L+1}$.

The aim for neural network approximation is to find proper $NET(\mathbf{x};\mathbf{w},\mathbf{b})$ such that

$$J(\mathbf{x}) = \|f(\mathbf{x}) - NET(\mathbf{x};\mathbf{w},\mathbf{b})\|_2^2,$$

approach a minimize value.

性质2.2.3 (Universal approximation capabilities)**.** Standard multilayer feedforward networks with as few as a single hidden layer and arbitrary bounded and non-constant activation function are universal approximation operator with respect to $L^p(\mu)$ performance criteria.

性质2.2.4 (Universal approximation capabilities for derivatives)**.** Networks with sufficiently smooth activation functions are capable of arbitrarily accurate approximation to a function and its derivatives.

The most important thing to solve neural network approximation is the back propagation algorithm, which is shown as following

$$\begin{aligned}\mathbf{w}^{n+1} &= \mathbf{w}^n - \Delta t\frac{\partial J}{\partial \mathbf{w}},\\ \mathbf{b}^{n+1} &= \mathbf{b}^n - \Delta t\frac{\partial J}{\partial \mathbf{b}}.\end{aligned}$$

The gradient can be computed by chain rules, which are shown for details in the following equations.

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{w}^l} &= 2(\mathbf{y}^{L+1} - f(\mathbf{x}))\frac{\partial \mathbf{y}^{L+1}}{\partial \mathbf{w}^l} = \frac{\partial \mathbf{y}^{L+1}}{\partial \mathbf{z}^l} * \frac{\partial \mathbf{z}^l}{\partial \mathbf{w}^l} = \frac{\partial \mathbf{y}^{L+1}}{\partial \mathbf{z}^l} * \mathbf{y}^{l-1},\\ \frac{\partial J}{\partial \mathbf{b}^l} &= 2(\mathbf{y}^{L+1} - f(\mathbf{x}))\frac{\partial \mathbf{y}^{L+1}}{\partial \mathbf{b}^l} = \frac{\partial \mathbf{y}^{L+1}}{\partial \mathbf{z}^l} * \frac{\partial \mathbf{z}^l}{\partial \mathbf{b}^l} = \frac{\partial \mathbf{y}^{L+1}}{\partial \mathbf{z}^l}.\end{aligned}$$

By defining $\delta^l := \frac{\partial \mathbf{y}^{L+1}}{\partial \mathbf{z}^l}, l = 1, 2, \ldots, L+1$, above equation yields that

$$\begin{aligned}&\delta^{L+1} = \sigma'_{L+1}(\mathbf{z}^{L+1}),\\ &\delta^l = \left(\mathbf{w}^{l+1}\right)^T * \delta^{l+1} \odot \sigma'_l(\mathbf{z}^l), \quad l = 1, 2, \ldots, L;\end{aligned}$$

where $\odot$ represent multiplication by corresponding elements of two vectors.

例2.2.4. Using neural network to approximate datas in example 2.2.3. The result can be find in the last example.

2.3 Quradrature Rule

2.3.1 Numerical Integration

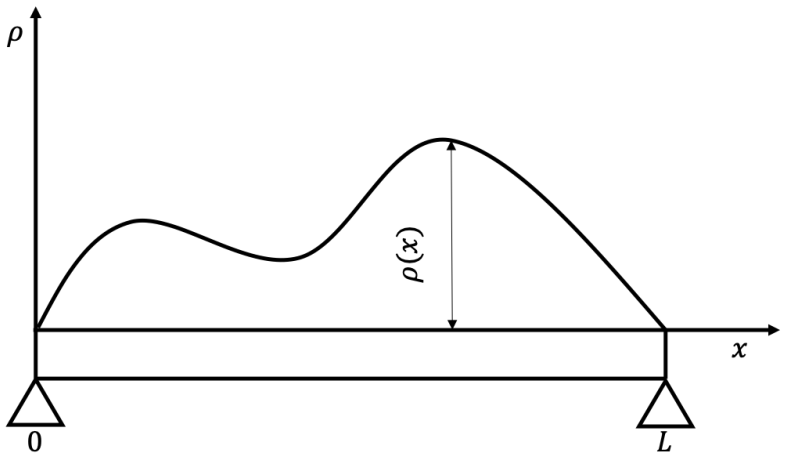
Integration is widely used in various fields, including integration transform, such as Laplace and Fourier transform, finite element method and boundary element method of partial differential equation, integral equation, variational method and statistics problem, in which many basic concepts, like probability distribution and expectation, are defined. In addition, the definition of system energy is also defined by integration in the field of classical and quantum physics.

例2.3.1 (Distributed load problem)**.** If $\rho(x)$ is the density function of distributed load, the total load W, shown in the following figure, can be computed as the measure of area， which is shown as

$$W = \int_0^L \rho(x)dx.$$

Moreover, the barycenter of the distributed load can be calculated as

$$\bar{x} = \frac{1}{W}\int_0^L x\rho(x)dx.$$



定义2.3.1 (Integration)**.** For a function f:$\mathbb{R} \to \mathbb{R}$ in interval $[a, b]$, the integration is defined as

$$\mathbf{I}(f) = \int_a^b f(x)dx,$$

which is illustrated for details as

$$R_n = \sum_{i=1}^{n}(x_{i+1} - x_i)f(\xi_i),$$

where $a = x_1 < x_2 < \cdots < x_n < x_{n+1} = b$, $\xi \in [x_i, x_{i+1}]$. By denoting $h_n = \max_{i=1,\ldots,n-1}\{x_{i+1} - x_i\}$, we call that f is Riemann integrable in the interval if

$$\lim_{h_n \to 0} R_n = R \neq \infty,$$

Discuss

The regularity, including existence, uniqueness and condition number of integral (model, method) is a very important problem to discuss, which is illustrated as:

- Existence is that if the integration above is solvable, which is illustrated that the bounded and continuous integration is integrable.

- Uniqueness is that the existence is independent with the choice of ξ_i and mesh.
- Stability is one of the most important topics in integration, which can be represented the propagation of errors with the noise of data.

Let us analysis the influence of noise in two ways: the noise about integral function and the integral interval, which is illustrated as:

$$|I(\hat{f}) - I(f)| \leq \int_a^b |\hat{f}(x) - f(x)|dx \leq (b-a)\|\hat{f} - f\|_\infty$$

and

$$\left| \int_a^{\hat{b}} f(x)dx - \int_a^b f(x)dx \right| \leq \int_b^{\hat{b}} f(x)dx \leq (\hat{b} - b)\|f\|_\infty$$

Numerical quadrature

Numerical quadrature is to compute the definite integral with some numerical methods, where the definite integral is defined as :

$$I(f) = \int_a^b f(x)dx,$$

The basic idea of numerical quadrature is the Quadrature Rule.

定义2.3.2 (Quadrature Rule)**.** Find a approximation $Q_n(f)$ of $I(f)$:

$$Q_n(f) = \sum_{i=1}^{n} \omega_i f(x_i),$$

where $a < x_1 < \cdots < x_n < b$ is nodes， ω_i is weight. The target of numerical quadrature is to choose proper nodes and weights, to get a satisfying approximation $Q_n(f)$ with reasonable calculation cost.

2.3.2 Formula with interpolation case(Newton-Cotes)

For convenience, let us consider nodes of equal distance, which can be written as:

$$x_i = a + i(b-a)/(n+1), i = 1, 2, \cdots, n.$$

Indefinite coefficient method

Indefinite coefficient method is one of the most important methods to get an integral formula. For various basic function $\phi_i(x)$, there establish that

$$\sum_{i=1}^{n} \omega_i \phi_j(x_i) = \int_a^b \phi_j(x)dx, j = 1, 2, \ldots, n,$$

where the matrix form is regarded as

$$Aw = f,$$

where

$$A = \begin{bmatrix} \phi_1(x_1) & \phi_1(x_2) & \cdots & \phi_1(x_n) \\ \phi_2(x_1) & \phi_2(x_2) & \cdots & \phi_2(x_n) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_n(x_1) & \phi_n(x_2) & \cdots & \phi_n(x_n) \end{bmatrix}, w = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{bmatrix}, f = \begin{bmatrix} \int_a^b \phi_1(x)dx \\ \int_a^b \phi_2(x)dx \\ \vdots \\ \int_a^b \phi_n(x)dx \end{bmatrix}$$

例2.3.2 (3-order monomial basis of indefinite coefficient method)**.** Compute the three-nodes integral formula with

$$Q_3(f) = \omega_1 f(x_1) + \omega_2 f(x_2) + \omega_3 f(x_3).$$

Firstly, let us choose three points $x_1 = a, x_2 = \frac{a+b}{2}, x_3 = b$ and the basic function is chosen by $\phi_j(x) = x^{j-1}, j = 1, 2, 3$. Then the matrix form of the above problem can be represented as

$$Aw = f, A = \begin{bmatrix} 1 & 1 & 1 \\ a & \frac{a+b}{2} & b \\ a^2 & \left(\frac{a+b}{2}\right)^2 & b^2 \end{bmatrix}, f = \begin{bmatrix} b-a \\ \frac{b^2-a^2}{2} \\ \frac{b^3-a^3}{3} \end{bmatrix}$$

And the solution is

$$\omega_1 = \frac{b-a}{6}, \omega_2 = \frac{2(b-a)}{3}, \omega_3 = \frac{b-a}{6}.$$

With the different choice of basic functions in indefinite coefficient method, the formula can be different. Especially, we can choose some interpolation basic function to obtain the formulas, for example, the Lagrange interpolation basic function:

$$\sum_{i=1}^{n} \omega_i l_j(x_i) = \int_a^b l_j(x)dx, j = 1, 2, \ldots, n,$$

In this sense, let us introduce some useful rules of numerical quadrature(Newton-Cotes' case)

Trapzoid rule

Choosing two nodes $x = a$ and $x = b$, then the trapzoid rule can be written as

$$Q_n(f) = (b-a)\frac{f(a)+f(b)}{2}$$

The error is:

$$E_2 = \left| \int_a^b f(x)dx - Q_n(f) \right| = \int_a^b \frac{(b-1)^2}{8} f''(\xi)dx = \frac{(b-a)^3}{8}\|f''\|_\infty$$

Compound trapezoidal formula($n+1$ nodes):

$$T_n = h\sum_{i=0}^{n-1} \frac{f(x_i)+f(x_{i+1})}{2} = \frac{b-a}{n}\left[\frac{1}{2}f(x_0) + f(x_1) + \cdots + f(x_{n-1}) + \frac{1}{2}f(x_n)\right]$$

With the err: $E_{rr}^T = -\frac{(b-a)\|f''\|_\infty}{12}h^2$

Simpson rule

$$S(f) = \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

Error :

$$E_3 = \left| \int_a^b f(x)dx - S(f) \right| = -\frac{(b-a)^5}{2880} f^{(4)}(\eta), a < \eta < b$$

Compound Simpson formula($n+1$ nodes) :

$$S_n = \frac{h}{6}\sum_{i=0}^{n-1}\left[f(x_i) + 4f(x_{i+\frac{1}{2}}) + f(x_{i+1})\right].$$

With the error: $E_{rr}^S = -\frac{(b-a)\|f^{(4)}\|_\infty}{2880}h^4$.

例2.3.3 (numerical quadrature:Newton-Cotes)**.** Using different formulas in Newton-Cotes to compute $I(f) = \int_0^1 e^{-x^2}dx$.(The exact solution is 0.746824)

$$M(f) = (1-0)e^{-0.25} = 0.778801$$

$$T(f) = \frac{1}{2}(e^0 + e^{-1}) = 0.683940$$

$$T(f) = \frac{1}{6}(e^0 + 4e^{-0.25} + e^{-1}) = 0.747180$$

2.3.3 Methods for improving the accuracy of integration

There are many methods to improve the accuracy of integration, such as

- Compound method is a useful way to improve the accuracy, whose main idea is similar to the piecewise interpolation.
- Adaptive method, whose main idea is similar to recursion method
- Richardson extrapolation method, whose main idea is similar to Romberg integration.

Moreover, there are other classical problems that many researchers are studying, those are

- Tabular data quadrature. The solution is to interpolate first, and then integrate.
- Singular integration. The solution is to obtain the limitation or do some variable substitution.
- Double and Triple integration. The solution is to use tensor product or simplex.
- High dimensional integration. Monte Carlo method(usually ill-conditioned).

2.3.4 Best quadrature rule

The accuracy of numerical integration formula for fixed nodes is limited, because the information is not fully used, which lead the best quadrature rule.

Clenshaw-Curtis rule

Some research shows that, the weights of integration can always be positive, by using the zero point of Chebyshev polynomial as the integral nodes. Moreover, the method do not need compute weights before integration any more, because we can use the FFT and recursive algorithm. The disadvantage of it is that the accuracy is not the best.

Gauss rule

The basic idea is to choose the best nodes and weights to make the accuracy being best.

例2.3.4 (Construct two point formula$G_2(f)$ in the interval (-1,1))**.**

$$I(f) = \int_{-1}^{1} f(x)dx \approx \omega_1 f(x_1) + \omega_2 f(x_2) := G_2(f),$$

Assume it establishes for $f_j(x) = x^{j-1}, j = 1, 2, 3, 4$

$$\begin{cases} \omega_1 + \omega_2 = \int_{-1}^{1} 1dx = 2, \\ \omega_1 x_1 + \omega_2 x_2 = \int_{-1}^{1} x dx = 0, \\ \omega_1 x_1^2 + \omega_2 x_2^2 = \int_{-1}^{1} x^2 dx = \frac{2}{3}, \\ \omega_1 x_1^3 + \omega_2 x_2^3 = \int_{-1}^{1} x^3 dx = \frac{2}{3}, \end{cases}$$

The solution is

$$x_1 = -1/\sqrt{3}, x_2 = 1/\sqrt{3}, \omega_1 = 1, \omega_2 = 1,$$

which yeilds

$$G_2(f) = f(-1/\sqrt{3}) + f(1/\sqrt{3}).$$

It is also worth to point out that

- it is important to learn how to compute by the orthogonal polynomials such as Legendre polynomial.
- The Gauss rule points are given in the unit interval, affine transformation is necessary in application.
- The integration can be computed by Gauss-Kronrod recursive method.

2.4 Numerical Methods for Initial Value Problems(IVPs)

2.4.1 Initial Value Problem

Considering first-order Ordinary Differential Equations(ODEs)

$$\mathbf{y}' = \mathbf{y}'(t) := \begin{bmatrix} y_1'(t) \\ y_2'(t) \\ \vdots \\ y_n'(t) \end{bmatrix} = \begin{bmatrix} f_1(t, \mathbf{y}) \\ f_2(t, \mathbf{y}) \\ \cdots \\ f_n(t, \mathbf{y}) \end{bmatrix} := \mathbf{f}(t, \mathbf{y}), \tag{2.3}$$

where $\mathbf{y} := (y_1, y_2, \cdots, y_n)^T$ and $\mathbf{f} : \mathbb{R}^{n+1} \to \mathbb{R}$. When $n > 1$, it is a system of coupled ODEs.

例2.4.1. An given n-th order ODE

$$y^{(n)} = f(t, y, y', y'', \cdots, y^{(n-1)})$$

can be transformed into a first-order system

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}).$$

The above ODEs (2.3) will admit a unique solution as soon as an initial condition

$$\mathbf{y}(t_0) = \mathbf{y}_0 \tag{2.4}$$

is given. In the literature, people referred (2.3)+(2.4) as the Initial Value Problems(IVPs) for ODEs. By integrating the ODE, its solution is given by the integral

$$\mathbf{y}(t) = \mathbf{y}_0 + \int_{t_0}^{t} \mathbf{f}(s)ds \tag{2.5}$$

定理2.4.1 (Existence and Uniqueness of the solution to IVPs)**.** If $\mathbf{f} : \mathbf{R}^{n+1} \to \mathbf{R}^n$ is Lipschitz continuous on D, which means

$$\|\mathbf{f}(t, \hat{\mathbf{y}}) - \mathbf{f}(t, \mathbf{y})\| \leq L\|\hat{\mathbf{y}} - \mathbf{y}\|,$$

then the solution to IVPs (2.3)+(2.4) has unique solution for any given $\mathbf{y}_0$ defined on D.

例2.4.2 (Newton's Second Law of Motion)**.** As well known as the formula $F = ma$, force equals mass times acceleration. It is straightforward that $a = s''(t)$, where t means the time and $s(t)$ is the displacement at time t. Considering the trajectory of falling object under the force of Earth's gravity, which means $F = -mg$. When the initial position $s(0)$ and initial velocity $v(0) = s'(0)$ are given, we have its solution

$$s(t) = -\frac{1}{2}gt^2 + s'(0)t + s(0).$$

Considering the linear and homogeneous system of ODEs in the form of

$$\mathbf{y}' = A\mathbf{y},$$

where A is an $n \times n$ constant coefficient matrix, which is assumed to be diagonalizable. The initial condition $\mathbf{y}(t_0) = \mathbf{y}_0$ is given, then it is easy to confirm that

$$\mathbf{y}(t) = \sum_{i=1}^{n} \alpha_i \mathbf{v}_i e^{\lambda_i t},$$

where $\{\lambda_i\}_{i=1}^n$ and $\{\mathbf{v}_i\}_{i=1}^n$ are eigenvalues and corresponding eigenvectors of A, and $\mathbf{y}_0 = \sum_{i=1}^{n} \alpha_i \mathbf{v}_i$.

性质2.4.1 (Stability of Solutions)**.**

2.4.2 Numerical Methods for ODEs

Finite Difference Approximations

It is essential to explain the approximation of the first order and the second order derivatives.

1. 1-order forward/backward difference quotient

$$f'(x) \approx D_h^+ := \frac{f(x+h) - f(x)}{h}$$

2. 1-order central difference quotient

$$f'(x) \approx D_{2h} := \frac{f(x+h) - f(x-h)}{2h}$$

3. One-sided difference quotient

$$f'(x) \approx D_{2h}^+ := \frac{-3f(x)+4f(x+h)-f(xi+2h)}{2h}$$

例2.4.3. Verifying the accuracy of the above schemes are$o(h)$ and $o(h^2)$ by Taylor formula, respectively.

4. 2-order central difference quotient

$$f''(x) = (f(x+h)-2f(x) + f(x-h))/h^2$$

Assuming that $D_h^2 f(x) = Af(x+h) + Bf(x) + Cf(x-h)$, then based on Taylor formula, there yields

$$\begin{cases} A + B + C = & 0; \\ h(A - C) = & 0; \\ \dfrac{h^2}{2}(A + C) = & 1. \end{cases}$$

The error is $-\frac{h^2}{12} f^{(4)}(x)$(How to prove?)

Euler's Method

Let us consider the 1-order 1-dimensional ODE initial value problem:

$$\frac{dy}{dx} = f(x, y), x \in [a, b]$$

$$y(a) = y_0,$$

Basic idea: From an initial value, integrate one step by tangent direction.

$$y(x_k) = y(x_{k-1}) + (x_k - x_{k-1})y'(x_{k-1}) = y(x_{k-1}) + h_k f(x_{k-1}, y)$$

where the error of k_{th} step, also called the truncation error can be written as:

$$R_k = \int_{x_{k-1}}^{x_k} f(s, y(s))ds - h_k f(x_{k-1}, y(x_{k-1})).$$

where $h_k := (x_k - x_{k-1})$ is step.

例2.4.4 (Euler)**.** Considering the following initial problem:

$$y' = y, y(0) = 1.$$

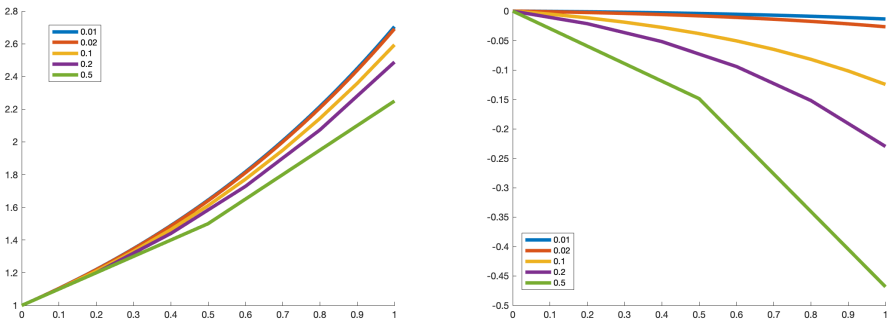


Figure 2.5: (Left)The Euler solution, (Right) Errors

Global and local errors Global error is not the sum of local errors， and it is worth to notice that the sum of local errors are much more large than the global error when the problem is divergence and vice versa.

Implicit Euler method

$$y(x_k) = y(x_{k-1}) + (x_k - x_{k-1})y'(x_k)$$

The method is unconditionally stable, which means independent with step h. The accuracy is just 1 order:

$$\left|\frac{1}{1-h\lambda}\right| = 1 + h\lambda + (h\lambda)^2 + \cdots < 1$$

Trapezoid/central method

$$y(x_k) = y(x_{k-1}) + (x_k - x_{k-1})(y'(x_{k-1}) + y'(x_k))/2$$

This method establishes 2-order convergence：

$$\left|\frac{1+h\lambda/2}{1-h\lambda/2}\right| = 1 + h\lambda + \frac{(h\lambda)^2}{2} + \frac{(h\lambda)^3}{4} + \cdots < 1$$

Runge-Kutta method

Denoting $a_{i,j}, c_i(i-2,3,\ldots,s; j<i)$ and $b_i(i=1,2,\ldots,s)$ as some unknown weights of real number, the s-level explicit Runge-Kutta method is defined as :

$$y_{m+1} = y_m + h(b_1k_1 + \cdots + b_sk_s),$$

where

$$\begin{aligned}
k_1 &= f(x_m, y_m),\\
k_2 &= f(x_m + c_2h, y_m + ha_{2,1}k_1),\\
k_3 &= f(x_m + c_3h, y_m + h(a_{3,1}k_1 + a_{3,2}k_2)),\\
&\cdots\cdots\\
k_s &= f(x_m + c_sh, y_m + h(a_{s,1}k_1 + \cdots + a_{s,s-1}k_s - 1))
\end{aligned}$$

The weights can be computed by Taylor formula. It is simple to find out that the weights are not unique because of the lack of definite condition. Heun and Gill formulas are most useful. Otherwise, we also avoid computing the high-order derivatives while ensuring high accuracy of difference. The most famous numerical methods for ordinary differential equation is the forth order Runge-Kutta formula, which is

$$y_{k+1} = y_k + \frac{h_k}{6}\big(k_1 + 2k_2 + 2k_3 + k_4\big), \tag{2.6}$$

with the definition of momentum

$$\begin{aligned}
k_1 &= f(t_k, y_k),\\
k_2 &= f(t_k + \frac{1}{2}h_k, y_k + \frac{1}{2}h_kk_1),\\
k_3 &= f(t_k + \frac{1}{2}h_k, y_k + \frac{1}{2}h_kk_2),\\
k_4 &= f(t_k + h_k, y_k + h_kk_3),
\end{aligned}$$

例2.4.5 (explicit Runge-Kutta method)**.** Solve the following ODE numerical via Runge-Kutta method:

$$y' = -2ty^2, y(0) = 1$$

Starting from $t_0 = 0$ to $t_1 = 0.25$ with time-step length $h = 0.25$, then we have $k_1 = f(t_0, y_0) = 0, k_2 = f(t_0 + h, y_0 + hk) = -0.5$. ...
It is not difficult to known that the analytic solution for the current problem is $y(t) = \frac{1}{1+t^2}$, so that one can evaluate at the specific time, for example $y(0.25) = 0.9412, y(0.5) = 0.8$, for the purpose of calculating numerical errors. We plot the numerical solution as well as it's error in Fig. 2.4.5

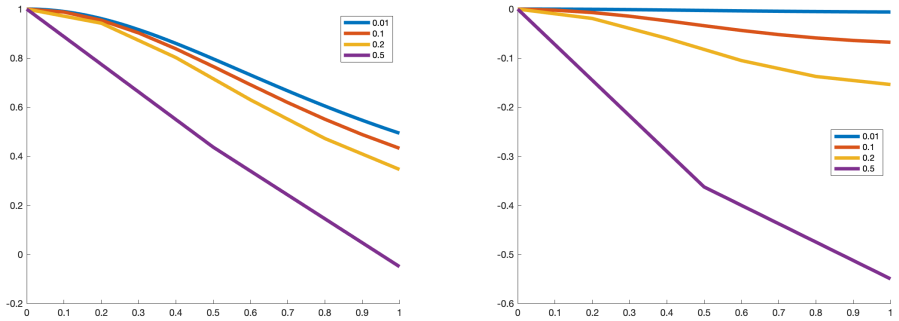


Figure 2.6: (Left)The Runge-Kutta solution with Heun formula, (Right) Errors

2.4.3 Stablity Improvements

implicit scheme

If the computation on k_i has used all values of k as:

$$y_{m+1} = y_m + h(b_1k_1 + \cdots + b_sk_s),$$

where k_i satisfies

$$k_i = f(x_m + c_ih, y_m + h(a_{i,1}k_1 + \cdots + a_{i,s}k_s))$$

The method is called implicit scheme. The common implicit methods are: implicit midpoint formula, Hammer and Hollingsworth formula, Kuntzmann and Butcher formula and so on. Let us present a numerical example as the exercise

例2.4.6 (implicit Runge-Kutta method)**.** Using Hammer method to solve the following ODE:

$$y' = -2xy^2, y(0) = 1$$

properties of one-step methods

It is worth to mention that the above method as following case

$$y_{m+1} = y_m + h\varphi(x_m, y_m, h),$$

is called one-step method, where φ is increment function增量函数。单步法的截断误差可表示为

$$R_m = y_{m+1} - y_m - h\varphi$$

定义2.4.1 (Compatibility)**.** The one-step methods is compatible if the increment function φ satisfies:

$$\varphi(x, y, 0) = f(x, y)$$

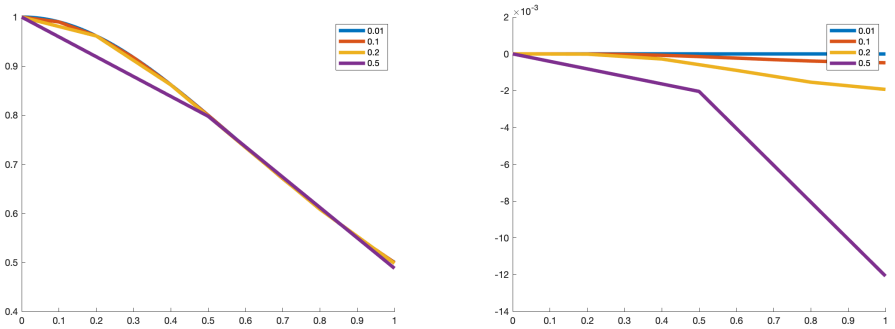


Figure 2.7: (Left)The Runge-Kutta solution with Hammer formula, (Right) Errors

定义2.4.2 (Stability)**.** The one-step methods is stable if for any $(x, y) \in \Omega$ and $h < H$, φ satisfies y's Lipschitz continuous condition.

定义2.4.3 (Convergence)**.** The one-step methods is convergent if there satisfies

$$\lim_{h\to 0} y_m = y(x), x = x_m,$$

moreover, the convergence is equal to compatibility if φ satisfies Lipschitz condition about x and h

Linear Multi-step Method

As the improvements, the multi-step schemes yield larger stable region as well as better precision. Considering problem $y' = f(x, y)$, where $y_m = y(x_m), f_m = f(x_m, y_m)$. We have

$$y_m = \sum_{i=1}^{k-1} \alpha_i y_{m-i} + h \sum_{i=0}^{k} \beta_i y'_{m-i}$$

where $k \in N^+$, $\{\alpha_i\}$ is given real number and h is the time-step length. Then the truncation error is

$$R(x_m, y_m, h) = y_m - \sum_{i=1}^{k-1} \alpha_i y_{m-i} - h \sum_{i=0}^{k} \beta_i y'_{m-i}$$

p-order : the most large number to make $O(h^{p+1})$ established. There are various methods to construct the structure of multi-step methods. Adams extrapolation method is an explicit method and Adams interpolation method is implicit.

例2.4.7 (p-order k-step linear multi-step method)**.** For the scheme

$$y_m = \sum_{i=1}^{k-1} \alpha_i y_{m-i} + h \sum_{i=0}^{k} \beta_i y'_{m-i},$$

we have if $\beta_0 = 0$, the scheme is explicit while $\beta_0 \neq 0$ the scheme is implicit. Let us consider the explicit scheme first, there are $2k-1$ parameters so that we need $\phi_j(x) = x^{j-1}, j = 1, 2, \ldots, 2k-1$.

$$x_m^{j-1} = \sum_{i=1}^{k-1} \alpha_i x_{m-i}^{j-1} + h \sum_{i=1}^{k} \beta_i x_{m-i}^{j-2}, j = 1, 2, \ldots, 2k-1,$$

the matrix form can be written as $Au = b$, where $b_j = x_m^{j-1}$ and

$$A = \begin{bmatrix} 1 & 1 & \ldots & 1 & 0 & 0 & \ldots & 0 \\ x_{m-1} & x_{m-2} & \ldots & x_{m-k+1} & h & h & \ldots & h \\ x_{m-1}^2 & x_{m-2}^2 & \ldots & x_{m-k+1}^2 & hx_{m-1} & hx_{m-2} & \ldots & hx_{m-k} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m-1}^{2k-2} & x_{m-2}^{2k-2} & \ldots & x_{m-k+1}^{2k-2} & hx_{m-1}^{2k-3} & hx_{m-2}^{2k-3} & \ldots & hx_{m-k}^{2k-3} \end{bmatrix}, u = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{k-1} \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \end{bmatrix},$$

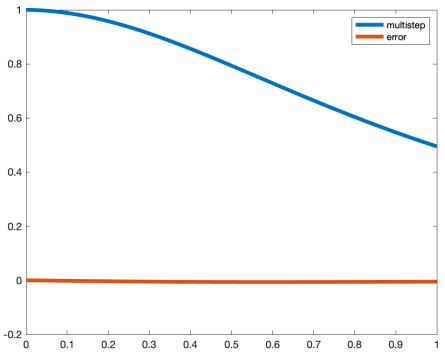
例2.4.8 (Linear multi-step method)**.** Using linear multi-step method to solve the following ODE:

$$y' = -2xy^2, y(0) = 1$$

Using the method in example 2.4.7, we can get one linear multi-step method as:

$$y_{k+1} = y_k + \frac{h}{2}(3y_k' - y_{k-1}')$$

```
1
2  f = @(x,y)-2*x.*y.^2;y0 = 1;y1 = 0.9999;
3  h = 0.01; x = 0:h:1; temp1 = y0; temp2 = y1;
4  y = zeros(1,length(x)); y(1) = y0; y(2) = y1;
5  for j = 3:length(x)
6      y(j) = temp2 + h/2*(3*f(x(j),temp2) - f(x(
           j-1),temp1));
7      temp1 = temp2; temp2 = y(j);
8  end
9  ye = 1./(1+x.^2);
10 plot(x,y,'linewidth',4);
11 hold on
12 plot(x,y-ye,'linewidth',4);
13 legend('multistep','error');
```



Stiffness

The Jacobi matrix of ODE is called stiffness matrix if the eigenvalues of it are very different from each other. Let us consider the following example to show the influence of stiffness matrix in ODE problem.

例2.4.9 (Stiffness)**.** Consider following initial value problem:

$$y' = -100y + 100x + 101, y(0) = 1$$

We use the Euler method, and some noise is added into the initial data. The result is shown in following table:

x	0	0.1	0.2	0.3	0.4
Exact solution	1	1.1	1.2	1.3	1.4
Euler solution	1	1.1	1.2	1.3	1.4
Euler solution	0.99	1.19	0.39	8.59	-64.21
Euler solution	1.01	1.01	2.01	-5.99	67.01

2.5 Numerical Methods for Two-point Boundary Value Problems(BVPs)

In this section, we are concerned with numerical methods for BVPs. As in the previous sections, we adopt a simple form for the purpose of introducing the numerical methods

$$u'' = f(x, u, u'), a < x < b. \tag{2.7}$$

The Dirichlet boundary condition is $u(a) = \alpha, u(b) = \beta$ is considered for convenience. It is worth to mention that the choice of β will play an important role on the uniqueness and stability. A theoretical proof will be more difficult than IVPs, let us explain it simply with an example as following.

例2.5.1. Let $f(x, u, u') = -u$, $u(0) = 0$ in above equation. In this sense, the exact solution of this problem is written as $u(x) = c \sin x$, where c is any constant. It is easy to find out that there are infinite solutions of this boundary problem when $\beta = 0$, and it is unsolvable when $\beta \neq 0$, if $b = n\pi$.

2.5.1 Finite Difference Approximation

Firstly, let us consider the one spatial dimensional BVPs for illustration.

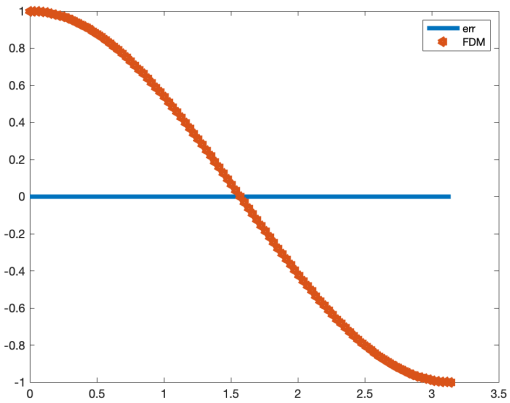
例2.5.2. Consider a uniform grid

$$x_i = a + ih, i = 0, 1, \ldots, n+1$$

for interval $[a, b]$, where $h = (b-a)/(n+1)$ is the grid size. The purpose of numerical solution to two-point boundary value problem

$$u'' = -3u + 2\cos x, u(0) = 1, u(\pi) = -1,$$

is to find approximations $u_i \approx u(x_i), i = 1, \ldots, n$ in the condition of given boundary value $u_0 = u(a) = \alpha$ and $u_{n+1} = u(b) = \beta$. A center divided scheme yields



$$u_{i-1} - (2+3h^2)u_i + u_{i+1} = -2h^2 \cos x_i, i = 1, 2, \ldots, \ldots.$$

The right figure shows the solution for $n = 20$.

2.5.2 Galerkin method

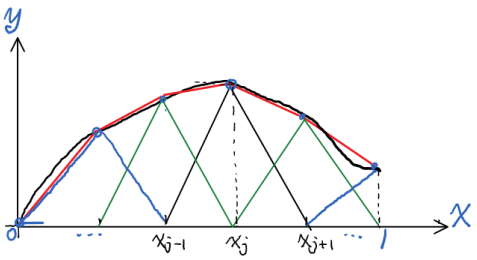
For the ease of representation, let us show how to calculate a finite element solution for a two-point boundary value problem

$$-\frac{d^2}{dx^2}u(x) = f(x), x \in (0, 1), \tag{2.8}$$

with Dirichlet boundary condition $u(0) = u(1) = 0$.

An nonuniform grid $0 = x_0 < x_1 < \cdots < x_{n+1} = 1$ with $n+1$ segment are considered for interval $[0, 1]$, and the length of segment i is defined as $h_i = (x_i - x_{i-1}), \forall i = 1, 2, \cdots, n+1$. Since the essential of numerical approximation is to find a 'closest' solution in certain finite-dimensional function spaces, it is convenient to give basis functions $\phi_k, k = 1, 2, \cdots, n$. Practically, the piecewise continuous linear function space $V_h := \{\phi_k\}_{k=1}^n$ are constantly used, where

$$\phi_k(x) = \begin{cases} \frac{x-x_{k-1}}{x_k-x_{k-1}}, & x \in (x_{k-1}, x_k], \\ \frac{x-x_{k+1}}{x_k-x_{k+1}}, & x \in (x_{k-1}, x_k), \\ 0 & otherwise. \end{cases}$$



In this sense, any function u_h belongs to space V_h can be expressed with combination

$$u_h = \sum_{j=1}^{n} u_j \phi_j(x). \tag{2.9}$$

It is obvious that the Dirchlet boundary condition $u_h(0) = u_h(1) = 0$ are satisfied in this case. The finite element solution to problem (2.8) with the Galerkin's method can be formulated as follows: Find $u_h \in V_h$, such that

$$\left(\frac{d}{dx}u_h, \frac{d}{dx}\phi_i\right) = \big(f, \phi_i\big), \forall i = 1, 2, \cdots, n.$$

where $\big(\cdot, \cdot\big)$ mean the inner-product defined on V_k. By substituting (2.9) into the above equation, it leads to a finite element system

$$\begin{bmatrix} K_{11} & \cdots & K_{1n} \\ \vdots & \vdots & \vdots \\ K_{n1} & \cdots & K_{nn} \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix} \tag{2.10}$$

with $K_{ij} = \big(\frac{d}{dx}\phi_i, \frac{d}{dx}\phi_j\big) := \int_0^1 \frac{d}{dx}\phi_i(x)\frac{d}{dx}\phi_j(x)dx$ and $f_i = \big(f, \phi_i\big) := \int_0^1 f(x)\phi_i(x)dx$.

For the ease of understanding, we write out the above procedure within a concise script, which runs on Matlab or Octave platform. We emphasis that the script is written in a general manner of element-by-element assembling, and more efficiency could be expected by utilizing vectorized. However, there is always a balance between the running efficiency and the ease of understanding.

```
%% − u''(x) = f(x), on (0,1); with u(a)=ua, u(b)=ub.
a=0; b=1; u = @(x) sin(pi*x); f = @(x) pi*pi*sin(pi*x); ua=u(a); ub=u(b);% EXE 1
n_elem = 16;  n = n_elem; x = a + (0:n)*(b − a)/n;                    % the mesh grid
xi = [−0.577350269; 0.577350269]; wi = [0.5;  0.5];                   % the quadurature
K = zeros(n+1, n+1);  rhs = zeros(n+1, 1);  K_temp =[2, −1; −1, 2];   % allocate memory
for e = 1:n
    hi = x(e+1)−x(e); K(e:e+1,e:e+1) = K(e:e+1,e:e+1) + K_temp/hi;    % assembling
    qx = x(e)*0.5*(1−xi) + x(e+1)*0.5*(1+xi);                         % xi's real coord
    rhs(e)  = rhs(e)  + sum(2.0*hi*wi.*f(qx).*(x(e+1)−qx)/hi);        % force node 1
    rhs(e+1)= rhs(e+1) + sum(2.0*hi*wi.*f(qx).*(qx − x(e))/hi);       % force node 2
end
K(1,1) = 1.0; K(1,2) = 0; rhs(1) = ua;                                % apply u(a) = ua
K(n+1, n) = 0; K(n+1, n+1) = 1.0; rhs(n+1) = ub;                      % apply u(b) = ub
u_h = K\rhs;                                                          % solving Ax = b
nn = 100; xx = a+(0:nn)/nn*(b−a); plot(xx,u(xx),'k−',x,u_h,'ro'); grid on;
```

Noticing that the basis functions $\phi_k(x), \forall k = 1, \cdots, n$ are all locally supported by their piecewise definition, it is obvious that $a_{ij} = 0$ if $|i - j| \geq 2$. So that the pattern of

coefficient matrix K of the finite element system (2.10) will be

$$\begin{bmatrix} * & * & 0 & 0 & \cdots & 0 \\ * & * & * & 0 & \cdots & 0 \\ 0 & * & * & * & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & * & * \end{bmatrix} \tag{2.11}$$

So that the sparse matrix operator will improve this procedure dramatically, for examples, the tri-diagonal solver mentioned in the first lecture is the most efficient one. Let us leave it as an exercise to interested readers, and the sparse structure in the next section when presenting the multi-dimensional cases.

2.5.3 Collocation Method

The main idea of collocation method is to find a solution $u(x) = \sum_{i=1}^{n} u_i \phi_i(x)$, satisfies

$$u'' = f(x, u, u'), u(a) = \alpha, u(b) = \beta, a < t < b,$$

where ϕ_i is basic function in interval $[a, b]$, U_i is undetermined coefficient, which yields

$$\sum_{i=1}^{n} u_i \phi_i''(x_i) = f(x_i, \sum_{i=1}^{n} u_i \phi_i(x_i), \sum_{i=1}^{n} u_i \phi_i'(x_i)), i = 2, \ldots, n-1.$$

And following the boundary condition, there yields

$$\sum_{i=1}^{n} u_i \phi_i(a) = \alpha, \sum_{i=1}^{n} u_i \phi_i(b) = \beta$$

By rewriting the problem as: $\mathcal{L}u = g(x)$, the matrix form can be written as the form of $Aw = b$:

$$A = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \ldots & \phi_n(x_1) \\ \mathcal{L}\phi_1(x_2) & \mathcal{L}\phi_2(x_2) & \ldots & \mathcal{L}\phi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{L}\phi_1(x_{n-1}) & \mathcal{L}\phi_2(x_{n-1}) & \ldots & \mathcal{L}\phi_n(x_{n-1}) \\ \phi_1(x_n) & \phi_2(x_n) & \ldots & \phi_n(x_n) \end{bmatrix}, w = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}, b = \begin{bmatrix} \alpha \\ g(x_2) \\ \vdots \\ g(x_{n-1}) \\ \beta \end{bmatrix},$$

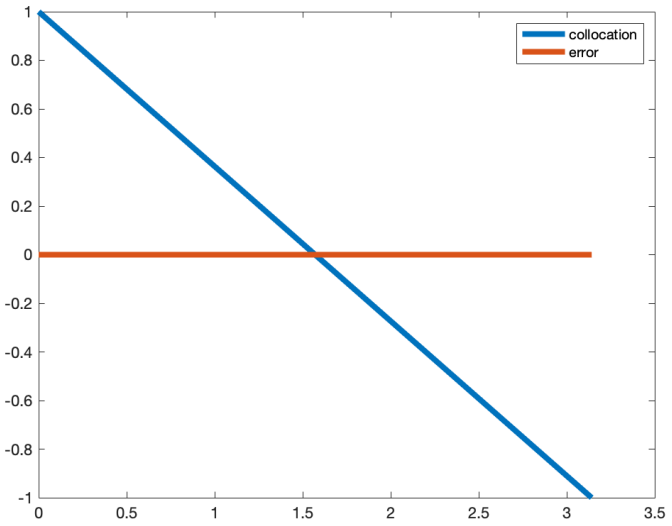
and the solution is computed as $u = Kw$, where

$$K = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \ldots & \phi_n(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \ldots & \phi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_n) & \phi_2(x_n) & \ldots & \phi_n(x_n) \end{bmatrix},$$

例2.5.3 (collocation)**.** Using the method of collocation so solve the following boundary problem:

$$u'' = -3u + 2\cos t, u(0) = 1, u(\pi) = -1,$$

by choosing $h = \frac{\pi}{2}$, the solution are shown as following:



2.5.4 Shooting Method

The fundamental idea of the shooting method is to fully utilizing the efficiency of explicit scheme. Assuming that the boundary value problem is

$$y' = f(x, y), a < t < b$$

s.t.

$$g(y(a), y(b)) = 0$$

It is obvious that the problem is equal to

$$h(x) := g(x, y(b; x)) = 0$$

例2.5.4 (shooting)**.** Using the method of shooting so solve the following boundary problem:

$$u'' = -3u + 2\cos t, u(0) = 1, u(\pi) = -1,$$

Assume that $u_1 = u', u_2 = u_1', u_2(0) = \beta$, and the first guess is $\beta = 0.1$

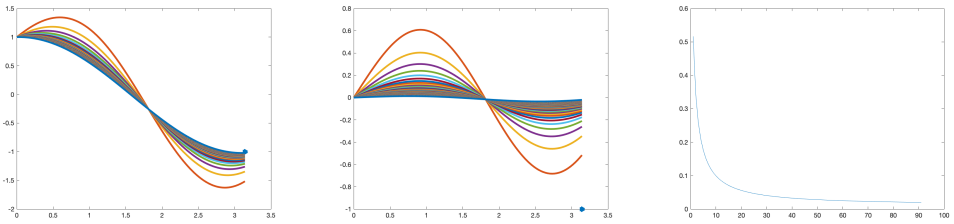


Figure 2.8: (Left)The solution of shooting method, (Middle) Errors, (Right) the objective function

2.5.5 Newton's method for nonlinear problems

Consider the following semi-linear problem:

$$-\triangle u + f(\mathbf{x}, u) = g, \quad \mathbf{x} \in \Omega \subset \mathbf{R}^2 \tag{2.12}$$

Set

$$f(\mathbf{x}, u) = u^3. \tag{2.13}$$

The the equation is constructed on all the inner points$i, j = 1, \cdots, n-1$

$$\frac{1}{h^2}\begin{pmatrix} & -u_{i,j+1} & \\ -u_{i-1,j} & 4u_{i,j} & -u_{i+1,j} \\ & -u_{i,j-1} & \end{pmatrix} + f(x_i, y_j, u_{i,j}) = g_{i,j}.$$

If there is not reaction-diffusion problem in f, $u_{i,j}$ is just consisted of f. Denoting the above equation as

$$A\mathbf{u} + f(\mathbf{u}) = \mathbf{g} \tag{2.14}$$

Assuming that $F(\mathbf{u}) = A\mathbf{u} + f(\mathbf{u}) - \mathbf{g}$, there yields Newton iteration as

$$\mathbf{u}^{new} = \mathbf{u}^{old} - F'(\mathbf{u}^{old})^{-1} F(\mathbf{u}^{old}) \tag{2.15}$$

where $F'(\mathbf{u}) = A + f'(\mathbf{u})$. The corresponding code in Matlab will bring some more parameters， more details can be founded in test_semi_newton.m

Chapter 3

Numerical Solution to Partial Differential Equations

Equations involving partial derivatives of an unknown funciton is named as Partial Differential Equations(PDEs). PDEs are the fundamental mathematical technique for modeling contimuous phenomena in many physical problem, such as linear elasticity equation, Maxwell's equation, Navier-Stokes equaitons, Schŕodinger's equation and Einstein's equations of general relativity, etc. All of the above models are the most famous one in various scientific research fields.

For the purpose of a clear concept of the numerical aspect of the approximation, we just consider some basic PDEs including *Parabolic* PDEs, such as *heat equation*

$$u_t = \nu u_{xx},$$

describe time-depedent and dissipative process which are volving toward a steady state, *Hyperbolic* PDEs, such as *wave equation*

$$u_{tt} = u_{xx},$$

describe time-depedent and conservative physical process which are not evolving toward a steady state, and *Elliptic* PDEs describe systems that have already reached a steady state or equilibrium, such as *Laplace equation*

$$u_{xx} + u_{yy} = 0.$$

Finite difference method(FDM) is the first method in the field of computer numerical simulation, and has already used in various fields up to now. In this method, domain is divided into difference grids, and the continuous domain is replaced by finite grid nodes. Taylor series expansion and other methods are used in FDM, in which the derivative in the control equation is replaced by the difference quotient of the function values for discretization so that the algebraic equations can be established with the unknown value

on the grid nodes. FDM is a direct numerical method to transfer the differential problem into an algebraic equations, and the mathematical concept and representation are obvious. It is a relatively mature numerical method which has been studied earlier.

As for the finite difference scheme, there are first-order scheme, second-order scheme and high-order scheme in the considering of accuracy of schemes. Considering the spatial form of the difference, it can be divided into the central scheme and the upwind scheme. Moreover by considering the influence of time, the difference schemes can also be divided into explicit scheme, implicit scheme, explicit-implicit scheme and so on. Recently, the common difference schemes are mainly combined of the above forms and different combinations construct different difference schemes. The difference method is mainly used on the structured grid, and the step size of the grid is determined according to the actual terrain and the Koran stable condition.

There are many ways to construct the difference and the main method is to use Taylor series. There are three kinds of basic difference expressions: first-order forward difference, first-order backward difference and first- or second- order center difference. Among those schemes, the first two are first-order accuracy and the last one are second-order. Various difference schemes can be constructed by the different combination of time and space schemes.

3.1 Parabolic Equations for Initial Value Problems(IVPs)

3.1.1 Explicit(iterative) numerical scheme

Let us consider a simple parabolic PDE related with time t and one spatial dimension x

$$u_t = \nu u_{xx}, \quad 0 < x < 1, \quad t > 0, \tag{3.1}$$

$$u(0,t) = u(1,t) = 0, \qquad t \geq 0, \tag{3.2}$$

$$u(x,0) = u^0(x), \qquad 0 \leq x \leq 1. \tag{3.3}$$

From physical point of view, this equation modeling the no-source heat diffusion on interval $[0,1]$ with homogeneous media. In this simple case, homogeneous Dirichlet boundary condition is imposed. Mathematically, the solution for equation (3.1) could be obtained by separation of variable. Assuming the solution be in the form of $u(x,t) = f(x)g(x)$, it is true that

$$u(x,t) = \sum_{m=1}^{\infty} a_m e^{-(m\pi)^2 t} \sin m\pi x, \tag{3.4}$$

where a_m is the Fourier coefficients

$$a_m = 2\int_0^1 u^0(x) sin m\pi x dx.$$

Since there is a finite integer m, (3.4) is basically a good approximation to the analytic solution, however, this strategy is hard to applied in more general partial differential equations.

In many applications, it is sufficient to obtain the solution at discrete spatial point x_i and certain time step t_n. For the ease of representation, let h and τ be the spacing, then $x_j = jh, t_n = n\tau$. So that

$$\begin{aligned}\frac{\partial u}{\partial t}(x_j,t_n) &\approx \frac{u(x_j,t_{n+1})-u(x_j,t_n)}{\tau} := \frac{U_j^{n+1}-U_j^n}{\tau}\\ \frac{\partial^2 u}{\partial x^2}(x_j,t_n) &\approx \frac{u(x_{j+1},t_n)-2u(x_j,t_n)+u(x_{j-1},t_n)}{(h)^2} := \frac{U_{j+1}^n-2U_j^n+U_{j-1}^n}{(h)^2}\end{aligned}$$

With the above notations, equation (3.1) holds at (x_j, t_n), which means spatial point x_j at time step t_n

$$U_j^{n+1} = U_j^n + \mu(U_{j+1}^n - 2U_j^n + U_{j-1}^n), \tag{3.5}$$

where $\mu = \nu\frac{\tau}{h^2}$. One can obtain the approximated solution iteratively at the next time step t_{n+1} by knowing $U_j^n, \forall j$. In this sense, the iterative scheme (3.5) is referred as an **explicit Scheme**.

Note: we will always use matlab script to describe the algorithm.

```
Given ν, f, [a, b] and N, T, τ;
h = (b − a)/N and set x_j = j ∗ h, ∀j = 0, 1, · · · , N;
u = zeros(N+1,T+1) ;
for n = 1, 2, · · · , T do
    u(0, n) = a(nτ); u(N, n) = b(nτ);
    for j = 1, 2, · · · , N − 1 do
        U_j^{n+1} = U_j^n + μ(U_{j+1}^n − 2U_j^n + U_{j−1}^n);
    end
end
```

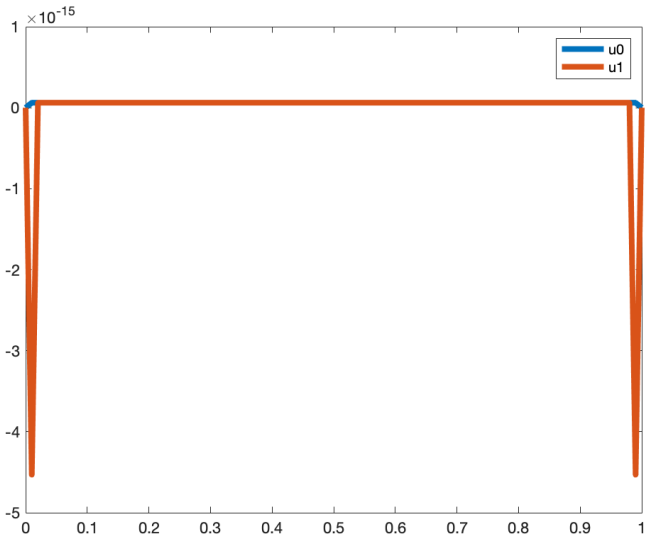
例3.1.1. Please calculate by (3.5) with $\nu = 5, f(x,0) = \cos\frac{\pi x}{2}, a(0,t) = 0, b(1,t) = 0$. Suggested computational parameters $N = 100, T = 1, \tau = 0.0015$. The result is shown as following figure

3.1.2 Error estimation for explicit scheme

Does we do the right thing to solve the parabolic equation (3.1)?

定理3.1.1 (Consistency)**.** Let $L = \frac{\partial}{\partial t} - \nu\frac{\partial^2}{\partial x^2}, (\nu > 0)$ be the operator and $U_j^{n+1} = L_h U_j^n$ be the finite difference scheme, where L_h dependent on the time and space step τ and h. It is defined that the finite difference scheme is consistent with the original differential equation, if

$$T(x_j,t_n) = \big(L_h u(x_j,t_n) - u(x_j,t_{n+1})\big) \to 0, \qquad \tau, h \to 0.$$



Truncation Error:

$$\begin{aligned} T(x,t) &= \frac{u(x,t+\tau)-u(x,t)}{\tau} - \nu\frac{(u(x+h,t)-2u(x,t)+u(x-h,t))}{h^2} \\ &= (u_t(x,t)+\frac{\tau}{2}u_{tt}(x,t)+\ldots) - \nu(u_{xx}+\frac{h^2}{12}u_{xxxx}+\ldots) \\ &\approx \frac{\tau}{2}u_{tt}(x,t) - \frac{\nu h^2}{12}u_{xxxx} \end{aligned}$$

Convergence: Is $U_j^n \to u(x_j,t_n)$?

定理3.1.2 (Convergent)**.** Using fixed initial and boundary values and $\mu = \tau/(h)^2$, and let $\tau \to 0, h \to 0$. If on any given position $(x^*,t^*) \in (0,1)\times(0,T)$,

$$U_j^n \to u(x_j,t_n), \forall x_j \to x^*, t_n \to t^*.$$

It is essential to calculate the **Approximation Error:** $e_j = U_j^n - u(x_j,t_n)$ to evaluate the quality of approximation. In this sense, the finite difference scheme $T(x,t)$ yields

$$e_{j+1} = (1-2\mu)e_j^n + \mu e_{j+1}^n + \mu e_{j-1}^n - T_j^n\tau,$$

which result in $E^n \le \frac{1}{2}\tau\big(M_{tt}+\frac{1}{6\mu}M_{xxxx}\big)$ if define $E^n = \max\{|e_j|, j=0,1,\cdots,n\}$ and M_{tt} and M_{xxxx} be the upper limit for u_{tt} and u_{xxxx} respectively.

性质3.1.1 (Stability Condition)**.** The explicit scheme (3.5) is convergent if $\mu := \frac{\tau}{h^2} \le \frac{1}{2}$.

3.1.3 Implicit schemes

The stability condition $\mu = \frac{\tau}{h^2} \leq \frac{1}{2}$ is too strict, which means too small time step $\tau \leq \frac{1}{2}h^2$ when the grid space $h \to 0$. The following scheme is another good choice

$$U_j^{n+1} = U_j^{n+1} + \mu(U_{j+1}^{n+1} - 2U_j^{n+1} + U_{j-1}^{n+1}) \tag{3.6}$$

The implicit scheme yields

$$-\mu U_{j-1}^{n+1} + (1+2\mu)U_j^{n+1} - \mu U_{j+1}^{n+1} = U_j^n, \qquad \forall j = 1, 2, \ldots, (N-1).$$

U_0^{n+1} and U_N^{n+1} are known with the boundary condition. It requires to solve a linear equation system with a tri-diagonal coefficient matrix. **Thomas algorithm** is the natural choice.

例3.1.2 (Fourier mode error estimation)**.** Error analysis for explicit scheme (3.5) with Fourier mode

$$U_j^n = (\lambda)^n e^{ik(jh)}.$$

It yields

$$\begin{aligned} \lambda &:= \lambda(k) = 1 + \mu(e^{ikh} - 2 + e^{-ikh}) \\ &= 1 - 2\mu(1 - cos(kh)) \\ &= 1 - 4\mu sin^2\frac{1}{2}kh \end{aligned}$$

Since $U_j^{n+1} = \lambda U_j^n$, λ is referred as **amplification factor**. In the mode $k = m\pi$, $\mu > \frac{1}{2}$ makes $\lambda > 1$, which make the iterative scheme (3.5) divergent! However in the case of convergence, there exist a K independent of k, which makes

$$|[\lambda(k)]^n| \leq K, \qquad \forall k, n\tau \leq T.$$

By applying the analysis on Fourier mode $U_j^n = (\lambda)^n e^{ik(jh)}$, it is trivial to verify the stability of the implicit scheme. We have

$$\lambda = \frac{1}{1 + 4\mu \sin^2 \frac{1}{2}kh} < 1,$$

which imply that the implicit scheme (3.6) is **unconditionally stable**. This mean that the time-step length could be much larger than the explicit scheme, however, its truncation error is still the same with explicit scheme,

3.1.4 Mixed method: θ-scheme

It is straightforward to consider a mixed/weighted version between the explicit one and the implicit one, in order to take full advantages in their own property. A general formulation looks like

$$U_j^{n+1} - Y_j^n = \mu\big[\theta\delta_x^2 U_j^{n+1} + (1-\theta)\delta_x^2 U_j^n\big], \qquad \forall j = 1, 2, \ldots, J-1$$

As for any $\theta \neq 0$, straightforward simplification results into a tri-diagonal linear system

$$-\theta\mu U_{j-1}^{n+1} + (1+2\theta\mu)U_j^{n+1} - \theta\mu U_{j+1}^{n+1} = \big[1 + (1-\theta)\mu\delta_x^2\big]U_j^n \tag{3.7}$$

例3.1.3 (Crand-Nickson)**.** There are many other stencils to solve the heat equation, which is mentioned in the text book of Richtmyer and Morton(1967). However, a most popularly stencil is the case of $\theta = \frac{1}{2}$, which is proposed in 1947 and is named after the authors.

The stability region are interested for θ-scheme. Considering the Fourier mode $U_j^n = \lambda^n e^{ik(j\triangle x)}$ as a solution of scheme (3.7).It finally leads to

$$\mu(1-2\theta) > \frac{1}{2}.$$

According to the above criteria, the θ-scheme for (3.7) is stable if $\mu \leq \frac{1}{2}\big(1-2\theta\big)^{-1}$ in the case of $0 \leq \theta < \frac{1}{2}$, and is stable for any given μ in the case of $\frac{1}{2} \leq \theta \leq 1$.

定理3.1.3 (Maximum value principle)**.** Assuming that $0 \leq \theta \leq 1$ and $\mu(1-\theta) \leq \frac{1}{2}$ hold for the parameters of the θ-method, then $\{u_j^n\}$, the solution of (3.7), satisfying

$$U_{min} \leq U_j^n \leq U_{max}, \tag{3.8}$$

where U_{min} and U_{max} are the minimum and maximum value among all initial position and boundary position.

General boundary conditions can also be treated in discrete. As for the Robin type

$$\frac{\partial u}{\partial x} = \alpha(t)u + g(t), \alpha(t) > 0, x = 0,$$

A first order scheme $\frac{U_1^n - U_0^n}{h} = \alpha^n U_0^n + g^n$ is sufficient in most of the applications. Second order scheme $\frac{2U_0^n - 3U_1^n + U_2^n}{h} = \alpha^n U_0^n + g^n$ is also constantly used for better resolutions.

例3.1.4 (Nonlinearity)**.** As for the nonlinear case, such as

$$u_t = b(u)u_{xx}, \forall x \in (0,1)$$

The linearization is necessary at each time step

$$U_j^{n+1} = U_j^n + \mu' b(U_j^n)(U_{j+1}^n - 2U_j^n + U_{j-1}^n)$$

The error analysis at each step is similar with the linear case.It is very hard to obtain a general global error analysis, which is dependent heaily on $b(u)$.

In practical applications, multiple spatial dimension are constantly concerned. Let Ω be a rectangular domain $(0, X) \times (0, Y)$
Find a function $u(x, y, t)$ defined on Ω

$$\begin{aligned} u_t(x, y, t) &= b(u_{xx}(x, y, t) + u_{yy}(x, y, t)), \qquad (b > 0) \\ &:= b\triangle u(x, y, t) := b\nabla^2 u(x, y, t), \end{aligned}$$

with proper Dirichlet boundary condition and initial value $u(x, y, 0)$

Explicit V.S. Implicit time step $\triangle t$, grid space $\triangle x$ and $\triangle y$

$$U^n_{r,s} \approx u(x_r, y_s, t_n), \qquad \forall r = 0, \cdots, Nx, s = 0, \cdots, Ny.$$

Explicit scheme

$$\frac{1}{\triangle t}\big(U^{n+1}_{r,s} - U^n_{r,s}\big) = \frac{b}{(\triangle x)^2}\big(U^n_{r+1,s} - 2U^n_{r,s} + U^n_{r-1,s}\big) - \frac{b}{(\triangle y)^2}\big(U^n_{r,s+1} - 2U^n_{r,s} + U^n_{r,s-1}\big)$$

Implicit scheme(**Jacobi** and **Gauss Sediel** solver)

$$\frac{1}{\triangle t}\big(U^{n+1}_{r,s} - U^n_{r,s}\big) = \frac{b}{(\triangle x)^2}\big(U^{n+1}_{r+1,s} - 2U^{n+1}_{r,s} + U^{n+1}_{r-1,s}\big) - \frac{1}{(\triangle y)^2}\big(U^{n+1}_{r,s+1} - 2U^{n+1}_{r,s} + U^{n+1}_{r,s-1}\big)$$

To solve the multiple-dimensional problem in an efficient manner, the Alternative Direction Interaction(ADI) method is proposed by Peaceman D.W. and Rachford H.H. Jr in 1955. It is convenient to take the two dimensional as illustration. Consider the Crank-Nicolson scheme

$$(1 - \frac{1}{2}\mu_x\delta_x^2 - \frac{1}{2}\mu_y\delta_y^2)U^{n+1} = (1 + \frac{1}{2}\mu_x\delta_x^2 + \frac{1}{2}\mu_y\delta_y^2)U^n,$$

a slight modification is made by approximation

$$(1 - \frac{1}{2}\mu_x\delta_x^2)(1 - \frac{1}{2}\mu_y\delta_y^2)U^{n+1} = (1 + \frac{1}{2}\mu_x\delta_x^2)(1 + \frac{1}{2}\mu_y\delta_y^2)U^n$$

which leads to a two step method

$$\begin{aligned} (1 - \frac{1}{2}\mu_x\delta_x^2)U^{n+\frac{1}{2}} &= (1 + \frac{1}{2}\mu_y\delta_y^2)U^n \\ (1 - \frac{1}{2}\mu_y\delta_y^2)U^{n+1} &= (1 + \frac{1}{2}\mu_x\delta_x^2)U^{n+\frac{1}{2}}. \end{aligned}$$

It is essentially an type of operator splitting technique.

例3.1.5 (改成字母Z，并给出matlab脚本)**.** Consider a two-dimensional hear equation on the unit square

$$u_t = u_{xx} + u_{yy}, \qquad (x, y) \in (0, 1) \times (0, 1),$$

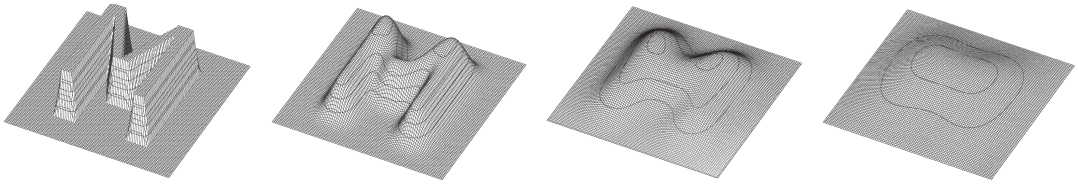


Figure 3.1: From left to right: the numerical solution at $t = 0, 0.001, 0.004$ and $t = 0.01$

with $u = 0$ are fixed on all the four boundaries and the initial value $u(x, y, 0) = u^0(x, y)$ can be arbitrary function, such as the first plot in Fig.3.1.

Standard explicit and implicit scheme as well as the ADI iterative method can also be applied. In numerical tests, $\triangle t = 0.001$ and $\triangle x = \triangle y = 0.01$ are used. Plot of the solution are listed in the following for the ease of comparison: Different value of grid size $\triangle x$ and $\triangle y$, for e.g. $\frac{1}{100}, \frac{1}{200}, \frac{1}{400}$ could be used for pricision test.

3.2 Hyperbolic Equation-based Initial Value Problems(IVPs)

Let us introduce the basic idea of numerical methods for hyperbolic equation-based IVPs. One dimensional linear convection equation is convenient for this purpose

$$\begin{cases} \frac{\partial u}{\partial t} + a(x,t)\frac{\partial u}{\partial x} = 0, & \forall t, x, \\ u(x,0) = u^0(x), & \forall x. \end{cases} \tag{3.9}$$

It is trivial to verify that u is a series of constants along the family of characteristic curves $x(t)$ satisfying

$$\frac{dx}{dt} = a(x,t).$$

Supposing that $a(x,t)$ is Lipschitz continuous with respective to x and continuous with respective to t, there will be no intersection between different $x(t)$. In this sense, the so-call *method of characters* gives the analytic solution

$$u(x,t) = u_0(x - a(x,t)t) := u_0(x - at).$$

3.2.1 Upwind scheme

It is necessary to develop numerical methods to evaluate the solution on fixed grid, since most of other cases could not be solved with the above strategy. Let U_j^n be the solution on point x_j at time step t_n, which is constantly denoted as the grid point (x_j, t_n). It is worth to mention that the very first concept is proposed in a famous paper on finite difference methods by Courant, Friedrichs and Lewy in 1928.

By doing finite difference approximations to the derivatives in (3.9), which yields finite difference equaiton

$$\frac{U_j^{n+1}-U_j^n}{\triangle t}+a\frac{U_j^n-U_{j-1}^n}{\triangle x}=0.$$

A simplification results into a iterative scheme

$$U_j^{n+1}=U_j^n-\frac{a\triangle t}{\triangle x}\big(U_j^n-U_{j-1}^n\big):=(1-\nu)U_j^n+\nu U_{j-1}^n, \tag{3.10}$$

where $\nu=\frac{a\triangle t}{\triangle x}$. From algebraic point of view, iterative scheme (3.10) is fundamentally a combination of certain neighbored grid point. As well as in the parabolic equation, it is referred as an explicit scheme to solve the linear convection equation. It is essential to mention that a necessary condition for the convergence of (3.10) is $\nu\leq 1$.

For the ease of analysis, it is preferred to define $\nu=|a|\triangle t/\triangle x$ as the *CFL*-number. Since we do not known the direction of characteristic curve at (x_j,t_n), which is determined by the sign of a. A classical compact and stable finite difference scheme is the *upwind scheme*

$$U_j^{n+1}=\begin{cases} U_j^n-\nu\big(U_{j+1}^n-U_j^n\big)=(1+\nu)U_j^n-\nu U_{j+1}^n, & a<0,\\ U_j^n-\nu\big(U_j^n-U_{j-1}^n\big)=(1-\nu)U_j^n+\nu U_{j-1}^n, & a>0,\end{cases} \tag{3.11}$$

3.2.2 Lax-Wendroff Scheme

The phase error of the upwind scheme is in fact smaller than high order schemes, however, the disappation error are too sevier to prevent it from practical using. Remind that larger stencil will leads to better approximation in the interpolation theory, it is natural to reconstruct a better approximation U using quadratic interpolation.

The Lax-Wendroff scheme has been the most intrisic scheme since it is proposed in 1960. Let us skip the details and only gives its formulation

$$U_j^{n+1}=\frac{1}{2}\nu(1+\nu)U_{j-1}^n+(1-\nu^2)U_j^n-\frac{1}{2}\nu(1-\nu)U_{j+1}^n. \tag{3.12}$$

The most significant advantage is its convenience in generalization to solving hyperbolic systems, and let us illustrated in the chapter of the finite element methods. Here we are intend to emphasis the stability of the scheme, which is independent on the sign of its Fourier modes Fourier mode analysis gives its amplification factor being

$$|\lambda(k)|^2=1-4\nu^2(1-\nu^2)\sin^4\frac{k}{2}\triangle x.$$

It shows that the Lax-Wendroff scheme is stable if $\nu\leq 1$.

例3.2.1. Solve the linear convection equation with upwind scheme and Laxwendroff scheme. Please show the numerical results under different initial conditions, including the impulse function and a smooth one

$$u(x,0)=\exp -10(4x-1)^2,$$

3.2.3 Leap-Frog scheme

In this scheme, three different time-levels are concerned, and it is another important scheme other than the Lax-Wendroff scheme.

$$\frac{U_j^{n+1} - U_j^{n-1}}{2\triangle t} + a\frac{U_j^{n+1} - U_{j-1}^n}{2\triangle x} = 0.$$

A simplification results into a iterative scheme

$$U_j^{n+1} = U_j^{n-1} - \nu\big(U_{j+1}^n - U_{j-1}^n\big), \tag{3.13}$$

where $\nu = a\triangle t/\triangle x$ is the same as usual.

Hyperbolic equation system.

例3.2.2. Solve the second order partial differential equation $u_{tt} = a^2 u_{xx}$, with discontinuous and smooth initial conditions, as the same as given in the previous example.
Solution: Firstly, convert the second order equation into a first order system

$$\begin{aligned} v_t + aw_x &= 0, \\ w_t + av_x &= 0. \end{aligned}$$

As for a two-component linear system, a stagger-grid/two-step scheme is preferred for practical reason

$$\begin{aligned} V_j^{n+1/2} &= V_j^{n-1/2} - \nu\big(W_{j+1/2}^n - W_{j-1/2}^n\big), \\ W_{j+1/2}^{n+1} &= W_{j+1/2}^n - \nu\big(V_{j+1}^{n+1/2} - V_j^{n+1/2}\big). \end{aligned} \tag{3.14}$$

3.2.4 Numerical analysis for the methods

Different with those for solving parabolic equations, explicit numerical schemes are widely interested for hyperbolic equations.

Convergence

We say that the numerical solution for a hyperbolic equation is convergent in the meaning of $\triangle x \to 0$ and $\triangle t \to 0$, it requires

The method be *consistent*, which promises the local truncation error goes to 0 as $\triangle t \to 0$. The method be *stable*, which means any small error in each timestep is under control(will not grow too fast)

Consistency

Denote the numerical method as $A^{n+1} = \mathcal{N}(Q^n)$ and the exact value as q^n and q^{n+1}. Then the local truncation error is defined as

$$\tau = \frac{\mathcal{N}(q^n) - q^{n+1}}{\triangle t}$$

We say that the method is *consistent* if τ vanished as $\triangle t \to 0$ for all smooth $q(x,t)$ satisfying the differential equation. It is usually stratightforward when Taylor expasions are used.

Stability

Courant-Friedrichs-Levy condition: the numerical domain of dependence contains the true domain of dependence domain of the PDE, at least in the limit as $\triangle t, \triangle x \to 0$

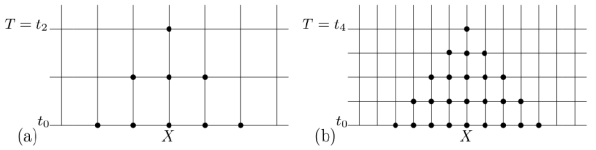


Fig. 4.3. (a) Numerical domain of dependence of a grid point when using a three-point explicit finite difference method, with mesh spacing Δx^a. (b) On a finer grid with mesh spacing $\Delta x^b = \frac{1}{2}\Delta x^a$.

For a hyperbolic system with characterestic wave speeds λ^p,

$$\frac{\triangle x}{\triangle t} \geq \max_p |\lambda^p|, \qquad p = 1, \cdots, m.$$

This condition is necessary but not sufficient !

定理3.2.1 (Lax Equivalence Theorem)**.** For a consistent difference approximation to a well-posed linear evolutionary problem, which is uniformly solvable in the sense of $\|B^{-1}\| \leq K_1 \triangle t$, where $B = \frac{1}{\triangle t} - \theta \frac{\delta_x^2}{(\triangle x)^2}$, the stability of the scheme is necessary and sufficient for convergence.

定理3.2.2 (Von Neumann Theorem)**.** A necessary condition for stability is that there exist a constant K' such that

$$|\lambda(\mathbf{k})| \leq 1 + K' \triangle t, \forall \mathbf{k}, \tag{3.15}$$

for every eigenvalue $\lambda(\mathbf{k})$ of the amplification matrix $G(\mathbf{k})$

where $u^{n+1}(\mathbf{k}) = G(\mathbf{k}) u^n(\mathbf{k})$.

3.3 FVMs for One Spatial Dimension

The finite volume method (FVM) is a constantly used discretization strategy for hyperbolic type partial differential equations, especially in the application of conservation law and other models arising from computational fluid dynamics.

FVMs seem to be perfectly suited to the conservation or divergence form of partial differential equations since they appear automatically in conservation form. Most of the numerical methods for conservation law can be category into FVMs. Let us take the following general form

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{f}(\mathbf{u})}{\partial x} = 0, \tag{3.16}$$

where $\mathbf{u} = \mathbf{u}(x,t)$ is the unknown vector-valued function and $\mathbf{f}(\mathbf{u})$ is referred as the *flux function*. It is convenient to define $\mathbf{F}(x,t) := \mathbf{f}(\mathbf{u})(x,t)$, and we shall omit the dependency on variable x and t if there is no confusing will arise, say, $\mathbf{F} = \mathbf{f}(\mathbf{u})$ as the flux function.

3.3.1 Finite Volume Formulas

The fundamental idea for Finite Volume Method(FVM) is to divide the domain into grid cells and then approximate the total integral of the flux over each grid cell. Denote cells $C_i = (x_{i-1/2}, x_{i+1/2})$ and mean values on cells

$$U_i^n \approx \frac{1}{|C_i|} \int_{x_{i-1/2}}^{x_{i+1/2}} u(x,t_n)dx.$$

In this sense, FVM update the value of U_i^{n+1} based on the fluxes F^n between the cells, which is shown in Fig.3.3.1.

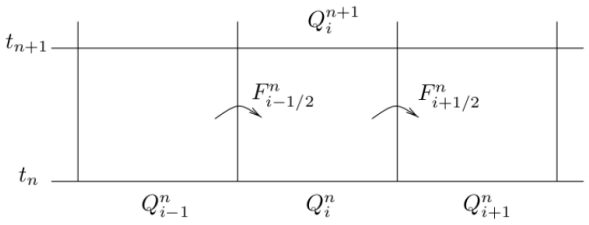


Figure 3.2: The concept of finite volume approximation.

Let us consider the FVM scheme for 1D conservation law. Remember that the FV approximation is performed in a cell-wise manner, then on any given cell $C_i := [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$, one can integrate CLAW (3.16) on cell, it yields

$$\frac{d}{dt} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \mathbf{u}(x,t)dx = \mathbf{f}(\mathbf{u}(x_{i-\frac{1}{2}},t)) - \mathbf{f}(\mathbf{u}(x_{i+\frac{1}{2}},t)).$$

时间方向从t_n到t_{n+1}积分后同除以$\triangle x$:

$$\frac{1}{\triangle x}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}}\mathbf{u}(x,t_{n+1})dx=\frac{1}{\triangle x}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}}\mathbf{u}(x,t_n)dx-\frac{\triangle t}{\triangle x}\left[\int_{t_n}^{t_{n+1}}\mathbf{f}(\mathbf{u}(x_{i-\frac{1}{2}},t))dt-\int_{t_n}^{t_{n+1}}\mathbf{f}(\mathbf{u}(x_{i+\frac{1}{2}},t))d\right.$$

According to the definition of mean values on cell $\mathbf{U}$ and flux funciton $\mathbf{F}$, we reach to

$$U_i^{n+1}=U_i^n-\frac{\triangle t}{\triangle x}\left(F_{i+\frac{1}{2}}^n-F_{i-\frac{1}{2}}^n\right), \tag{3.17}$$

where $\mathbf{F}_{i-\frac{1}{2}}\approx\frac{1}{\triangle t}\int_{t_n}^{t_{n+1}}\mathbf{f}(\mathbf{u}(x_{i-1/2},t))dt$.

For the purpose of evaluate (3.17) repeatedly, we need to do the integration in $\mathbf{F}_{i-\frac{1}{2}}$ (as well as $\mathbf{F}_{i+\frac{1}{2}}$) numerically. However, more grid point should be added if numerical quadrature is applied. To do it efficiently, only the value of U_{i-1} and U_i by

$$\mathbf{F}_{i-1/2}^n=\mathcal{F}(U_{i-1},U_i), \tag{3.18}$$

where $\mathcal{F}$ is referred as *numerical flux function*. Finally, the numerical methods reads

$$U_i^{n+1}=U_i^n-\frac{\triangle t}{\triangle x}\left(\mathcal{F}(U_{i-1}^n,U_i^n)-\mathcal{F}(U_i^n,U_{i+1}^n)\right). \tag{3.19}$$

Practically, there some different choice for the numerical flux, $\mathcal{F}(U_{i-1}^n,U_{i+1}^n)=\frac{1}{2}\big[f(u_{i-1}^n)+f(u_i^n)\big]$ is an unstable one, which simply take the mean flux function value at the boundary of the cell. A stable version looks into the direction from which the flow come from(upwind), for e.g. $u_t+\lambda u_x=0$ with $\lambda>0$. It yields the so-called Godunov Scheme

$$U_i^{n+1}=U_i^n-\lambda\frac{\triangle t}{\triangle x}(U_i^n-U_{i-1}^n). \tag{3.20}$$

例3.3.1 (此处应有一例). content

3.3.2 Riemann Solver

线性化 例: the advection equation

$$\begin{cases}\omega_t+\lambda\omega_x=0,\\ \omega(x,0)=\omega_0(x)\end{cases}$$

solved with the method of characteristics $\omega(x,t)=\omega_0(x-\lambda t)$.
Boundary condition for IBVP($a\le x\le b$)?
The hyperbolic equation with initial data

$$q_0(x)=\begin{cases}q_l & x<0\\ q_r & x>0\end{cases}$$

Consider the linear hyperbolic IVP

$$\begin{cases} q_t + Aq_x = 0, \\ q(x,0) = q_0(x) \end{cases}$$

Then we can write $A = R\Lambda R^{-1}$, where $R \in \mathbb{R}^{m\times m}$ is the matrix of eigenvectors and $\Lambda \in \mathbb{R}^{m\times m}$ is the matrix of eigenvalues. Making the substitution $q = Rw$, we get the decoupled system

$$w_t^p + \lambda^p w_x^p = 0, \quad p = 1 \dots m.$$

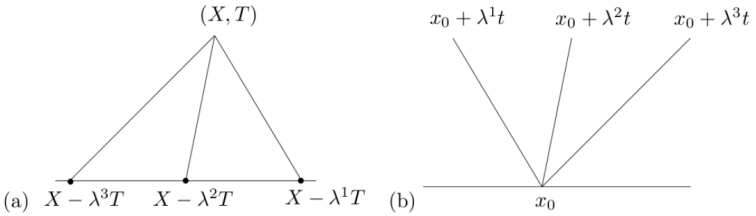


Fig. 3.2. For a typical hyperbolic system of three equations with $\lambda^1 < 0 < \lambda^2 < \lambda^3$, (a) shows the domain of dependence of the point (X, T), and (b) shows the range of influence of the point x_0.

(R. Leveque, 2002)

is known as the Riemann problem.

For the linear constant-coefficient system, the solution is

$$\begin{aligned} q(x,t) &= q_l + \sum_{p:\lambda^p<x/t} \big[l^p(q_r - q_l)\big] r^p \\ &= q_r - \sum_{p:\lambda^p\geq x/t} \big[l^p(q_r - q_l)\big] r^p \end{aligned}$$

Roe 的方案 Recall the numerical method for Conservation Law

$$Q_i^{n+1} = Q_i^n - \frac{\triangle t}{\triangle x}\big[\mathcal{F}(Q_i^n, Q_{i+1}^n) - \mathcal{F}(Q_i^n, Q_{i+1}^n)\big],$$

A linearized choice of the numerical flux based on the Godunov's method for the nonlinear problems. Define $|A| = R|\Sigma|R^{-1}$, where $|\Sigma| = diag(|\lambda^p|)$, then we can derive the Roe's flux as

$$F_{i-\frac{1}{2}}^n = \frac{1}{2}\big[f(Q_{i-1}) + f(Q_i)\big] - \frac{1}{2}|A|\big[Q_{i-1} + Q_i\big]$$

Remark:In this sense, R is properly chosen, such that A is a good enough approximation to nonlinear functional $\mathcal{F}$.

Godunov 的方案 **Remark**: Evolve step (2) requires solving the Riemann problem.

The following *REA algorithm* was proposed by Godunov (1959):

1. **Reconstruct** a piecewise polynomial function $\tilde{q}^n(x, t_n)$ from the cell averages Q_i^n. In the simplest case, $\tilde{q}^n(x, t_n)$ is piecewise constant on each grid cell:

$$\tilde{q}^n(x, t_n) = Q_i^n, \quad \text{for all } x \in C_i.$$

2. **Evolve** the hyperbolic equation with this initial data to obtain $\tilde{q}^n(x, t_{n+1})$.
3. **Average** this function over each grid cell to obtain new cell averages

$$Q_i^{n+1} = \frac{1}{\Delta x} \int_{C_i} \tilde{q}^n(x, t_{n+1})\, \mathrm{d}x.$$

Recall the solution to the Riemann problem form a linear system

$$Q_i - Q_{i-1} = \sum_{p=1}^{m} \left[l^p \big(Q_{i+1} - Q_i \big) \right] r^p = \sum_{p=1}^{m} \mathcal{W}^p_{i-\frac{1}{2}}$$

If $\triangle t$ is small enough, waves from adjacent cells do not interact!

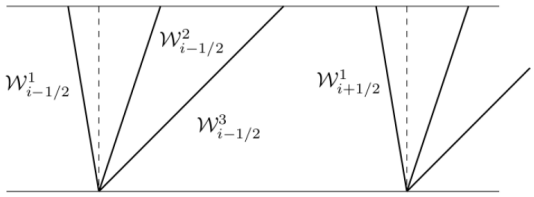


Fig. 4.7. An illustration of the process of Algorithm 4.1 for the case of a linear system of three equations. The Riemann problem is solved at each cell interface, and the wave structure is used to determine the exact solution time Δt later. The wave $\mathcal{W}^2_{i-1/2}$, for example, has moved a distance $\lambda^2 \Delta t$ into the cell.

Godunov's method for General Conservation Laws 最后通过如下"迎风"组合获得流通量表达式

$$F^n_{i-\frac{1}{2}} = f(Q_{i-1}) + \sum_{p=1}^{m} (\lambda^p)^- \mathcal{W}^p_{i-\frac{1}{2}},$$

or

$$F^n_{i-\frac{1}{2}} = f(Q_i) + \sum_{p=1}^{m} (\lambda^p)^+ \mathcal{W}^p_{i-\frac{1}{2}},$$

where $\lambda^+ = \max(\lambda, 0)$ and $\lambda^- = \min(\lambda, 0)$ is an upwind choice.

3.3.3 Total Variation Diminision(TVD)

Recall the numerical method for Conservation Law

$$Q_i^{n+1} = Q_i^n - \frac{\triangle t}{\triangle x}\left[\mathcal{F}(Q_i^n, Q_{i+1}^n) - \mathcal{F}(Q_i^n, Q_{i+1}^n)\right],$$

where $\mathcal{F}(Q_i^n, Q_{i+1}^n) \approx F_{i+\frac{1}{2}}^n = h(Q_{i+\frac{1}{2}}^-, Q_{i+\frac{1}{2}}^+)$.

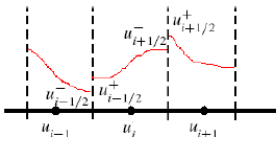
TVD: It is required that the numerical flux function $h(\cdot,\cdot)$ is monotone(Lipschitz continuous, monotone, $h(a,a) = a$)

Example

$$h(a,b) = 0.5(f(a) + f(b) - \alpha(b-a)),$$

where $\alpha = \max_u |f'(u)|$

(Weighted) Essentially Non-Oscillatory((W)ENO) 方案 The main concept of (W)ENO is where $\{u_i\}_{i=0}^n$ are the given **cell average** of a function $q(x)$.



Use ENO/WENO to compute $u^{\pm}_{i+1/2}$

$u^-_{i+1/2} = p_i(x_{i+1/2}) = v_i(u_{i-r}, ..., u_{i+s})$

$u^+_{i+1/2} = p_{i+1}(x_{i+1/2}) = v_{i+1}(u_{i-r}, ..., u_{i+s})$

technische universiteit eindhoven

Construct polynomials $p_i(x)$ of degree $k-1$, for each cell C_i, such that it is a k-th order accurate approximation to the function $q(x)$, which means

$$p_i(x) = q(x) + \mathcal{O}(\triangle^k) \qquad \forall x \in C_i, i = 0, 1, \cdots, N.$$

Finally, one can evaluate u at each cell interface($u^-_{i+1/2}$ and $u^+_{i+1/2}$).

例3.3.2 (此处应再有一例). content

3.4 Elliptic Equations for BVPs

3.4.1 Five-point scheme for elliptic equations

Let us introduce the basic two dimensional model problem of elliptic equations at first.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + f(x,y) = 0, (x,y) \in \Omega, \tag{3.21}$$

$$u = 0, (x,y) \in \partial\Omega, \tag{3.22}$$

Let us further compared FEM with FDM. Denote $u_{i,j}$ as the function values on grid points， then

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{-u_{i-1,j} + 2u_{i,j} - u_{i+1,j}}{h_x^2}, \qquad \frac{\partial^2 u}{\partial y^2} \approx \frac{-u_{i,j-1} + 2u_{i,j} - u_{i,j+1}}{h_y^2}.$$

Assuming that $h_x = h_y = h$, there yields

$$-\triangle u := -(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) \approx \frac{1}{h^2} \begin{pmatrix} & -u_{i,j+1} & \\ -u_{i-1,j} & 4u_{i,j} & -u_{i+1,j} \\ & -u_{i,j-1} & \end{pmatrix} \tag{3.23}$$

Sparse Laplace operator in Matlab:
There are two different ways computing Sparse Laplace operator in Matlab

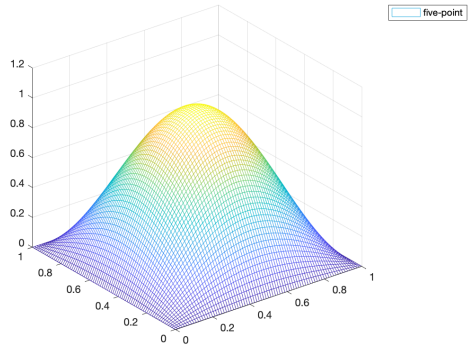
- kron — based on the inner unknown points(the number is $(n-1) \times (n-1)$);
- spLaplacian.m — based on all points(the number is $(n+1) \times (n+1)$).

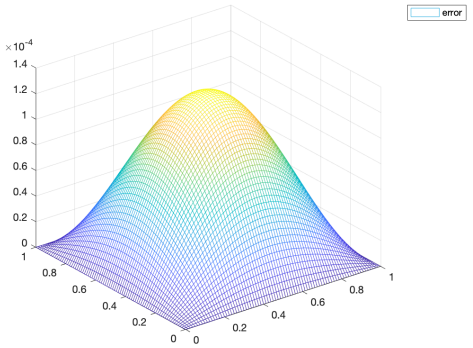
The latter scheme is recommended with the cost of taking the boundary condition as an equation. While the advantage is that the discretization of operator and boundary condition processing are relatively independent, so that it is more intuitive in mathematic form. we refer to testpoisson.m and related m files for details.

例3.4.1 (Five-point differential scheme)**.** Using the above five-point differential scheme to solve the following poisson equation

$$\begin{aligned} -\triangle u &= \pi^2 sin(\pi x) sin(\pi y), (x, y) \in [0, 1] \times [0, 1], \\ u &= 0, (x, y) \in \partial[0, 1] \times [0, 1] \end{aligned}$$

Using the above schemes, and m file of spLaplacian.m, the numerical result and error are shown in the following figure:





3.4.2 Error analysis

It is obvious that the truncation error is

$$T(x,y) = \frac{1}{12}h^2(u_{xxxx} + u_{yyyy}) + o(h^2).$$

Indeed this is bounded by T, where

$$|T(x,y) \leq T := \frac{1}{12}h^2(M_{xxxx} + M_{yyyy})$$

定理3.4.1 (Convergent)**.** Using fixed boundary values, and let $h \to 0$. If on any given position $(x^*, y^*) \in (0,1) \times (0,1)$,

$$u_{ij} \to u(x_i, y_j), \forall x_i \to x_i^*, y_j \to y_j^*.$$

proof Let us define an operator as:

$$L_h u_{ij} := \frac{1}{h^2}(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}),$$

in this sense, the exact and numerical solution satisfies the following two equations, respectively

$$L_h u_{ij}^* + f_{ij} = 0; \quad L_h u_{ij} + f_{ij} = T_{ij}.$$

Then the error in the usual way satisfies:

$$L_h e_{ij} = -T_{ij}. \tag{3.24}$$

To obtain a bound, let us first introduce a comparison function $\phi_{ij} := (x_i - \frac{1}{2})^2 + (y_j - \frac{1}{2})^2$, then there establishes $L_h\phi_{ij} = 4$. Write $\psi_{ij} := e_{ij} + \frac{1}{4}T\phi_{ij}$, there establishes

$$L_h\phi_{ij} = L_h e_{ij} + \frac{1}{4}TL_h\psi_{ij} = -T_{ij} + T \geq 0 \tag{3.25}$$

In another word, ϕ_{ij} can not be greater than all the neighbouring values. Hence there is a positive maximum value of ϕ. It is obvious that the maximum of ψ is $\frac{1}{2}$, which yields $\phi_{ij} \leq \frac{1}{8}$, then

$$e_{ij} \leq \phi_{ij} \leq \frac{1}{8}T = \frac{1}{9}6h^2(M_{xxxx} + M_{yyyy}). \tag{3.26}$$

□

3.4.3 The multigrid method

Multigrid method has been developed into a very efficient method for solving systems of algebraic equations which come from PDEs. Assuming that we have got the systems of equations from PDEs by some discrete method as

$$A\mathbf{x} = \mathbf{b}$$

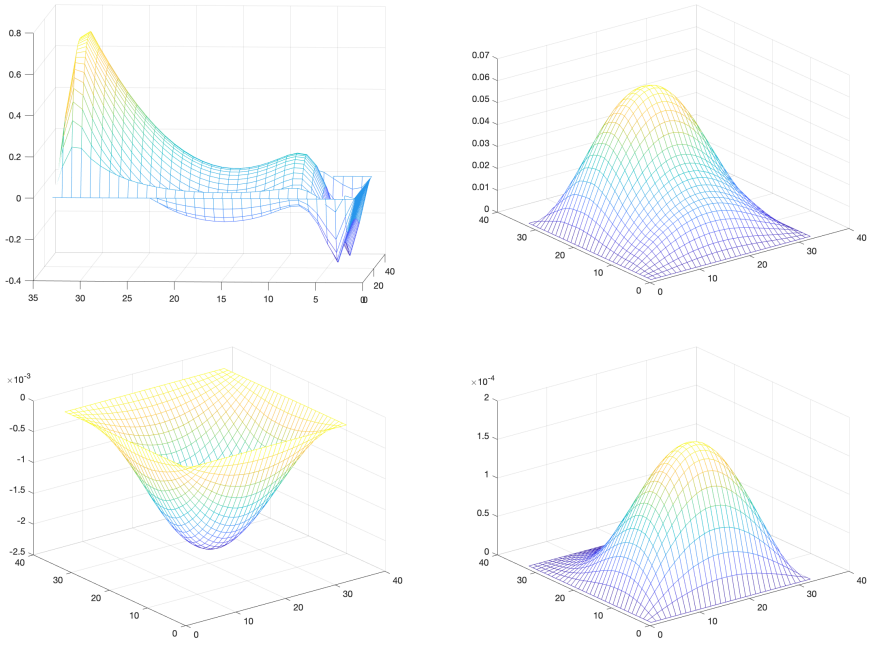
Considering a transfer matrix P and construct a new systems of equations on coarse mesh as

$$P^T AP\mathbf{x}_c = P^T\mathbf{b}_c$$

By solving this new problem, the corresponding solution $\mathbf{x}_c^k$ and error $\mathbf{e}^k$ are given as:

$$r^k = \mathbf{b} - AP\mathbf{x}_c^k; \quad Ae^k = r^k \tag{3.27}$$

例3.4.2 (Multigrid method)**.** Using multigrid method to solve the problem in example 3.4.1. The iteration is convergent after 14 steps, here we show four errors(step 1, 5, 10, 14) in those steps in the following figure.



3.5 Finite Element Methods

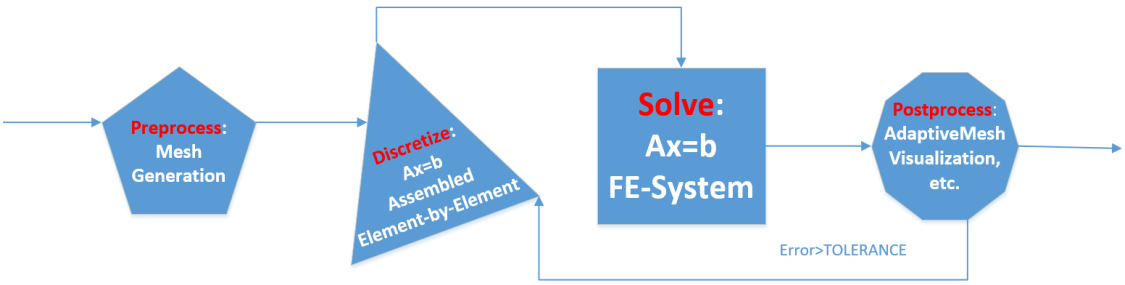
In the previous section, we present the Galerking method for two-point boundary value problems for ODEs. Considering PDE-based BVPs, the shape of the spatial domain

should be discretized by quadrilaterals or triangles in two dimensions and hexahedra or tetrahedra in three dimensions. Practical applications in scientific and engineering computing result this metholodgy into a general finite element framework for solving PDEs.

随着计算机及计算技术的飞速发展，出现了开发对象的自动离散及有限元分析结果的计算机可视化显示的热潮，使有限元分析的“瓶颈”现象得以逐步解决。在科学计算应用中，有限元分析的终极目标是能实现全自动求解。有限元方法de基本思路和解题步骤可归纳为(1)建立积分方程，根据变分原理或方程余量与权函数正交化原理，建立与微分方程初边值问题等价的积分表达式，这是有限元法的出发点。(2) 区域单元剖分，根据求解区域的形状及实际问题的物理特点，将区域剖分为若干相互连接、不重叠的单元。区域单元划分是采用有限元方法的前期准备工作，这部分工作量比较大，除了给计算单元和节点进行编号和确定相互之间的关系之外，还要表示节点的位置坐标，同时还需要列出自然边界和本质边界的节点序号和相应的边界值。(3)确定单元基函数，根据单元中节点数目及对近似解精度的要求，选择满足一定插值条件的插值函数作为单元基函数。有限元方法中的基函数是在单元中选取的，由于各单元具有规则的几何形状，在选取基函数时可遵循一定的法则。(4)单元分析：将各个单元中的求解函数用单元基函数的线性组合表达式进行逼近；再将近似函数代入积分方程，并对单元区域进行积分，可获得含有待定系数(即单元中各节点的参数值)的代数方程组，称为单元有限元方程。(5)总体合成：在得出单元有限元方程之后，将区域中所有单元有限元方程按一定法则进行累加，形成总体有限元方程。(6)边界条件的处理：一般边界条件有三种形式，分为本质边界条件(狄里克雷边界条件)、自然边界条件(黎曼边界条件)、混合边界条件(柯西边界条件)。对于自然边界条件，一般在积分表达式中可自动得到满足。对于本质边界条件和混合边界条件，需按一定法则对总体有限元方程进行修正满足。(7)解有限元方程：根据边界条件修正的总体有限元方程组，是含所有待定未知量的封闭方程组，采用适当的数值计算方法求解，可求得各节点的函数值。

In this chapter we introduce the Finite Element Methods(FEMs) based on elliptic Boundary Value Problems(BVPs). Basic numerical analysis of FEMs are briefly presented.

A general procedure for finite element calculation can be illustrated with the following figure:



例3.5.1 (Elasticity equation). d

例3.5.2. In this example, we consider a practical problem which describes the electrostatic potential u in a charged body $\Omega = \{(x, y)|x^2 + y^2 \leq 1\}$,

$$\begin{cases} -\Delta u + e^u &= 0, & x \in \Omega, \\ u &= 0, & x \in \partial\Omega. \end{cases}$$

The analytic solution is chosen to be

$$u = 2\ln\left(\frac{B+1}{B\left(x^2+y^2\right)+1}\right),$$

where $B = -5 + 2\sqrt{6}$ is a properly choosed constant with physical meaning.

For the purpose of avoiding the pollution from the errors of the discrete boundaries, the mesh refine near the domain boundary is the most practical way. As an alternative choice, the using of curved boundary triangle element can also meet the requirements. We would like to leave this as another topic for practical using. Since the analytic form of u is known in the current example, we simply evaluate u with its analytic expression instead of imposing $u = 0$ everywhere on the boundary edges. Different order finite element spaces are used for calculation and the errors are tabulated in the following table.

Degree	2	3	4	5	6	7	8	9
$\|u_h - u\|_1$	1.36e-03	6.19e-05	1.41e-06	5.27e-08	1.51e-09	5.34e-11	1.71e-12	1.43e-13

例3.5.3 (Driven Cavity)**.** The first test problem is the driven cavity problem on the unit square $[-1, 1] \times [-1, 1]$. The three sides $x = -1, x = 1$, and $y = -1$ are the fixed walls and the side $y = 1$ is moving from left to right with velocity 1. It is clear that the boundary condition $\mathbf{u} = (u, v) = (0, 0)$ for fixed boundary and $\mathbf{u} = (u, v) = (1, 0), \quad \forall \quad 0 \leq x \leq 1$ for moving boundary is suitable. Both meshes are generated by hand, where the boundary layer is resolved with more dense elements. With such settings of the meshes, any choice of above two meshes yields almost the same result in current example. The purpose of listing two meshes here is to emphasis the necessary of resolve the boundary layer in this example. We simulate the case of the Reynolds numbers equal to $100, 1000, 5000$ and 10000 respectively. The contours of the stream function is displayed.

It is remarkable that the investigations on the two dimensional driven cavity problem is fruitful in the literatures of Computational Fluid Dynamics, where various numerical schemes are utilized to get higher pricision. At the same time, it is widely accepted that very high value of Reynolds number has no physical meaning. However, it is a big challenge for numerical method to solve high Reynolds number case, and the purpose of such case is only to test the robustness of the numerical algorithms. We found that the simulation results in this paper are very similar with those in the literature.

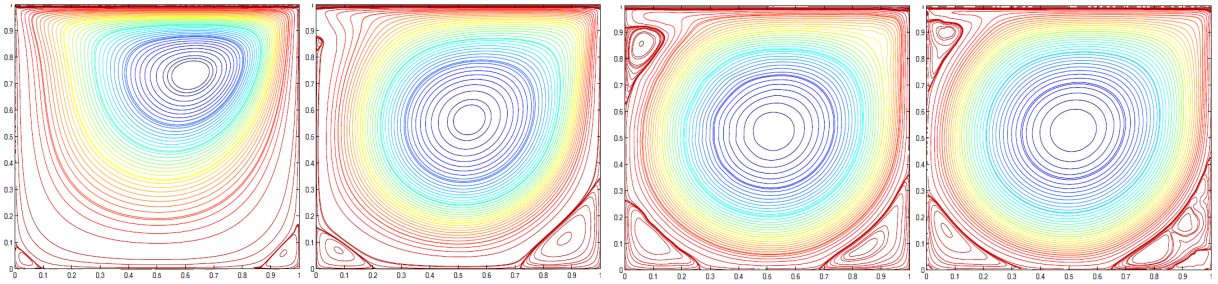


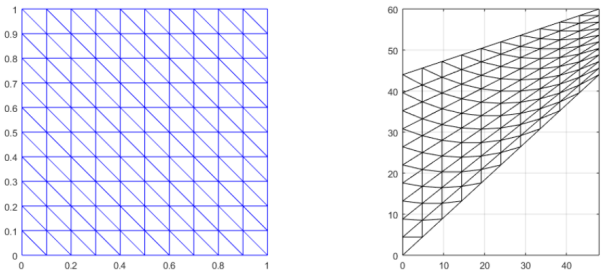
Figure 3.3: The contours of stream function when Reynolds number equals to 100, 1000, 5000 and 10000.

3.5.1 Finite element mesh - generation and adaptivity

In finite element applications, algorithm for mesh generation is essential for practical using. Finite element meshes with good quality will not only imrpove the accuracy of the final approximation, but also be a good preconditioning technique for the discretized finite element system. A fine-tune mesh for domains with specific geometry could be archived by experienced users, on the other hand, automatic mesh adaptive algorithms are more chanlledge for various application. For the purpose of generality, only the triangular mesh generation and adaptation are concerned.

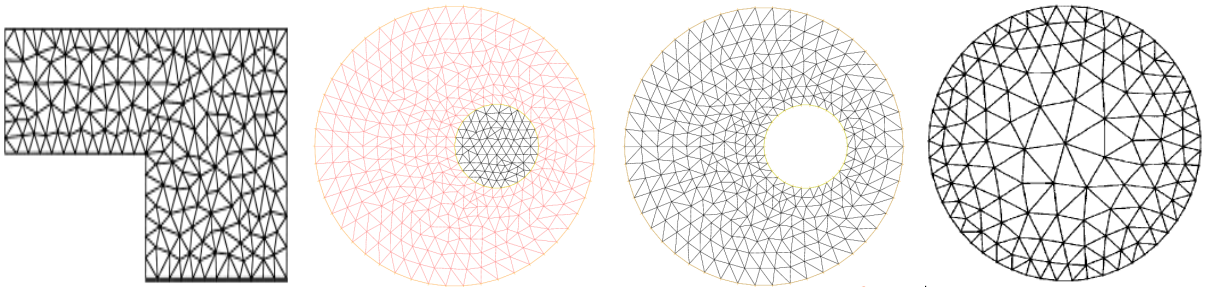
The conformity of the finite element mesh is essential for a legal approximation. It requires that 合法性:一个单元的结点不能落入其他单元内部，在单元边界上的结点均应作为单元的结点，不可丢弃。相容性:单元必须落在待分区域内部，不能落入外部，且单元并集等于待分区域。逼近精确性。待分区域的顶点(包括特殊点)必须是单元的结点，待分区域的边界(包括特殊边及面)被单元边界所逼近。

Mesh quality is evaluated based on the quality of each element shape. The best element shape is the equi-laterral triangle. secondly, a smooth transition between the neighboured element is important for continous approximation. Beside that, the density of nodes and the size of the elements vary automatically at the shape corner or regions with large gradients corresponding to concerned unknown solution. There are many practical techniques for generating triangular mesh. The interested readers are referred to the monograph written by K. Ho-Le.



映射法是一种半自动网格生成方法，根据映射函数的不同，主要可分为超限映射和

等参映射。因前一种映射在几何逼近精度上比后一种高，故被广泛采用。映射法的基本思想是：在简单区域内采用某种映射函数构造简单区域的边界点和内点，并按某种规则连接结点构成网格单元。这种方法可以很方便地生成四边形和六面体单元，若需要，也很容易转换成三角形和四面体单元。该法的主要缺点：首先必须将待分区域子划分为所要求的简单区域，这是一个十分复杂且很难实现自动化的过程。对复杂域采用手工方法划分甚至不可能。通常各简单区域边界采用等份划分。另外，该法在控制单元形状及网格密度方面是困难的。鉴于简单区域自动划分的困难性，Blacker试图采用知识系统和联合体素方法解决，但在复杂多孔域上仍难以处理，主要是体素数量和形状有限，将待分区域全自动划分为有限的预定体素并集是很难完全实现的。



Delaunay法，常用的代码实现二维EasyMesh, 三维的TetGen. 以二维为例，算法的基本原理：任意给定N个平面点Pi(i=1,2,...,N)构成的点集为S，称满足下列条件的点集Vi为Voronoi多边形。其中，Vi满足下列条件：Vi = X:—X- Pi—(—X- Pj—,X(R2,i(j,j=1,2,...,
Vi为凸多边形，称 Vimi=1为Dirichlet Tesselation图或对偶的Voronoi图。连接相邻Voronoi多边形的内核点可构成三角形Tk，称集合 Tk 为Delaunay三角剖分。DT法的最大优点是遵循“最小角最大”和“空球”准则。因此，在各种二维三角剖分中，只有Delaunay三角剖分才同时满足全局和局部最优。

定理3.5.1 (最大最小角定理). 用Delaunay算法生成的平面三角网格剖分具有最大的最小内角。

实现Delaunay三角剖分有多钟方法。Lee和Schachter操作很有效，但很难实现。而Watson、Cline和Renka、Sloan因操作容易、时间效率较好等优点而被广泛采用。为了进一步提高效率，Sloan研究其算法操作，提出了时间复杂性为O(N)(N为结点总数)的操作方法，从而为快速Delaunay三角剖分提供了有效途径。虽然DT法既适用于二维域也适用于三维域，但直接的Delaunay三角剖分只适用于凸域，不适用于非凸域，因此发展了多种非凸域的Delaunay剖分。

另一种非结构网格生成方法是AFM法，它的常用实现程序Netgen。基本原理：设区域的有向离散外边界集和边界前沿点集已经确定，按某种条件沿区域边界向区域内部扣除三角形(四面体)直到区域为空集。AFM法的关键技术有两个：一是区域的边界离散和和内点的合理生成。二是扣除三角形条件。目前，扣除三角形的条件有多种：最短距离条件，最大角条件，最大形状质量条件，以及最小外接圆(球)条件，即空球条件。AFM法最大优点是不仅在区域内部而且在区域边界所生成的网格单元形状均优良，网格生成全自动，可剖分任意实体。如果将板/壳、实体和梁采用统一的数据结

构，则可采用该原理实现不同维数和多种材料等混合工况结构件的网格自动剖分。若配合误差估计，则这种方法在自适应网格再生技术中使用效果甚佳。

Non-uniform meshes can save considerable computational efforts,especially for problems with two or three spatial dimension. The nonuniform mesh generation is a important topic in practical finite element applications 有限元的自适应性就是在现有网格基础上，根据有限元计算结果估计计算误差、重新剖分网格和再计算的一个闭路循环过程。当误差达到预规定值时，自适应过程结束。因此，有效的误差估计和良好的自适应网格生成是自适应有限元分析两大关键技术。就目前国外研究来看，自动自适应网格生成从大的方面可分为两类：网格重生成、网格细分以及网格移动

根据当前获得的计算结果，有限元后处理技术可以获得关于当前解函数的后验误差估计，以度量数值解在不同单元上对于真实解的毕竟逼近好坏。配合这个后验误差来确定新的结点密度分布，然后重新划分网格，再计算并重复上述过程直到求解精度达到预定目标为止。目前，网格再生技术在平面区域已得到了较好地实现，从理论上讲，该原理可扩充到三维实体域，但由于三维实体域难以完全自动用等结点密度曲面来分割任意实体。在实践中，网格重生成技术的优势主要表现在收敛速度快、网格单元形状稳定。FreeFem++是一个基于有限元方法，数值求解偏微分方程的免费软件，它是一个拥有自己高级编程语言的集成化产品。值得一提的是，它针对二维问题实现了网格重分功能，且算法的效率经过优化，在应用数学问题研究中具有一定的实用价值。它的主要机制是：。。。。。从FreeFEM++的手册中参考. h-型采用有选择地进一步子划分网格单元来细化网格以提高自由度,该法使用特别广泛.

Meshes in the finite element methods are always regard as mappings between two different coordinates, in two dimensional case,

$$x = x(\xi, \eta), \quad y = y(\xi, \eta),$$

where (ξ, η) and (x, y) are always referred as the computational mesh and the physical mesh respectively. It is remarkable that the two meshes have the same number of nodes and the same topological mesh structure. The computational mesh (ξ, η) is fixed and uniform in all time steps, while the physical mesh or the map (x, y) is nonuniform and exactly what we want to find out during the calculation.

The concept of mesh mapping make it possible to modify the position of the nodes in the physical mesh (x, y) at each time step for different requirements. In this sense, the nodes of the physical mesh change constantly, so that people refer the strategy as the moving mesh method. As have been stated in the introduction, the fundamental idea of the moving mesh methods is to find an optimal mesh to minimize the mesh-energy functional. The Winslow's equation is the exact Euler-Lagrange equation for the functional. Then the minimization problem is equal to solve

$$\nabla \cdot (\frac{1}{\omega} \nabla \xi(x, y)) = 0, \quad \nabla \cdot (\frac{1}{\omega} \nabla \eta(x, y)) = 0,$$

where $\xi(x, y), \eta(x, y)$ is actually the inverse mesh mapping, and $\nabla := (\nabla_x, \nabla_y)^T$ means taking derivatives with respective to coordinate system (x, y). In this model, ω is referred

as the monitor function, which is responsible for adjusting the density of the mesh nodes locally. However, additional cost should be paid to find the optimal mesh mapping since the calculation of the inverse map is required in this model, which seems to be too complicated when a problem defined on simple domains, as well as unit square, are considered. A new but similar model is considered to adaptively moving mesh

$$\nabla \cdot (\omega \nabla \mathbf{x}) = 0, \quad \nabla \cdot (\omega \nabla \mathbf{y}) = 0, \tag{3.28}$$

where $\nabla := (\nabla_\xi, \nabla_\eta)$ means taking derivatives with respective to (ξ, η). Such elliptic system are actually the formulation of equi-distribution theory based on the computational meshed.

Boundary mesh movement is a interesting research topic, and several standard strategies can be followed from the moving mesh literature, one can either solve a independent 1D moving mesh equation only for the boundary nodes or minimize a functional with boundary restrictions for harmonic maps. For a concise presentation, homogeneous Neumann boundary condition for the both coordinates is illustrated here. If further assume that the corner nodes of the domain keep fixed, such boundary settings keep the domain size fixed although the mesh is changed constantly. Standard finite difference scheme is efficient for computation, which yields

$$X_{i,j} = \frac{(\omega_{i-\frac{1}{2},j} X_{i-1,j} + \omega_{i+\frac{1}{2},j} X_{i+1,j} + \omega_{i,j-\frac{1}{2}} X_{i,j-1} + \omega_{i,j+\frac{1}{2}} X_{i,j+1})}{(\omega_{i,j-\frac{1}{2}} + \omega_{i,j+\frac{1}{2}} + \omega_{i,j-\frac{1}{2}} + \omega_{i,j+\frac{1}{2}})}, \tag{3.29}$$

where the $\frac{1}{2}$ in the subscription means to evaluate the monitor function at the half-point of the corresponding mesh edge. Practically it is evaluated by taking the average between two neighboring nodes, such as $\omega_{i,j-\frac{1}{2}} = \frac{1}{2}(\omega_{i,j-1} + \omega_{i,j})$. The extension for the iteration scheme (3.29) to the case of unstructured triangle mesh by updating position of the center nodes in a similar manner as well as those on Cartesian meshes, roughly,

$$X_i = \sum_j (\omega_i^j X_i^j) / \sum_j \omega_i^j,$$

where X_i^j is the j'th neighbor of node X_i and ω_i^j is the value of the monitor function defined on edge $X_i X_j$. Considerable time saving could be archived since only 3-5 iterations of (3.29) is enough to obtain satisfied adaptive meshes in practical computations, while the whole linear system should be solved in any other moving mesh methods.

The monitor function ω is responsible for adjusting the densities of the mesh locally, in a way that the mesh nodes are dense where the value of ω are large. In the framework of the moving mesh method, the choice of the monitor function is the key issue for a robust moving mesh algorithm. The most common choice of ω is related with the gradient of the solution

$$\omega(u) = \sqrt{\alpha + \beta |\nabla u|^2 + \gamma u^2}, \tag{3.30}$$

where α, β and γ are all parameters for different configurations. Practically, α is used to prevent the monitor function from too small value, which may destroy the smoothness of the generated mesh, and a larger value of β yields more mesh nodes near the region where the numerical solution varying rapidly, while γ is useful when you are intend to capture the position of singularity accurately for a potentially singular problems

Finally, the numerical solutions on the current time step can be fed to the next time step as the initial values after executing the interpolation. A convenience and constantly used version is actually the first order approximation based on the grid system of (x, y):

$$u^* = u + \nabla_x u(x^{n+1} - x^n) + \nabla_y u(y^{n+1} - y^n), \tag{3.31}$$

where u,$\nabla_x u$ and $\nabla_y u$ denote the value and two partial derivatives of u with respect to coordinate x and y of the old mesh (x^n, y^n), and u^* is the updated value onto the new mesh (x^{n+1}, y^{n+1}). We remark here that when the time step is small enough, the meshes change continuously so that the update scheme (3.31) is accurate enough.

3.5.2 Mixed FEM

参考NotesZou.pdf的8-9章，写3-5页

3.6 Iterative Methods for Large Sparse Linear Systems

在科学计算中人们往往需要解决具有大量自由度的计算问题，如一个在三维立方体上采用100x100x100的网格的有限差分方法离椭圆方程，将得到一个大约1,000,000阶的线性方程组问题。 因此出现了“高性能科学计算”这一概念。高性能计算涉及到硬件架构、编程框架和编程语言、软件设计以及算法的并行化设计等诸多方面，在这里我们关注的是如何尽可能地减少算法在存储空间和运算时间等方面的开销。我们将针对典型的稀疏线性方程组求解问题

$$Ax = b \tag{3.32}$$

以及稀疏矩阵特征值问题

$$Bx = \lambda x \tag{3.33}$$

where A and B are sparse matrix, which is defined as the following

定义3.6.1 (Sparse Matrix)**.** It is the definition of Sparse matrix!

直接法在矩阵规模比较小(在偏微分方程数值解中小于 1 百万)的情形是高效的,但是实际问题往往要求解很大的n的矩阵,而且往往含有大量的0元素,在用直接法时就会耗费大量的时间和存储单元。稀疏矩阵的直接求逆是不可取的，因为在消元过程中会将大量原本为0的元素被消元运算转换为非0。尽管有一些自由度重排序算法减少稀疏矩阵的”带宽“以尽可能减少这种非零元填充事件的发生，随着矩阵规模增长(如至千万维)，直接求逆算法将很快耗尽内存和计算力。

3.6.1 稀疏矩阵

另一方面注意到稀疏矩阵的乘法具有较高的效率，迭代法

基 记秩为N的矩阵,可以由一组线性无关的向量$\{\psi_k\}_{k=1}^N$。

大规模线性方程组的求解是有困难的. 带来困难的原因不仅仅在于其规模之大, 另一个重要的原因是线性方程组的不适定性(即矩阵奇异). 利用线性空间的理论, 基于变分(即投影)理论的迭代解法是计算近似解的有效方法.

Let A be an $n \times n$ real matrix and $\mathcal{K}$ and $\mathcal{L}$ be two m -dimensional subspaces of R_n. A projection technique onto the subspace $\mathcal{K}$ and orthogonal to $\mathcal{L}$ is a process described as

$$\text{Find } \tilde{x} \in x_0 + \mathcal{K}, \text{ such that } b - A\tilde{x} \perp \mathcal{L}$$

等价于

$$\text{Find } \tilde{x} = x_0 + \delta, \delta \in \mathcal{K}, \text{ such that } (r_0 - A\delta, \omega) = 0, \forall \omega \in \mathcal{L}$$

最速下降法和极小残量法分别是由两类不同的投影方法导致的:

最速下降(Steep Descent)法

```
for j = 0, 1, ..., until convergence do
    r_j = b − A x_j;
    α_j = (r_j, r_j)/(A r_j, r_j);
    x_{j+1} = x_j + α_j r_j;
end
```

极小残量(Minimal Residual)法

```
for j = 0, 1, ..., until convergence do
    r_j = b − A x_j;
    α_j = (A r_j, r_j)/(A r_j, A r_j);
    x_{j+1} = x_j + α_j r_j;
end
```

1950年,美国国家标准局数值分析研究所 Hestenes，Stiefel 和 Lanczos，发明了Krylov子空间迭代法求解$A\mathbf{x} = \mathbf{b}$。构造迭代

$$K\mathbf{x}_{i+1} = K\mathbf{x}_i + (\mathbf{b} - A\mathbf{x}_i)$$

其中，K(来源于作者俄国人Nikolai Krylov姓氏的首字母)是一个用投影法构造得到的接近于A的矩阵，根据K所属空间$\mathcal{K}$的不同取法得到求解不同类型的迭代格式。

该迭代形式的算法的妙处在于，它将复杂问题化简为阶段性的易于计算的子步骤。特别地在第m步，构造子空间

$$\mathcal{K}_m(A, \mathbf{r}_0) = span\{\mathbf{r}_0, A\mathbf{r}_0, \cdots, A^{m-1}\mathbf{r}_0\}.$$

只要分别采用$\mathbf{L}_m = \mathbf{K}_m$ 或$\mathbf{L}_m = A\mathbf{K}_m$ 即可得到不同类型的变分迭代法. 让我们来了解一下如何构造子空间的数值方法.

首先,可以利用Arnoldi算法构造正交子空间$\mathbf{K}_m$,如下述算法:

```
Choose a unit vector v_1;
for j = 1, 2, ..., m do
    Compute h_ij = (Av_j, v_i),   ∀i = 1, 2, ··· , j;
    Compute u_j = Av_j − Σ_{i=1}^{j} h_ij v_i;
    h_{j+1,j} = ‖u_j‖_2;
    if h_{j+1,j} == 0 then
        Stop
    else
        v_{j+1} = u_j / h_{j+1,j}
    end
end
```

接着实施Arnoldi-Modified Gram-Schmidt正交化过程:

```
Choose a unit vector v_1;
for j = 1, 2, ..., m do
    Compute u_j = Av_j;
    for i = 1, ..., j do
        h_{i,j} = (u_j, v_i);
        u_j = u_j − h_{i,j} v_i;
    end
    h_{j+1,j} = ‖u_j‖_2;
    if h_{j+1,j} == 0 then
        Stop
    else
        v_{j+1} = u_j / h_{j+1,j}
    end
end
```

Full Orthogonalization Method(FOM)结合了上述两个过程:

1 Compute $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\beta = \|\mathbf{r}_0\|_2$, and $\mathbf{v}_1 = \mathbf{r}_0/\beta$;
2 Define the $m \times m$ matrix $H_m = \{h_{i,j}\}_{i,j=1,2,\ldots,m}$, set $H_m = 0$;
3 **for** $j = 1, 2, \ldots, m$ **do**
4 Compute $\mathbf{u}_j = A\mathbf{v}_j$;
5 **for** $i = 1, \ldots, j$ **do**
6 $h_{i,j} = (\mathbf{u}_j, \mathbf{v}_i)$;
7 $\mathbf{u}_j = \mathbf{u}_j - h_{i,j}\mathbf{v}_i$;
8 **end**
9 $h_{j+1,j} = \|\mathbf{u}_j\|_2$;
10 **if** $h_{j+1,j} == 0$ **then**
11 set $m = j$, and goto 16
12 **else**
13 Comput $\mathbf{v}_{j+1} = \mathbf{u}_j/h_{j+1,j}$
14 **end**
15 **end**
16 Compute $\mathbf{y}_m = H_m^{-1}(\beta\mathbf{e}_1)$ and $\mathbf{x}_m = \mathbf{x}_0 + V_m\mathbf{y}_m$;

极小残差(Generalized Minimized Residual)法

简称GMRes. 通过上述算法的铺垫,可以归结为如下算法

1 Compute $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\beta = \|\mathbf{r}_0\|_2$, and $\mathbf{v}_1 = \mathbf{r}_0/\beta$;
2 Define the $(m+1) \times m$ matrix $H_m = \{h_{i,j}\}_{1\le i\le(m+1)}^{1\le j\le m}$, set $H_m = 0$;
3 **for** $j = 1, 2, \ldots, m$ **do**
4 Compute $\mathbf{u}_j = A\mathbf{v}_j$;
5 **for** $i = 1, \ldots, j$ **do**
6 $h_{i,j} = (\mathbf{u}_j, \mathbf{v}_i)$;
7 $\mathbf{u}_j = \mathbf{u}_j - h_{i,j}\mathbf{v}_i$;
8 **end**
9 $h_{j+1,j} = \|\mathbf{u}_j\|_2$;
10 **if** $h_{j+1,j} == 0$ **then**
11 set $m = j$, and goto 16
12 **else**
13 Comput $\mathbf{v}_{j+1} = \mathbf{u}_j/h_{j+1,j}$
14 **end**
15 **end**
16 Compute $\mathbf{y}_m$ to **minimize** $\|\beta\mathbf{e}_1 - H_m\mathbf{y}\|_2$ and $\mathbf{x}_m = \mathbf{x}_0 + V_m\mathbf{y}_m$;

共轭梯度法(Conjugate Gradient)及其变形

GMRes迭代算法对于非对称矩阵A尤其适用. 在A对称时的特殊情形,利用Lanczos三项递推关系可将FOM简化成如下简洁的算法:

```
Compute r_0 = b − Ax_0, p_0 = r_0;
for j = 0, 1, ..., until convergence do
    α_j = (r_j, r_j)/(Ap_j, p_j);
    x_{j+1} = x_j + α_j p_j;
    r_{j+1} = r_j − α_j A p_j;
    β_j = (r_{j+1}, r_{j+1})/(r_j, r_j);
    p_{j+1} = r_{j+1} + β_j p_j;
end
```

同理得到的迭代方法即为共轭梯度法(CG), 该迭代法对于对称矩阵A具有收敛性. 令$\mathbf{x}_m$ 是执行第$m-$步Conjugate Gradient algorithm得到的近似解, 并且$\mathbf{x}^*$是精确解，那么

$$||\mathbf{x}^* - \mathbf{x}_m||_A \leq \left[\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right]^m ||\mathbf{x}^* - \mathbf{x}_0||_A$$

其中κ是矩阵A最大与最小特征值的比值，即所谓条件数。可见κ越接近1，则共轭梯度法收敛越快!

CG算法的其他主要常见变形有

- ICCG : Incomplete Cholesky预处理的CG迭代
- BiCG : 双正交共轭梯度法
- BiCGstab : 稳定化的BiCG

3.6.2 预条件处理技术

预处理技术可以减少迭代法运行时的迭代次数(注意不一定能减少计算量). 预处理的优点在于能结合不同的迭代法优势,在很多情形可以减少总体的计算量. 以CG迭代法为例，该方法的主要思想是: 设M是non-singular矩阵，并且$M^{-1}A$的条件数相对教小，则求解

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

相对容易，或者

$$(AM^{-1})\mathbf{y} = \mathbf{b},$$

再求$M\mathbf{x} = \mathbf{y}$得到原方程的解。

显然地,$M\mathbf{x} = \mathbf{y}$相比于原问题更容易求解. 另一方面,CG算法采用的M也需要是对称正定的, 这样不会破坏CG算法的收敛性. 举例来说, 假设$A = L + D + L^T$,那么可以构造Jacobi预处理子

$$M := M_{Jacobi} = D^{-1}$$

或SOR预处理子

$$M := M_{SOR} = \frac{1}{\omega(2-\omega)}(D+\omega L)D^{-1}(D+\omega L^T), 0 < \omega < 2.$$

例3.6.1.

Bibliography

[1] Toader A Allaire G, Jouve F. Structural optimization using sensitivity analysis and a level-set method. *Journal of Computational Physics*, 2004.

[2] Francois Murat and Jacques Simon. *Etude de problemes d'optimal design*. Springer Berlin Heidelberg, 1976.

[3] J. Simon. Differentiation with respect to the domain in boundary value problems. *Numerical Functional Analysis Optimization*, 2(7):649–687, 2010.

[4] O .C. Zienkiewicz and K. Morgan: Finite Elements and Approximation, 1983; The Finite Element Method, 6th ed.

[5] S. C. Brenner and L. R. Scott: The mathematical theory of finite element methods, 3rd ed.

[6] ark. S. Gockenbach: Understanding And Implementing the Finite Element Method

[7] H. Elman, etc.: Finite Elements and Fast Iteraive Solvers - with applications in incompressible fluid dynamics

[8] Jian-Ming Jin: The Finite Element Method in Electromagnetics, 2nd ed.