

Contents

1 Matrix Computation	3
1.1 Linear Equation Solver	3
1.1.1 Linear equations	3
1.1.2 Solve with direct method - Gauss ellimination	4
1.1.3 稀疏矩阵	6
1.1.4 Classical iterative methods	7
1.2 超定线性方程组的最小二乘解	10
1.2.1 正规方程解法	10
1.2.2 其他常用的解法	12
1.2.3 矩阵的正交变换	13
1.2.4 矩阵分解	14
1.3 Eigenvalue Problem	16
1.3.1 敏感性分析	17
1.3.2 blas	19
1.3.3 lapack	19
1.3.4 spooles	19
1.3.5 mumpus	20
2 Numerical Approximations	22
2.1 Roots	22
2.1.1 The mathematical model	22
2.1.2 Numerical methods for $f(x) = 0$	23
2.1.3 Numerical methods for equation system	26
2.2 插值与拟合问题	29
2.2.1 Interpolation with polynomials	30
2.2.2 Interpolation with piecewise polynomials	32
2.2.3 Best approximation	34
2.2.4 Least square approximation	34
2.2.5 Neural network approximation	37
2.3 Quradrature Rule	38
2.3.1 Numerical Integration	38

2.3.2	Formula with interpolation case(Newton-Cotes)	40
2.3.3	Methods for improving the accuracy of integration	43
2.3.4	Best quadrature rule	43
2.4	Numerical Methods for Initial Value Problems(IVPs)	44
2.4.1	Initial Value Problem	44
2.4.2	Numerical Methods for ODEs	46
2.4.3	Stablity Improvements	49
2.5	Numerical Methods for Two-point Boundary Value Problems(BVPs)	52
2.5.1	Finite Difference Approximation	52
2.5.2	Galerkin method	53
2.5.3	Collocation Method	55
2.5.4	Shooting Method	56
2.5.5	Newton's method for nonlinear problems	57
4	Numerical PDE	59
4.1	Parabolic Equations for Initial Value Problems(IVPs)	60
4.1.1	Explicit(iterative) numerical scheme	60
4.1.2	Error estimation for explicit scheme	61
4.1.3	Implicit schemes	63
4.1.4	Mixed method: θ -scheme	64
4.2	Hyperbolic Equation-based Initial Value Problems(IVPs)	66
4.2.1	Upwind scheme	66
4.2.2	Lax-Wendroff Scheme	67
4.2.3	Leap-Frog scheme	67
4.2.4	Numerical analysis for the methods	68
4.3	FVMs for One Spatial Dimension	69
4.3.1	Finite Volume Formulas	70
4.3.2	Riemann Solver	71
4.3.3	Total Variation Diminision(TVD)	73
4.4	Elliptic Equations for BVPs	74
4.4.1	Five-point scheme for elliptic equations	74
4.4.2	Error analysis	75
4.5	The multigrid method	76
4.6	Mathematical Framework for FEM	77
4.7	Iterative Methods for Large Sparse Linear Systems	79
4.7.1	稀疏矩阵	80
4.7.2	预条件处理技术	84
4.8	Finite element mesh - generation and adaptivity	84
4.9	Mixed FEM and Navier-Stokes equation solver	88

Chapter 1

Matrix Computation

1.1 Linear Equation Solver

1.1.1 Linear equations

科学计算应用经过建模和数值离散(如数据的插值与拟合,微分方程离散等)之后,都归结于求解一个或若干个线性方程组。用矩阵 - 向量的符号表示,即

$$A\mathbf{x} = \mathbf{b}. \quad (1.1)$$

其中, A 表示一个 $n \times n$ 的系数矩阵

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}.$$

右端项 $\mathbf{b} = [b_1, b_2, \dots, b_n]^T$ 是一个 n 维列向量。关于矩阵的对称, 正定, 相似, 逆, 秩等概念参考线性代数/高等代数。

定义1.1.1 (Singularity). 称 $n \times n$ 矩阵 A 是非奇异的, 如过它满足下列条件之一:

1. A 满秩, 即 $\text{rank}(A) = n$
2. 行列式不为零, 即 $|A| \neq 0$
3. 存在逆矩阵 A^{-1}

否则, 称矩阵 A 是奇异的。

线性方程组解的存在唯一性与其稀疏矩阵的奇异性有必然的联系。事实上，在 A^{-1} 存在的情形，可以用Crame法则获得该解，即

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

更一般地，对于给定系数矩阵 A 和右端项 \mathbf{b} 的线性方程组，其解的情况有如下三种：

1. 唯一解： A 非奇异， \mathbf{b} 任意；
2. 无数解： A 奇异， $\mathbf{b} \in \text{span}(A)$ ；
3. 无解： A 奇异， $\mathbf{b} \notin \text{span}(A)$.

1.1.2 Solve with direct method - Gauss ellimination

求解该问题最经典的方法是Gauss消元法，读者可能已经在线性代数或高等代数课程中学习过。消元法是线性方程组问题的直接求解方法，迭代法是另一类在实用计算中更常用的方法将在下一章中讲。此处仅举一例，目的是为了展示如何完成算法实现。本书中数值算法的实现采用了MATLAB，但是相关算法流程也被给出，因此读者可以根据自身情况采用其他任何一种计算机语言实现。

例1.1.1 (Gauss消元法). 记矩阵 A 和向量 b 为某个线性方程组的系数矩阵和右端向量，请写出求解线性方程组 $Ax = b$ 的Gauss消元法过程。

```

1 function x = gauss(A, b);
2 n = size(A,1);
3 for k = 1:n-1
4     A(k,:)=A(k,:)/A(k,k);
5     for j = k+1:n
6         factor = -A(j,k)/A(k,k);
7         A(j,k:end) = A(j,k:end) + factor*A(k,k:end);
8         b(j) = b(j) + factor*b(k);
9     end
10 end
11 b(n) = b(n)/A(n,n);
12 for k = n:-1:2
13     b(1:k-1) = b(1:k-1) - A(1:k-1,k)*b(k);
14 end
15 x = b;

```

Conditioning and Stability

时间复杂度：考虑算法执行过程中所做的乘除法次数。一步消元过程所做的乘除适定性

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

回代过程所做的乘法次数为

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

综上,高斯消去法所需总的乘除法次数为

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

空间复杂度: 不需额外存储空间、原址存储分解结果

稳定性: 当 $|a_{kk}| \approx 0$ 时可导致浮点数溢出, 用、条件数 $\kappa(A)$ 衡量

系数矩阵的度量是衡量线性方程组求解难易程度的一个重要标识。

常用的矩阵范数的定义是常见的, $\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^m |a_{ij}|$, $\|A\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^m |a_{ij}|$,

和谱范数 $\|A\|_2 = \sqrt{\lambda_{\max}(A^T A)}$ 。值得指出的是, 任何矩阵范数满足 $\|A\| \geq \rho(A)$ 。

谱半径: $\rho(A) = \max |\lambda(A)|$, 即绝对值最大的特征值。

矩阵的条件数: $\kappa(A) = \|A\| \cdot \|A^{-1}\|$ 。若取 L_2 范数 $\|\cdot\|_2$, 则 $\kappa(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}$, 其中, σ 表示最大和最小奇异值, 我们将在稍后的章节中介绍它们。有如下性质:

1、若 A 是正规矩阵 ($A^T A = AA^T$), 则 $\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}$.

2、若 A 是酉矩阵 ($A^T A = AA^T = I$), 则 $\kappa(A) = 1$.

定义1.1.2 (Condition of Matrix). 非奇异方阵 A 的条件数定义为

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

其中, $\|\cdot\|$ 是给定的矩阵范数。

可以从定义得到条件数的一些重要性质:

1. 对于任意非奇异方阵, $\|A\| \geq 1$
2. 对于单位矩阵, $\|A\| = 1$
3. 对于任意非奇异方阵 A 和 $\gamma \in \mathbb{R}$, $\text{cond}(\gamma A) = \text{cond}(A)$

4. 对于任何对角阵 $D = \text{diag}(d_i)$, $\text{cond}(D) = \frac{\max_{1 \leq i \leq n} |d_i|}{\min_{1 \leq i \leq n} |d_i|}$.

如果有一系列具有相同系数矩阵(不同右端项)的线性方程组组要求解, 如固定网格下的时间相关偏微分方程的隐格式离散, 显然消元的过程都是重复的。利用系数矩阵的L(下三角)U(上三角)矩阵分解

$$A = LU$$

可以消除这种重复计算。具体地，让我们以一个实例来说明：若对矩阵 A 的高斯消去过程时稳定的，则存在

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

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

由最后一步的结果可以分离出 L 和 U 分别为：

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

一般地，对于 n 阶非奇异矩阵 A ，可以用如下脚本实现 LU 分解：

```

1 function [L, U] = lu_primer(A);
2 n = size(A,1);
3 for k = 1:n-1
4     A(k+1:n,k) = A(k+1:n,k)/A(k,k);
5     for i = k+1:n
6         A(i,k+1:end) = A(i,k+1:end) + A(i,k)*A(k,k+1:end);
7     end
8 end
9 L = tril(A); U = triu(A) + diag(A);

```

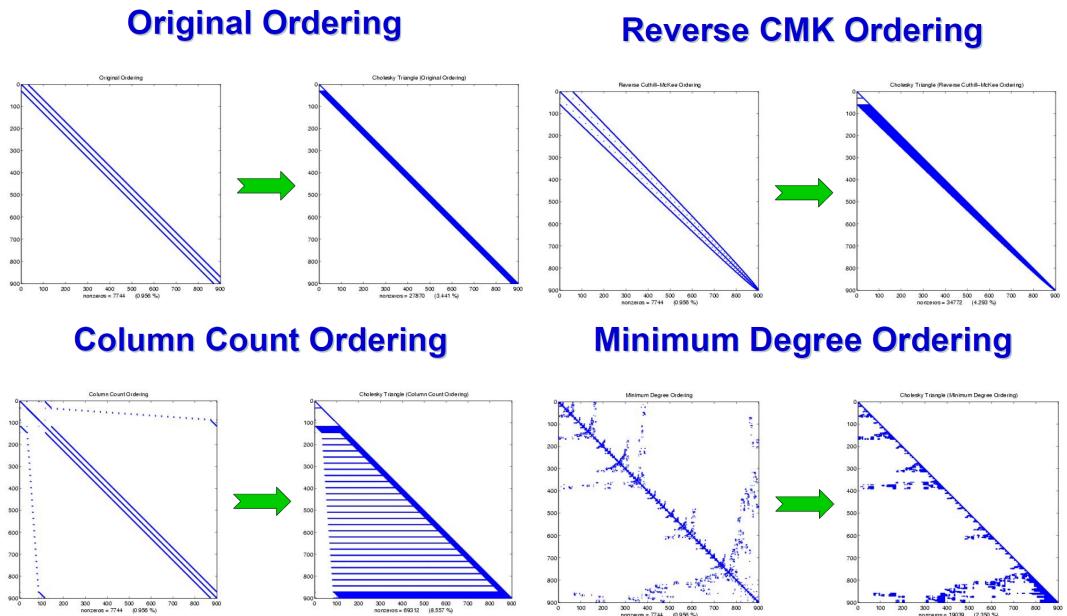
当矩阵 A 是对称情形，显然分解所得的两个三角矩阵也是对称的，即 $U = L^T$ 。这也被称为对称矩阵的Dolittle分解。

1.1.3 稀疏矩阵

稀疏矩阵排序技术主要是通过减少“填零”来实现的。如下所示的是一种减少“带宽”的技术：

三对角矩阵追赶法

矩阵中非零元素只占极少部分，带状矩阵与一般稀疏矩阵。可以设计特殊的存储结构以加快运算的速度。两点边值问题有限差分或有限元离散将会得到带三对角系数矩阵的线性方程组问题。所谓“三对角”，意味着只有矩阵的主对角线及其与其平行的上下两条对角线上的元素非零的情形。



1.1.4 Classical iterative methods

这就需要将线性方程组进行改写。可以将系数矩阵进行

$$\mathbf{x} = G\mathbf{x} + \mathbf{b}$$

那么,当 $\mathbf{x}^{(0)}$ 给定后,可以利用迭代格式

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

得到序列 $\{\mathbf{x}^{(k)}\}_{k=0}^n$.

- 序列的收敛只与算子 G 有关,而与初值 $\mathbf{x}^{(0)}$ 的选取无关!
- 收敛性: 谱半径 $\rho(G) < 1$

定理1.1.1 (一半迭代法收敛性). 迭代矩阵普半径 ≤ 1

Jacobi迭代法

为纪念普鲁士著名数学家雅可比

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

$$\begin{array}{c} \text{对 } Ax = b \\ A = \begin{bmatrix} 0 & & & & \\ a_{21} & 0 & & & \\ a_{31} & a_{32} & 0 & & \\ \vdots & \vdots & \vdots & \ddots & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & 0 \end{bmatrix} + \begin{bmatrix} a_{11} & & & & \\ & a_{22} & & & \\ & & \ddots & & \\ & & & a_{nn} & \\ & & & & \end{bmatrix} + \begin{bmatrix} 0 & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{23} & \cdots & a_{2n} \\ 0 & \cdots & a_{3n} \\ \vdots & & \ddots & \vdots \\ 0 & & & 0 \end{bmatrix} \\ \text{下三角阵} \quad \text{对角阵} \quad \text{上三角阵} \\ = L + D + U \end{array}$$

$\because a_{ii} \neq 0$ (Jacob假设) $\therefore D$ 可逆

进一步, 有: $(L + D + U)x = b$

$$Dx = -(L + U)x + b$$

$$x = -D^{-1}(L + U)x + D^{-1}b \quad \rightarrow \text{理论分析}$$

Gauss-Seidel 迭代法

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

矩阵 G 对角占优时两类迭代均收敛。但可以构造不同的例子显示: Gauss-Seidel 法收敛时, Jacobi 法可能不收敛; 另一方面, Jacobi 法收敛时, Gauss-Seidel 法也可能不收敛!

定理 1.1.2 (对称正定矩阵收敛). content

超松弛(Relaxation)迭代法

在 Gauss-Seidel 基础上的一种加权修正, 分两步

1.

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

2.

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

A 对称正定时, 松弛迭代收敛 $0 < \omega < 2$. 收敛的快慢与松弛因子 ω 的选择有密切关系。 $0 < \omega < 1$ under-relaxation methods, $\omega > 1$ over-relaxation methods. 如何选取最佳松弛因子, 使矩阵谱半径达到最小, 是一个尚未很好解决的问题. 经验上可取 $1.4 < \omega < 1.6$.

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 $k = 1, 2, \dots$, 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	...	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	...	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

在上图所列的算例中, 计算的精度要求误差达到小数点后第 7 位. 从计算的结果可以知道: Gauss-Siedel 迭代法计算了 34 步, 而 SOR 迭代法只用了 14 步. 请使用相同的参数设定, 找出下述例题的 SOR 迭代解法中最优的 ω 的数值:

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

定理 1.1.3 (SOR 法收敛性定理). content

1.2 超定线性方程组的最小二乘解

当矩阵 A 的行数多于列数, 即 $m > n$ 的情形, 我们称问题

$$Ax = b$$

是一个超定线性方程组。若 $span(A)$ 是图凸集、 $\mathbf{r} = \mathbf{b} - Ax$ 是凸映射, 则存在唯一; A 列满秩 \leftrightarrow 存在唯一; A 秩亏损, 解仍存在, 但不唯一!

1.2.1 正规方程解法

为了求解 $Ax = b$, 寻找使 $\|Ax - b\|_2$ 的梯度为零的 x , 故有正规方程 (相对易求解)

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

不管 m 与 n 的大小关系, 上述方程组总是具有 n 个未知 n 个方程。Hessian 矩阵 $A^T A$ 是对称的、正定的。

以上述例子为例, 容易列出正规方程组可求得正规方程的解为 $\mathbf{x}^T = [1236, 1943, 2416]^T$,

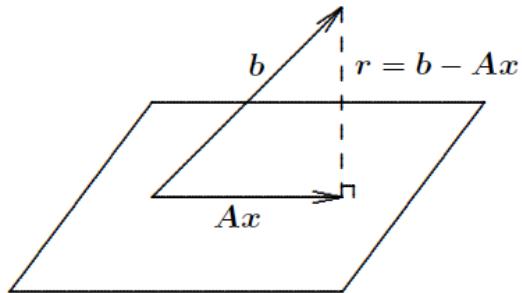
$$\mathbf{A}^T \mathbf{A} \mathbf{x} = \begin{bmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -651 \\ 2177 \\ 4069 \end{bmatrix} = \mathbf{A}^T \mathbf{b}.$$

将该数值解代入原问题, 可得到原方程残量的范数为 $\|\mathbf{r}\|_2^2 = 35$ 。

几何的观点：正交投影法

也可以直接考虑求解 $Ax = \mathbf{b}$)。对于 $m > n$ 的问题, 一般 $\mathbf{b} \notin span(A)$, 则由投影关系可知 \mathbf{r} 垂直于 $Ax - \mathbf{b}$, 即 (等同于正规方程组)

$$A^T(Ax - \mathbf{b}) = 0$$



上述正规方程组方法即求 A 的广义逆

$$A^{-*} = (A^T A)^{-1} A^T$$

其中， $P = A^T A$ 是在 $\text{span}(A)$ 上的正交投影阵， $\text{cond}_2(A) = \|A\|_2 \|A^{-*}\|_2$ 这个量体现了与秩亏损矩阵的接近程度，也容易因此知道正规方程组的条件数与原问题条件数几乎成平方关系。

扰动分析

求解 $A^T A \mathbf{x} = A^T \mathbf{b}$ ，此时考虑右端项引起的扰动

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

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

对于例1的分析：

$$\text{cond}(A) = \|A\|_2 \cdot \|A^{-*}\|_2 = 2 \cdot 1 = 2$$

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

此例的条件数和角度 θ 都足够小，问题良态！忽略高阶项并简化之后，有 因此，

对于矩阵 A 的扰动 $A+E$ ，解的扰动由正规方程

$$(A+E)^T (A+E)(x + \Delta x) = (A+E)^T b$$

$$\Delta x \approx (A^T A)^{-1} E^T r - (A^T A)^{-1} A^T E x = (A^T A)^{-1} E^T r - A^+ E x,$$

一个反例

考虑到数值稳定性，也可以将最小二乘问题转换为其对应的增广(Augment)方程组或者三角最小二乘问题求解。

$$\begin{aligned}
\frac{\|\Delta \mathbf{x}\|_2}{\|\mathbf{x}\|_2} &\lesssim \|(\mathbf{A}^\top \mathbf{A})^{-1}\|_2 \cdot \|\mathbf{E}\|_2 \cdot \frac{\|\mathbf{r}\|_2}{\|\mathbf{x}\|_2} + \|\mathbf{A}^\top\|_2 \cdot \|\mathbf{E}\|_2 \\
&= [\text{cond}(\mathbf{A})]^2 \cdot \frac{\|\mathbf{E}\|_2}{\|\mathbf{A}\|_2} \cdot \frac{\|\mathbf{r}\|_2}{\|\mathbf{A}\|_2 \cdot \|\mathbf{x}\|_2} + \text{cond}(\mathbf{A}) \frac{\|\mathbf{E}\|_2}{\|\mathbf{A}\|_2} \\
&\leq ([\text{cond}(\mathbf{A})]^2 \cdot \frac{\|\mathbf{r}\|_2}{\|\mathbf{A}\|_2} + \text{cond}(\mathbf{A})) \frac{\|\mathbf{E}\|_2}{\|\mathbf{A}\|_2} \\
&= ([\text{cond}(\mathbf{A})]^2 \tan\theta + \text{cond}(\mathbf{A})) \frac{\|\mathbf{E}\|_2}{\|\mathbf{A}\|_2}.
\end{aligned}$$

例 3.6 条件数平方效应 考虑如下矩阵及扰动

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

其中 $\epsilon \ll 1$, 接近 $\sqrt{\epsilon_{\text{mach}}}$, 计算

$$\text{cond}(\mathbf{A}) = 1/\epsilon, \quad \|\mathbf{E}\|_2 / \|\mathbf{A}\|_2 = \epsilon.$$

对右端向量 $\mathbf{b} = [1, 0, \epsilon]^\top$, 有 $\|\Delta \mathbf{x}\|_2 / \|\mathbf{x}\|_2 = 0.5$, 因此解的相对扰动大约为 $\text{cond}(\mathbf{A})$ 乘以 \mathbf{A} 的相对扰动. 对这个右端向量不存在条件数平方效应, 这是因为此时残差较小, 且 $\tan\theta \approx \epsilon$, 在条件数估计中扰动平方项被忽略了.

另一方面, 对于右端向量 $\mathbf{b} = [1, 0, 1]^\top$, 有 $\|\Delta \mathbf{x}\|_2 / \|\mathbf{x}\|_2 = 0.5/\epsilon$, 因此解的相对扰动等于 $\text{cond}(\mathbf{A})$ 的平方乘以 \mathbf{A} 的相对扰动. 对这样的右端向量, 残差的范数大约为 1, 且 $\tan\theta \approx 1$, 此时在扰动估计式中条件数平方项不能忽略, 这时解的敏感性非常高. ■

1.2.2 其他常用的解法

Augment方法

特点是化矩为方, 设解向量 \mathbf{x} 和残差向量 \mathbf{r} , 让它们满足

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

写成矩阵形式, 有

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

为了调节相对误差(rescale技巧), 有

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

一般地, 取参数 $\alpha = \max_{i,j} |a_{i,j}|/1000$. 会导致计算量增加, 但是对于大型问题可利用其结构的特殊性获得简化. 在MATLAB中实际计算很有效!

三角最小二乘问题

- 正交变换保欧氏距离

- 等价的三角最小二乘问题
- QR分解实现矩阵的正交分解;

1.2.3 矩阵的正交变换

Gram-Schmidt正交化是一般技巧，上述两种实现方式是等价的，只有在形式上的一些

算法 3.2 古典格拉姆-施密特正交化

```

for k=1 to n           {列循环}
    q_k = a_k
    for j=1 to k-1      {从当前列
        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 stop {如果线性}
    q_k = q_k / r_kk
    end
end

```

算法 3.3 改进的格拉姆-施密特正交化

```

for k=1 to n           {列循环}
    r_kk = || a_k ||_2
    if r_kk = 0 then stop {如果线性}
    q_k = a_k / r_kk
    for j=k+1 to n       {将当前列
        r_jk = q_k^T a_j
        a_j = a_j - r_jk q_k
    end
end

```

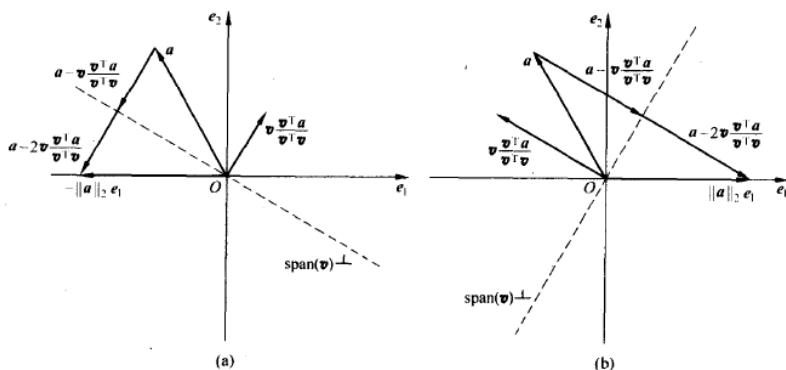
区别：被动减与主动消。另外，改进版本可以选主元，以增加稳定性！此外，常见的正交化方法还有Householder变换和Givens旋转

HouseHolder 变换

$$H\mathbf{a} = \alpha \mathbf{e}_1$$

其中，

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



根据几何意义，可取

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

Householder矩阵性质

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

- symmetric
- orthogonal

例：将向量 $\mathbf{a} = [2, 1, 2]^T$ 实施household变换。第一步，取

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

第二步，实施变换，即矩阵向量乘法 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变换

记旋转矩阵

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

如：可把向量 $[a_1, a_2]^T$ 旋转成 $[\sqrt{a_1^2 + a_2^2}, 0]^T$ ，此时，

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

需要指出的是：变换矩阵G是正交矩阵，也不改变范数 $\|\cdot\|_2$ ；G变换一次得一个0元，H变换一次只剩一个非0元；也可用于实现QR分解；G变换工作量、存储量均比H变换多。

1.2.4 矩阵分解

在求解线性方程组的直接法中，我们已经注意到Gauss消元法可以用矩阵的LU分解法实现。事实上，对于更一般的矩阵还有其他的分解方式，它们适用于不同

QR分解

定理1.2.1. 设 A 是 $m \times n$ 阶矩阵， $m \geq n$. 假如 A 是列满秩的（非奇异），则存在一个唯一的正交阵 Q ($Q^T Q = I$) 和唯一的具有正对角元 $r_{ii} > 0$ 的 $n \times n$ 阶上三角阵 R ，使得 $A = QR$

证明 记 $A = [\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n]$ 的列向量形式, 将这些列向量做正交化即得到正交矩阵 $Q[\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_n]$, 记 $r_{ji} = \mathbf{q}_j^T \mathbf{a}_i$, 即有

$$\mathbf{a}_i = \sum_{j=1}^i r_{ji} \mathbf{q}_j,$$

当然, QR分解是可以通过Gram-Schmidt正交化来实现的:

```

1 for i = 1, 2, ..., n do
2   q_i = a_i;
3   for j = 1, 2, ..., i do
4     r_{ji} = q_j^T q_i 或 q_j^T a_i;
5     q_i = q_i - r_{ji} q_j;
6   end
7   r_{ii} = ||q_i||_2;
8   if r_{ii} == 0 then
9     quit
10  else
11    q_i = q_i / r_{ii}
12  end
13 end

```

当然在实际应用中, Givens 变换、Householder 变换以及 Gram-Schmidt 正交化等都能实现QR分解; $>> [Q, R] = qr(A);$

例1.2.1 (QR分解实例).

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

奇异值分解

定理1.2.2. 对于任意给定的 $m \times n$ 矩阵 A 具有如下形式所谓SVD分解

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

其中, U 是 $m \times m$ 正交矩阵 ($U^H U = I$), V 是 $n \times n$ 正交矩阵 ($V^H V = I$), $\Sigma = diag(\sigma_1, \sigma_2, \dots, \sigma_n)$ 是一个 $m \times n$ 对角矩阵。

其中， Σ 对角上的数值称为矩阵 A 的奇异值：

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

而 U 和 V 的列向量分别称为对应的左、右奇异向量。

例1.2.2 (SVD分解的一个例子).

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

当极小化 $\|Ax - b\|_2$ 时,若 A 秩亏或“接近”于秩亏,常称为秩亏的最小二乘问题。SVD可计算其低维近似解。

SVD在现代科学应用中具有广泛的应用,在求解诸如数字图像恢复、某些积分方程的解或从噪声数据中提取信号等病态问题中,常用的正则化(regularization)来解决秩亏问题。下面是几个参考链接,在图像分析中的应用,感兴趣的读者可以自行参考:

- <http://zh.wikipedia.org/wiki/奇异值分解>
- <http://www.cnblogs.com/LeftNotEasy/archive/2011/01/19/580337.html>
- SVD的图像压缩

1.3 Eigenvalue Problem

计算矩阵一个或多个特征值的数值方法是科学计算中关心的重要问题。

定义1.3.1 (矩阵特征值). 给定一个代表 n 维向量空间上的线性变化的 $n \times n$ 矩阵 A ,希望找某个非零向量 x 和标量 λ ,满足

$$Ax = \lambda x$$

则称 λ 为 A 的一个特征值, x 为相应的(右)特征向量。

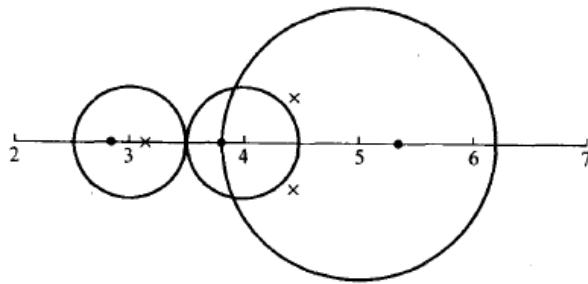
直接估算特征值

谱半径: 特征值集合 $\lambda(A)$ 中模最大,记为 $\rho(A)$

定理1.3.1 (Gershgorin圆盘定理). 设 A 是一个任意的矩阵,则 A 的特征值 λ 位于由 $|\lambda - a_{ii}| \leq \sum_{j \neq i} |a_{ij}|$, ($i = 1, \dots, n$) 所定义的 n 个如下图所示的圆盘的并中:

例1.3.1. 请估算下列矩阵特征值所在的区间范围。

$$A_1 = \begin{bmatrix} 4.0 & -0.5 & 0.0 \\ 0.6 & 5.0 & -0.6 \\ 0.0 & 0.5 & 3.0 \end{bmatrix} \quad A_2 = \begin{bmatrix} 4.0 & 0.5 & 0.0 \\ 0.6 & 5.0 & 0.6 \\ 0.0 & 0.5 & 3.0 \end{bmatrix}$$



1.3.1 敏感性分析

考虑特征值的扰动问题

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

忽略二阶项，再化简、左乘左特征向量 \mathbf{y}^H ，可得：

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

那么，运用范数不等式可得估计(其中 θ 是左、右特征向量夹角)

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

实对称矩阵或复共轭矩阵的左右特征向量相同，良态！

求解特征值可粗略地分为：直接法和迭代法。

计算特征值的直接法可以从其定义出发，通过其与特征多项式零点的关系，将问题转换为方程求根问题：特征值问题 $A\mathbf{x} = \lambda\mathbf{x}$ 等价于求解方程

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

\mathbf{x} 有非零解的充要条件是系数矩阵奇异，即特征多项式的零点

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

就是 A 的特征值。 特征值问题 \leftrightarrow 方程求根问题

例1.3.2. 算例：特征多项式零点

迭代法是求解矩阵特征值的重要方法，迭代法通常用于稀疏矩阵，或能方便地执行矩阵向量乘法的隐式算子的情形。注意直接法中也需要用到迭代；如果其中用一个固定的迭代次数收敛几乎不失败，也称为直接法。幂法和反幂法是最简单的求解矩阵特征值的迭代法。

例 4.3 特征多项式 例 4.2 中的第 3 个矩阵的特征多项式为

$$\det\left(\begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) = \det\left(\begin{bmatrix} 3-\lambda & 1 \\ 1 & 3-\lambda \end{bmatrix}\right)$$

$$= (3-\lambda)(3-\lambda) - 1 \times 1 = \lambda^2 - 6\lambda + 8 = 0,$$

利用求根公式,这个多项式的根为

$$\lambda = \frac{6 \pm \sqrt{36 - 32}}{2} = \frac{6 \pm 2}{2},$$

因此,特征值为 $\lambda_1 = 4, \lambda_2 = 2$.

幂法

幂法只能求矩阵 A 绝对值最大的特征值

```

1 给定  $\mathbf{x}_0$ ;
2 for  $i = 0, 1, \dots$  until convergence do
3   计算  $\mathbf{y}_{i+1} = A\mathbf{x}_i$ ;
4   计算近似特征向量  $\mathbf{x}_{i+1} = \mathbf{y}_{i+1}/\|\mathbf{y}_{i+1}\|_2$ ;
5   计算近似特征值  $\lambda_{i+1} = \mathbf{x}_{i+1}^T A \mathbf{x}_{i+1}$ ;
6 end

```

反幂法

相反, 反幂法将找到最小特征值, 但可以通过“位移”的技巧计算任意数 σ 附近的那个特征值, 其算法如下:

```

1 给定  $\mathbf{x}_0$ ;
2 for  $i = 0, 1, \dots$  until convergence do
3   计算  $\mathbf{y}_{i+1} = (A - \sigma I)^{-1} \mathbf{x}_i$ ;
4   计算近似特征向量:  $\mathbf{x}_{i+1} = \mathbf{y}_{i+1}/\|\mathbf{y}_{i+1}\|_2$ ;
5   计算近似特征值:  $\lambda_{i+1} = \mathbf{x}_{i+1}^T A \mathbf{x}_{i+1}$ ;
6 end

```

正交化是一类十分高效的计算技巧, 将正交技巧用于矩阵特征值计算的迭代法中也有十分良好的效果。基于 QR 正交分解, 人们提出了如下的 QR 算法:

```

1 给定  $A_0$ ;
2 for  $i = 0, 1, \dots$  until convergence do
3   分解  $\mathbf{A}_i = Q_i R_i$  (QR 分解);
4   计算  $\mathbf{A}_{i+1} = R_i Q_i$ ;
5 end

```

利用该方法, 我们将能同时得到矩阵 A 的所有特征值和特征向量, 最后 $\{A_i\}_{i=1}^\infty$ 将收敛到一个上三角矩阵, 其对角元素即为所求所有特征值。这是因为:

$$A_{i+1} = R_i Q_i = (Q_i^T Q_i) R_i Q_i = Q_i^T (Q_i R_i) Q_i = Q_i^T A_i Q_i$$

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)

瑞利Releigh商迭代

```

1 给定  $\mathbf{x}_0$  满足  $\|\mathbf{x}_0\|_2 = 1$  以及精度  $TOL$ ;
2  $\rho_0 = \frac{\mathbf{x}_0^T A \mathbf{x}_0}{\mathbf{x}_0^T \mathbf{x}_0}$ ;
3 for  $i = 0, 1, \dots$  until  $\|A \mathbf{x}_{i+1} - \rho_{i+1} \mathbf{x}_{i+1}\|_2 < TOL$  do
4    $\mathbf{y}_{i+1} = (A - \rho_i I)^{-1} \mathbf{x}_i$ ;
5    $\mathbf{x}_{i+1} = \mathbf{y}_{i+1} / \|\mathbf{y}_{i+1}\|_2$ ;
6    $\rho_{i+1} = \frac{\mathbf{x}_{i+1}^T A \mathbf{x}_{i+1}}{\mathbf{x}_{i+1}^T \mathbf{x}_{i+1}}$ ;
7 end

```

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 $\mathbf{x}_0 = [0, \dots, 0, 1]^T$ 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。

软件包与参考教材

1.3.2 blas

提供矩阵-向量以及矩阵-矩阵运算得软件包, 支持不同精度(如float,double)等类型的运算, 是其他很多软件包的基础。

1.3.3 lapack

集成了大量基本数值线性代数运算, 由计算数学领域多位专家主持开发工作, 是数值代数研究人员的必备工具。

1.3.4 spooles

求解线性方程组的直接解法。开源库安装案例: Spooles 2.2

1. 下载最新版本，目前为2.2版本，更新慢；
2. 解压缩: tar -xvf spooles.2.2.tar.gz;
3. 修改目录下Make.inc中的编译命令，最简单是gcc；
4. 构建库: make lib.

使用案例- A 为14922阶的稀疏矩阵求解线性方程

$$AX = Y$$

where A is square, large and sparse, and X and Y are dense matrices with one or more columns. 在Spooles 2.2 中

1. 内存中建立线性方程组
 - (a) Constructing an InpMtx object that holds the entries of A ;
 - (b) Constructing a DenseMtx objct that holds the entries of Y ;
 - (c) Constructing a DenseMtx object to hold the entries of X .
2. 利用Spooles 中的Bridge类进行求解:
 - (a) Initialization and setup step;
 - (b) Factorization step;
 - (c) Solution step.

1.3.5 mumpus

求解线性方程组的直接解法的另一个常用软件包。

superLU

基于L U分解实现的线性方程求解器，有MPI/OpenMP并行版本以及串行版本

Intel MKL

非开源，但是Linux系统下可以免费使用。针对Intel的CPU优化了编译指令，具有比其他开源软件快得多得执行速度。

数值线性代数算法是高性能科学计算的重要基础，有不少经典教材提供了更为详细的算法细节和拓展介绍，感兴趣的读者可以通过这些教材获得更多有用的信息：

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

Exercises

1. write gs ellimination
2. perform dolittle decomposision
3. do the least square calculation
4. write gs ellimination
5. perform dolittle decomposision
6. do the least square calculation
7. write gs ellimination
8. perform dolittle decomposision
9. do the least square calculation
10. write gs ellimination
11. perform dolittle decomposision
12. do the least square calculation

Chapter 2

Numerical Approximations

2.1 Roots

2.1.1 The mathematical model

定义2.1.1 (求根问题). 已知 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, f 的取值恰好为 \mathbf{y} , 即使得

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

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):

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

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/ ∞ 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 $x = f^{-1}(0)$ 。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). $f: \mathbb{R}^n \rightarrow \mathbb{R}^n$ 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 $g(S) \subseteq S$ ，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 $f(x) = x^3 - 4\cos x = 0$.

```

1 function x = bisect(f,a,b,tol)
2     xlow = a; plow = f(xlow);
3     xhigh = b; phigh = f(xhigh);
4     while xhigh - xlow > 2*tol,
5         xmid = (xlow + xhigh)/2; pmid = f(
6             xmid);
7         if pmid*plow < 0,
8             xhigh = xmid; phigh = pmid;
9         elseif pmid*phigh < 0,
10            xlow = xmid; plow = pmid;
11        else
12            xlow = xmid; xhigh = xmid;
13        end
14    end
15    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

```

1 %f = @(x) x.^2 - 2;
2 f = @(x) sqrt(x+2); x0 = 0;
3 err = 1; temp = x0; i = 0;
4 while err>1e-5
5     i = i+ 1;
6     x(i) = f(temp);
7     err = abs(x(i)-temp);
8     temp = x(i);
9 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：

```

x0 = 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 $f(x) = x^2 - 4\cos x = 0$.

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

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

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.

```

1 f = @(x) (x.^2-4*cos(x))./(2*x + 4*sin(
2 x));
3 x0 = 3; temp = x0; err = 1; i = 1;
4 while err > 1e-5 && i <= 10
5 x(i) = temp - f(temp);
6 err = abs(f(temp));
7 temp = x(i); i = i+1;
8 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：

```

x0 = 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 $f(x) = x^2 - 4\cos x = 0$.

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 $x_0 = 3$, $x_1 = 2$.

```

1 f = @(x) x.^2-4*cos(x); x0 = 3; x1 = 2;
2 err = 1; temp1 = x0; temp2 = x1; i = 1;
3 x(i) = temp2;
4 while err > 1e-5
5 i = i+1;
6 x(i) = temp2 - f(temp2).*((temp2-
7 temp1)./(f(temp2)-f(temp1)));
8 err = abs(f(temp2).*((temp2-temp1)
9 ./(f(temp2)-f(temp1))));
10 temp1 = temp2;
11 temp2 = x(i);
12 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 $g: \mathbb{R}^n \rightarrow \mathbb{R}^n$ ，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 $\mathbf{f}: \mathbb{R}^n \rightarrow \mathbb{R}^n$, 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 $k=0,1,2,\dots$

解 $\mathbf{J}(\mathbf{x}_k)\mathbf{s}_k = -\mathbf{f}(\mathbf{x}_k)$ 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). \mathbf{F}'' exists at x^* .

定理2.1.3 (Global Convergence). \mathbf{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 $k=0,1,2,\dots$

解 $\mathbf{B}_k\mathbf{s}_k = -\mathbf{f}(\mathbf{x}_k)$ 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,

$$\mathbf{F}(\mathbf{x}) := \begin{bmatrix} 3x_1 - \cos x_1 - \sin x_2 \\ 4x_2 - \sin x_1 - \cos x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

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:

$$\mathbf{x} = G(\mathbf{x}) = \begin{bmatrix} (\cos x_1 + \sin x_2)/3 \\ (\sin x_1 + \cos x_2)/4 \end{bmatrix}$$

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

```

1 f = @(x1,x2) [cos(x1) +
    sin(x2))/3;sin(x1)
    + cos(x2))/4];
2 h = @(x1,x2)[3*x1-cos(x1)
    - sin(x2);4*x2-sin
    (x1) - cos(x2)];
3 x0 = [1.5;1]; err = 1; temp
    = x0; i = 1;
4 while err>1e-5
5     x(:, i) = f(temp(1),temp
        (2));
6     err = max(abs(h(x(1,i),
        x(2,i))));
7     temp = x(:,i);      i = i
        + 1;
8 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:

$$J = \begin{bmatrix} 3 + \sin x_1 & -\cos x_2 \\ -\cos x_1 & 4 + \sin x_2 \end{bmatrix}$$

. In this sense, the calculation is performed in matlab, in which the initial guess is $\mathbf{x}_0 =$

```

1 J = @(x1,x2) [3+sin(x1),-cos(x2);-cos(
    x1),4+sin(x2)];
2 h = @(x1,x2)[3*x1-cos(x1) - sin(x2);4*
    x2-sin(x1) - cos(x2)];
3 x0 = [1.5;1]; temp = x0; err = 1; i = 1;
4 while err > 1e-5 && i <= 10
5     s = J(temp(1),temp(2))\(-h(temp(1),
        ,temp(2)));
6     x(:, i) = temp + s;
7     err = max(abs(h(x(1,i),x(2,i))));
8     temp = x(:,i);      i = i+1;
9 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

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

```

1 J = @(x1,x2) [3+sin(x1),-cos(x2);-cos(
    x1),4+sin(x2)];
2 h = @(x1,x2)[3*x1-cos(x1) - sin(x2);4*
    x2-sin(x1) - cos(x2)];
3 x0 = [1.5;1]; J0 = J(x0(1),x0(2));
4 tempx1 = x0; tempJ = J0; err = 1; i = 1;
5 while err > 1e-5 && i <= 10
6     x(:, i) = tempx1;
7     s = tempJ\(-h(tempx1(1),tempx1(2)));
8     tempx2 = tempx1 + s;
9     yk = h(tempx2(1),tempx2(2)) - h(
    tempx1(1),tempx1(2));
10    tempJ = tempJ + ((yk - tempJ*s)*s
    ')/(s'*s);
11    err = max(abs(h(x(1,i),x(2,i))));
12    tempx1 = tempx2; i = i+1;
13 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

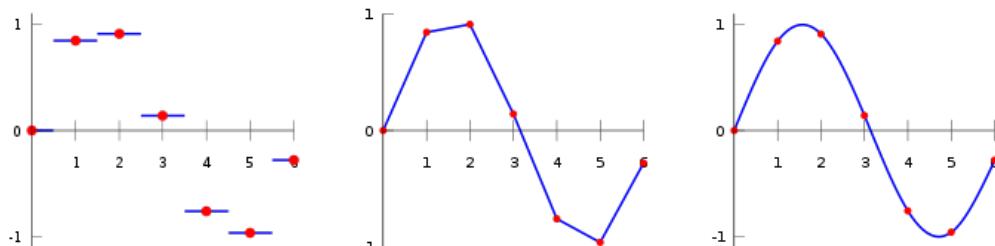
插值问题的来源于各类实际应用，如要求画出一条通过所有离散点的光滑曲线、利用列表函数求中间值、求列表函数的导数或积分、使用简单函数代替复杂函数，从而可以快速方便的求出需要求得的值等等。当然，这些问题又是来自于各类科学与工程计算问题的实际需求中抽象出来的。让用数学的语言进行定义插值问题：

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

$$f(t_i) = y_i, \quad \forall i = 0, 1, \dots, n$$

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 $(t_i, y_i), i = 1, 2, \dots, n$, 选择基函数 $\phi_1(t), \phi_2(t), \dots, \phi_n(t)$, 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：

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

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：

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

这里， \mathbf{A} 是一个范德蒙矩阵，当 $x_0 < x_1 < \dots < x_n$ 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 $\mathcal{O}(n^3)$ ，计算成本高！

拉格朗日插值公式

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

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

显然，我们有结论：

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

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, \dots, 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 $(t_i, y_i), i = 1, 2, \dots, n$, $n-1$ 次牛顿基函数为：

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

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, monomial and Newton interpolation and point out the difference between them. Try to compute the corresponding secondary Interpolation polynomials with datas $(-1, 1), (0, 0), (1, 1)$.

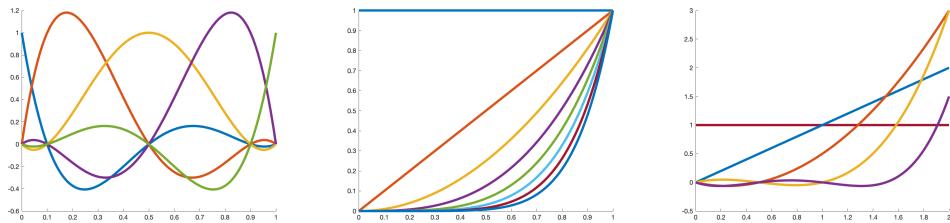


Figure 2.1: (Left) Lagrange base, (Middle) monomial base, (Right) Newton base

差商

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

差商的性质：

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

则可以得到插值多项式：

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

并且此时的误差为：

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

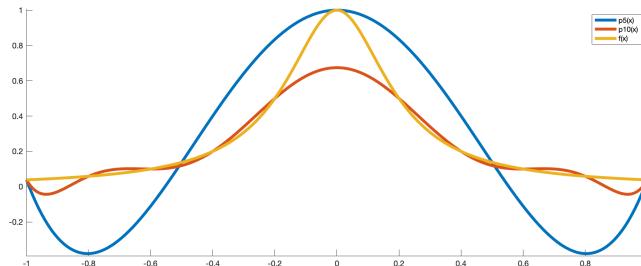
其中可以计算出二阶的误差为： $E_2 = -\frac{h^2}{8}f''(\xi)$

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 $(t_i, y_i), i = 1, \dots, n, t_1 < t_2 < \dots < t_n$, 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。



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

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

```

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

```

```

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

```

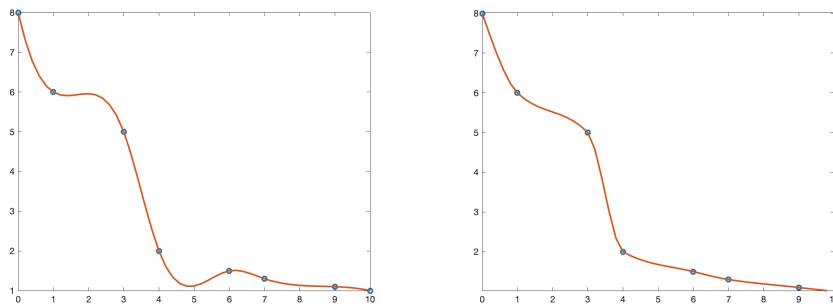


Figure 2.2: (Left)Interpolation with cubic splines;(Right)Interpolation with cubic Hermitt

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 $f : \mathbb{R} \rightarrow \mathbb{R}$, which satisfies:

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

Choosing some base functions $\phi_1(t), \phi_2(t), \dots, \phi_n(t)$, 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

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

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 $\{t_i, y_i\}, i = 1, 2, \dots, n$, 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

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

Non-polynomial base for data fitting

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

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

B-spline base for data fitting

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

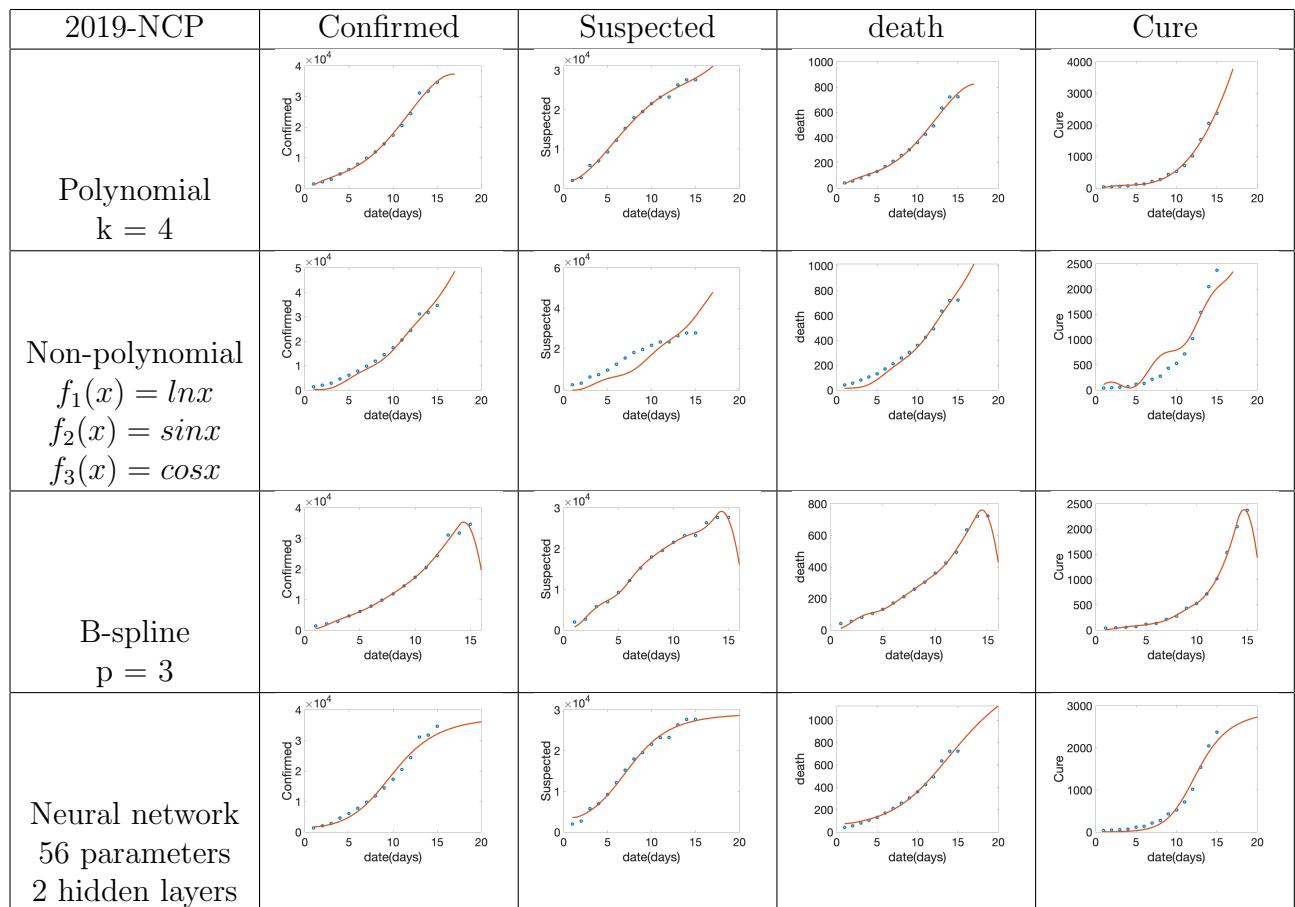
$$A^T Ax = A^T y, A = \begin{bmatrix} B_1^p(t_1) & B_2^p(t_1) & \dots & B_k^p(t_1) \\ B_1^p(t_2) & B_2^p(t_2) & \dots & B_k^p(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ B_1^p(t_n) & B_2^p(t_n) & \dots & B_k^p(t_n) \end{bmatrix}$$

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



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 , $l = 0, 1, \dots, L$, let $\mathbf{w}^l, \mathbf{b}^l$ denote

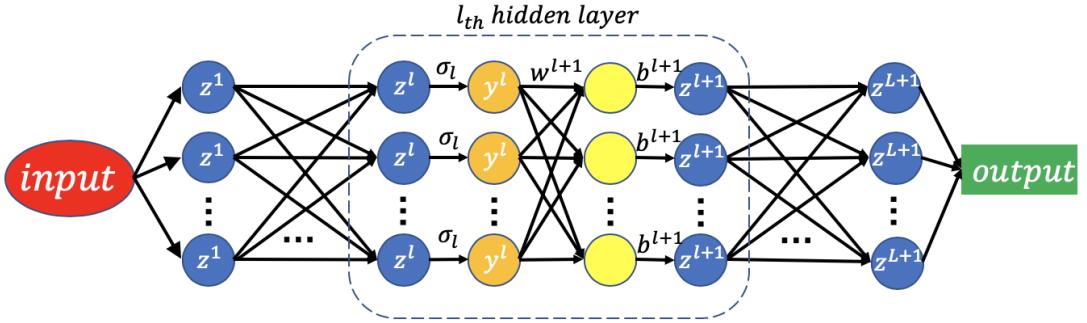


Figure 2.4: Structures of ANN with L hidden layers

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 $\mathbf{w} = \{\mathbf{w}^1, \mathbf{w}^2, \dots, \mathbf{w}^{L+1}\}$, $\mathbf{b} = \{\mathbf{b}^1, \mathbf{b}^2, \dots, \mathbf{b}^{L+1}\}$, 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 $\delta^l := \frac{\partial \mathbf{y}^{L+1}}{\partial \mathbf{z}^l}$, $l = 1, 2, \dots, L + 1$, above equation yields that

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

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 Quradrule Rule

2.3.1 Numerical Integration

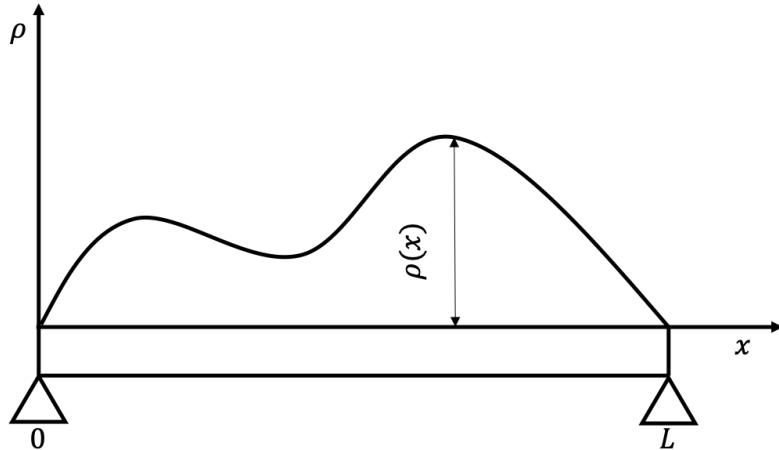
Integration is widely used in various fields, including integration transform, such as Laplace and Fourier transform, finite element method and boundary element method of partial differential equation, integral equation, variational method and statistics problem, in which many basic concepts, like probability 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 $f: \mathbb{R} \rightarrow \mathbb{R}$ 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 $a = x_1 < x_2 < \dots < x_n < x_{n+1} = b$, $\xi \in [x_i, x_{i+1}]$. By denoting $h_n = \max_{i=1, \dots, n-1} \{x_{i+1} - x_i\}$, we call that f is Riemann integrable in the interval if

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

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 $a < x_1 < \dots < x_n < b$ 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:

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

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

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

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:

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

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:

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

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.

Clebsch-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 $f_j(x) = x^{j-1}, j = 1, 2, 3, 4$

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

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)

$$\mathbf{y}' = \mathbf{y}'(t) := \begin{bmatrix} y'_1(t) \\ y'_2(t) \\ \vdots \\ y'_n(t) \end{bmatrix} = \begin{bmatrix} f_1(t, \mathbf{y}) \\ f_2(t, \mathbf{y}) \\ \dots \\ f_n(t, \mathbf{y}) \end{bmatrix} := \mathbf{f}(t, \mathbf{y}), \quad (2.3)$$

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

例2.4.1. An given n -th order ODE

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

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

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

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

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

定理2.4.1 (Existence and Uniqueness of the solution to IVPs). If $\mathbf{f} : \mathbf{R}^{n+1} \rightarrow \mathbf{R}^n$ 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

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

例2.4.3. Verifying the accuracy of the above schemes areo(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

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

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:

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

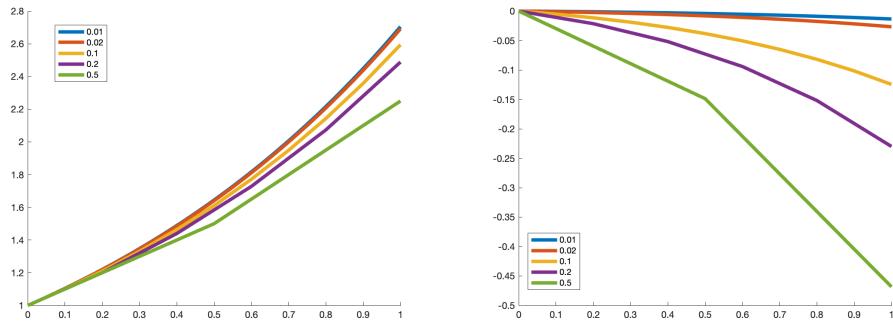


Figure 2.5: (Left)The Euler solution, (Right) Errors

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:

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

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 :

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

Runge-Kutta method

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

$$y_{m+1} = y_m + h(b_1 k_1 + \dots + b_s k_s),$$

where

$$\begin{aligned} k_1 &= f(x_m, y_m), \\ k_2 &= f(x_m + c_2 h, y_m + h a_{2,1} k_1), \\ k_3 &= f(x_m + c_3 h, y_m + h(a_{3,1} k_1 + a_{3,2} k_2)), \\ &\dots \\ k_s &= f(x_m + c_s h, y_m + h(a_{s,1} k_1 + \dots + a_{s,s-1} k_{s-1})) \end{aligned}$$

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

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

with the definition of momentum

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

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

$$y' = -2t y^2, y(0) = 1$$

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

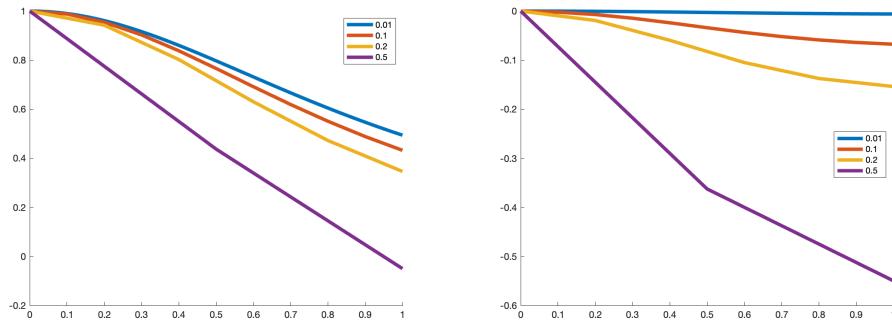


Figure 2.6: (Left)The Runge-Kutta solution with Heun formula, (Right) Errors

2.4.3 Stability Improvements

implicit scheme

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

$$y_{m+1} = y_m + h(b_1 k_1 + \dots + b_s k_s),$$

where k_i satisfies

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

The method is called implicit scheme. The common implicit methods are: implicit mid-point 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:

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

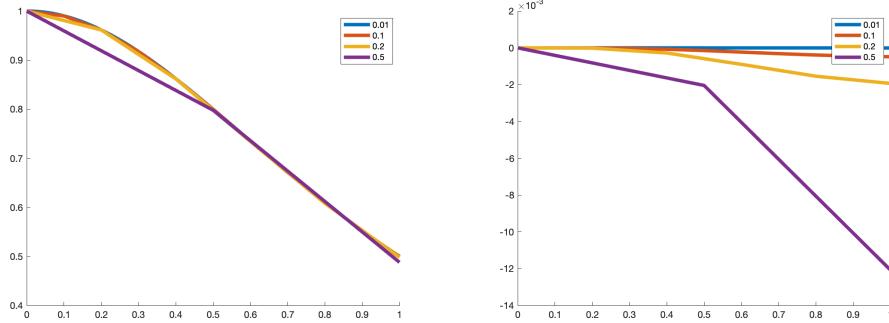


Figure 2.7: (Left)The Runge-Kutta solution with Hammer formula, (Right) Errors

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

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

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

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

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

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

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

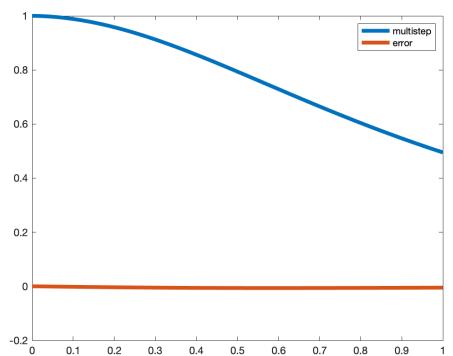
$$y' = -2xy^2, y(0) = 1$$

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 f = @(x,y)-2*x.*y.^2;y0 = 1;y1 = 0.9999;
2 h = 0.01; x = 0:h:1; temp1 = y0; temp2 = y1;
3 y = zeros(1,length(x)); y(1) = y0; y(2) = y1;
4 for j = 3:length(x)
5     y(j) = temp2 + h/2*(3*f(x(j),temp2) - f(x(
6         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

$$u'' = f(x, u, u'), a < x < b. \quad (2.7)$$

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

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

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

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

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

2.5.2 Galerkin method

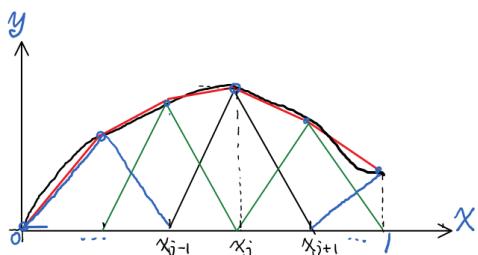
For the ease of representation, let us show how to calculate a finite element solution for a two-point boundary value problem

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

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

An nonuniform grid $0 = x_0 < x_1 < \dots < x_{n+1} = 1$ with $n + 1$ segment are considered for interval $[0, 1]$, and the length of segment i is defined as $h_i = (x_i - x_{i-1}), \forall i = 1, 2, \dots, n + 1$. 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 $\phi_k, k = 1, 2, \dots, n$. Practically, the piecewise continuous linear function space $V_h := \{\phi_k\}_{k=1}^n$ are constantly used, where

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



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

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

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

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

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

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

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

For the ease of understanding, we write out the above procedure within a concise script, which runs on Matlab or Octave platform. We emphasize 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.

```

1 %% - u''(x) = f(x), on (0,1); with u(a)=ua, u(b)=ub.
2 a=0; b=1; u = @(x) sin(pi*x); f = @(x) pi*pi*sin(pi*x); ua=u(a); ub=u(b); % EXE 1
3 n_elem = 16; n = n_elem; x = a + (0:n)*(b - a)/n; % the mesh grid
4 xi = [-0.577350269; 0.577350269]; wi = [0.5; 0.5]; % the quadrature
5 K = zeros(n+1, n+1); rhs = zeros(n+1, 1); K_temp =[2, -1; -1, 2]; % allocate memory
6 for e = 1:n
7     hi = x(e+1)-x(e); K(e:e+1,e:e+1) = K(e:e+1,e:e+1) + K_temp/hi; % assembling
8     qx = x(e)*0.5*(1-xi) + x(e+1)*0.5*(1+xi); % xi's real coord
9     rhs(e) = rhs(e) + sum(2.0*hi*wi.*f(qx).*((x(e+1)-qx)/hi)); % force node 1
10    rhs(e+1)= rhs(e+1) + sum(2.0*hi*wi.*f(qx).*((qx - x(e))/hi)); % force node 2
11 end
12 K(1,1) = 1.0; K(1,2) = 0; rhs(1) = ua; % apply u(a) = ua
13 K(n+1, n) = 0; K(n+1, n+1) = 1.0; rhs(n+1) = ub; % apply u(b) = ub
14 u_h = K\rhs; % solving Ax = b
15 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 $\phi_k(x), \forall k = 1, \dots, n$ 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

$$\begin{bmatrix} * & * & 0 & 0 & \cdots & 0 \\ * & * & * & 0 & \cdots & 0 \\ 0 & * & * & * & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & * & * \end{bmatrix} \quad (2.11)$$

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

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

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 $Aw = b$:

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

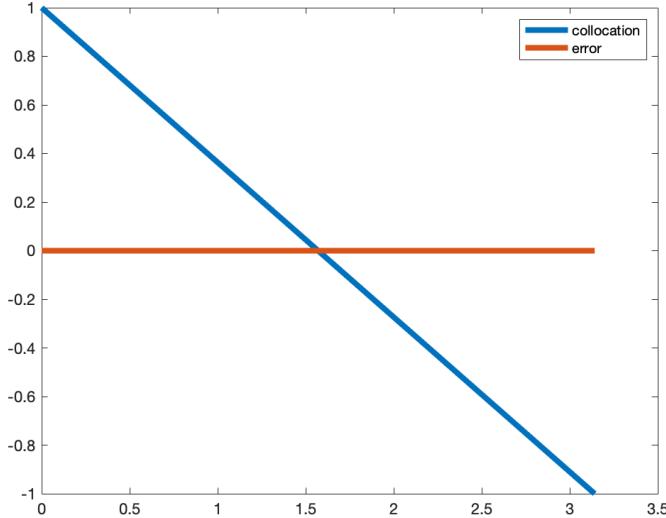
and the solution is computed as $u = Kw$, where

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

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 $\beta = 0.1$

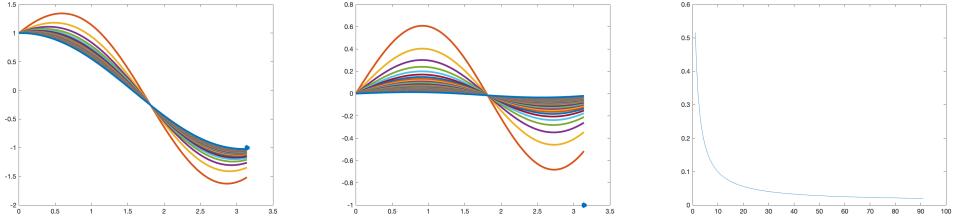


Figure 2.8: (Left) The solution of shooting method, (Middle) Errors, (Right) the objective function

2.5.5 Newton's method for nonlinear problems

Consider the following semi-linear problem:

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

Set

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

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

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

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

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

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

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

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

Practise

- Modify u in the example ??, record the L_2 or L_∞ errors in numerical solutions when $h = 10, 20, 40, 80, 160, 320$ and compute the convergent order.
- Modify the example of semi-linear problem and compute the convergent order of Newton method.
- Try to expand above method in the case of $h_x \neq h_y$, in which $(x, y) \in [a, b] \times [c, d]$
- Try to change the numerical order of equations.
- Further study on how to use Newton iterative method to find the inverse matrix and other methods that may speed up the solution of linear equation system.

软件包与参考材料

Exercise

Chapter 4

Numerical Solution to Partial Differential Equations

Equations involving partial derivatives of an unknown function is named as Partial Differential Equations(PDEs). PDEs are the fundamental mathematical technique for modeling continuous phenomena in many physical problem, such as linear elasticity equation, Maxwell's equation, Navier-Stokes equations, Schrödinger'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-dependent and dissipative process which are evolving toward a steady state, *Hyperbolic* PDEs, such as *wave equation*

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

describe time-dependent 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.

4.1 Parabolic Equations for Initial Value Problems(IVPs)

4.1.1 Explicit(iterative) numerical scheme

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

$$u_t = \nu u_{xx}, \quad 0 < x < 1, \quad t > 0, \quad (4.1)$$

$$u(0, t) = u(1, t) = 0, \quad t \geq 0, \quad (4.2)$$

$$u(x, 0) = u^0(x), \quad 0 \leq x \leq 1. \quad (4.3)$$

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 (4.1) could be obtained by separation of variable. Assuming the solution be in the form of $u(x, t) = f(x)g(t)$, it is true that

$$u(x, t) = \sum_{m=1}^{\infty} a_m e^{-(m\pi)^2 t} \sin m\pi x, \quad (4.4)$$

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 , (4.4) 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 (4.1) holds at (x_j, t_n) , which means spatial point x_j at time step t_n

$$U_j^{n+1} = U_j^n + \mu(U_{j+1}^n - 2U_j^n + U_{j-1}^n), \quad (4.5)$$

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 (4.5) is referred as an **explicit Scheme**.

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

```

1 Given  $\nu, f, [a, b]$  and  $N, T, \tau$ ;
2  $h = (b - a)/N$  and set  $x_j = j * h, \forall j = 0, 1, \dots, N$ ;
3  $u = \text{zeros}(N+1, T+1)$  ;
4 for  $n = 1, 2, \dots, T$  do
5    $u(0, n) = a(n\tau); u(N, n) = b(n\tau);$ 
6   for  $j = 1, 2, \dots, N - 1$  do
7      $| U_j^{n+1} = U_j^n + \mu(U_{j+1}^n - 2U_j^n + U_{j-1}^n);$ 
8   end
9 end

```

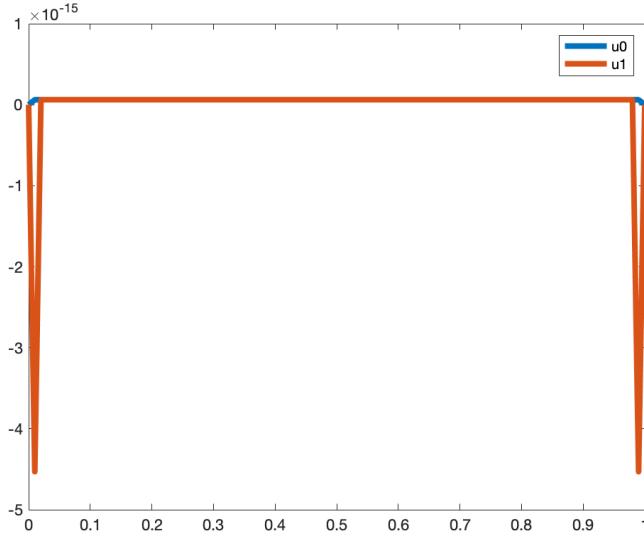
例4.1.1. Please calculate by (4.5) with $\nu = 5, f(x, 0) = \cos \frac{\pi x}{2}, a(0, t) = 0, b(1, t) = 0$. Suggested computational parameters $N = 100, T = 1, \tau = 0.0015$. The result is shown as following figure

4.1.2 Error estimation for explicit scheme

Does we do the right thing to solve the parabolic equation (4.1)?

定理4.1.1 (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

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



Truncation Error:

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

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

定理4.1.2 (Convergent). Using fixed initial and boundary values and $\mu = \tau/(h)^2$, and let $\tau \rightarrow 0, h \rightarrow 0$. If on any given position $(x^*, t^*) \in (0, 1) \times (0, T)$,

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

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

性质4.1.1 (Stability Condition). The explicit scheme (4.5) is convergent if $\mu := \frac{\tau}{h^2} \leq \frac{1}{2}$.

4.1.3 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 $h \rightarrow 0$. The following scheme is another good choice

$$U_j^{n+1} = U_j^n + \mu(U_{j+1}^{n+1} - 2U_j^{n+1} + U_{j-1}^{n+1}) \quad (4.6)$$

The implicit scheme yields

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

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4.1.2 (Fourier mode error estimation). Error analysis for explicit scheme (4.5) 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 (4.5) divergent! However in the case of convergence, there exist a K independent of k , which makes

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

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 (4.6) 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,

4.1.4 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

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

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

$$-\theta \mu U_{j-1}^{n+1} + (1 + 2\theta\mu) U_j^{n+1} - \theta \mu U_{j+1}^{n+1} = [1 + (1 - \theta)\mu \delta_x^2] U_j^n \quad (4.7)$$

例4.1.3 (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 $U_j^n = \lambda^n e^{ik(j\Delta x)}$ as a solution of scheme (4.7). It finally leads to

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

According to the above criteria, the θ -scheme for (4.7) is stable if $\mu \leq \frac{1}{2}(1 - 2\theta)^{-1}$ 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4.1.3 (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 (4.7), satisfying

$$U_{min} \leq U_j^n \leq U_{max}, \quad (4.8)$$

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4.1.4 (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 heavily 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

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

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

Explicit V.S. Implicit time step Δt , grid space Δx and Δy

$$U_{r,s}^n \approx u(x_r, y_s, t_n), \quad \forall r = 0, \dots, Nx, s = 0, \dots, Ny.$$

Explicit scheme

$$\frac{1}{\Delta t}(U_{r,s}^{n+1} - U_{r,s}^n) = \frac{b}{(\Delta x)^2}(U_{r+1,s}^n - 2U_{r,s}^n + U_{r-1,s}^n) - \frac{b}{(\Delta y)^2}(U_{r,s+1}^n - 2U_{r,s}^n + U_{r,s-1}^n)$$

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

$$\frac{1}{\Delta t}(U_{r,s}^{n+1} - U_{r,s}^n) = \frac{b}{(\Delta x)^2}(U_{r+1,s}^{n+1} - 2U_{r,s}^{n+1} + U_{r-1,s}^{n+1}) - \frac{1}{(\Delta y)^2}(U_{r,s+1}^{n+1} - 2U_{r,s}^{n+1} + U_{r,s-1}^{n+1})$$

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4.1.5. Consider a two-dimensional heat equation on the unit square

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

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(改成字母Z?), such as the first plot in Fig.4.1.

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

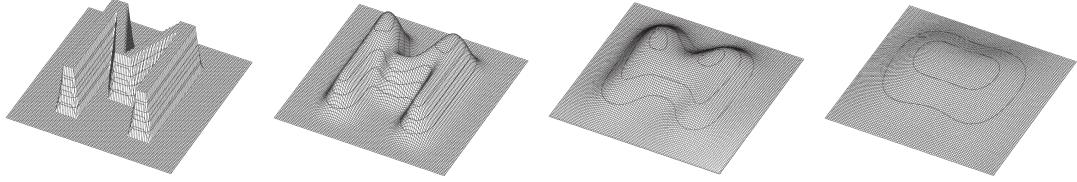


Figure 4.1: From left to right: the numerical solution at $t = 0, 0.001, 0.004$ and $t = 0.01$

4.2 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

$$\begin{cases} \frac{\partial u}{\partial t} + a(x, t) \frac{\partial u}{\partial x} = 0, & \forall t, x, \\ u(x, 0) = u^0(x), & \forall x. \end{cases} \quad (4.9)$$

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

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

4.2.1 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 (4.9), which yields finite difference equaiton

$$\frac{U_j^{n+1} - U_j^n}{\Delta t} + a \frac{U_j^n - U_{j-1}^n}{\Delta x} = 0.$$

A simplification results into a iterative scheme

$$U_j^{n+1} = U_j^n - \frac{a \Delta t}{\Delta x} (U_j^n - U_{j-1}^n) := (1 - \nu)U_j^n + \nu U_{j-1}^n, \quad (4.10)$$

where $\nu = \frac{a\Delta t}{\Delta x}$. From algebraic point of view, iterative scheme (4.10) 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 (4.10) is $\nu \leq 1$.

For the ease of analysis, it is preferred to define $\nu = |a|\Delta t/\Delta x$ 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*

$$U_j^{n+1} = \begin{cases} U_j^n - \nu(U_{j+1}^n - U_j^n) = (1 + \nu)U_j^n - \nu U_{j+1}^n, & a < 0, \\ U_j^n - \nu(U_j^n - U_{j-1}^n) = (1 - \nu)U_j^n + \nu U_{j-1}^n, & a > 0, \end{cases} \quad (4.11)$$

4.2.2 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 intrinsic scheme since it is proposed in 1960. Let us skip the details and only gives its formulation

$$U_j^{n+1} = \frac{1}{2}\nu(1 + \nu)U_{j-1}^n + (1 - \nu^2)U_j^n - \frac{1}{2}\nu(1 - \nu)U_{j+1}^n. \quad (4.12)$$

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 emphasize the stability of the scheme, which is independent on the sign of its Fourier modes Fourier mode analysis gives its amplification factor being

$$|\lambda(k)|^2 = 1 - 4\nu^2(1 - \nu^2) \sin^4 \frac{k}{2} \Delta x.$$

It shows that the Lax-Wendroff scheme is stable if $\nu \leq 1$.

例4.2.1. 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

$$u(x, 0) = \exp -10(4x - 1)^2,$$

4.2.3 Leap-Frog scheme

In this scheme, three different time-levels are concerned, and it is another important scheme other than the Lax-Wendroff scheme.

$$\frac{U_j^{n+1} - U_j^{n-1}}{2\Delta t} + a \frac{U_j^{n+1} - U_{j-1}^n}{2\Delta x} = 0.$$

A simplification results into a iterative scheme

$$U_j^{n+1} = U_j^{n-1} - \nu(U_{j+1}^n - U_{j-1}^n), \quad (4.13)$$

where $\nu = a\Delta t/\Delta x$ is the same as usual.

Hyperbolic equation system.

例4.2.2. 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

$$\begin{aligned} V_j^{n+1/2} &= V_j^{n-1/2} - \nu(W_{j+1/2}^n - W_{j-1/2}^n), \\ W_{j+1/2}^{n+1} &= W_{j+1/2}^n - \nu(V_{j+1}^{n+1/2} - V_j^{n+1/2}). \end{aligned} \quad (4.14)$$

4.2.4 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 $\Delta x \rightarrow 0$ and $\Delta t \rightarrow 0$, it requires

The method be *consistent*, which promises the local truncation error goes to 0 as $\Delta t \rightarrow 0$. 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

$$\tau = \frac{\mathcal{N}(q^n) - q^{n+1}}{\Delta t}$$

We say that the method is *consistent* if τ vanished as $\Delta t \rightarrow 0$ for all smooth $q(x, t)$ satisfying the differential equation. It is usually straightforward when Taylor expansions 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 $\Delta t, \Delta x \rightarrow 0$

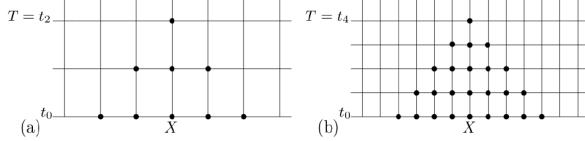


Fig. 4.3. (a) Numerical domain of dependence of a grid point when using a three-point explicit finite difference method, with mesh spacing Δx^a . (b) On a finer grid with mesh spacing $\Delta x^b = \frac{1}{2}\Delta x^a$.

For a hyperbolic system with characteristic wave speeds λ^p ,

$$\frac{\Delta x}{\Delta t} \geq \max_p |\lambda^p|, \quad p = 1, \dots, m.$$

This condition is necessary but not sufficient !

定理4.2.1 (Lax Equivalence Theorem). For a consistent difference approximation to a well-posed linear evolutionary problem, which is uniformly solvable in the sense of $\|B^{-1}\| \leq K_1 \Delta t$, where $B = \frac{1}{\Delta t} - \theta \frac{\delta_x^2}{(\Delta x)^2}$, the stability of the scheme is necessary and sufficient for convergence.

定理4.2.2 (Von Neumann Theorem). A necessary condition for stability is that there exist a constant K' such that

$$|\lambda(\mathbf{k})| \leq 1 + K' \Delta t, \forall \mathbf{k}, \quad (4.15)$$

for every eigenvalue $\lambda(\mathbf{k})$ of the amplification matrix $G(\mathbf{k})$

where $u^{n+1}(\mathbf{k}) = G(\mathbf{k})u^n(\mathbf{k})$.

4.3 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

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{f}(\mathbf{u})}{\partial x} = 0, \quad (4.16)$$

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.

4.3.1 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.4.3.1.

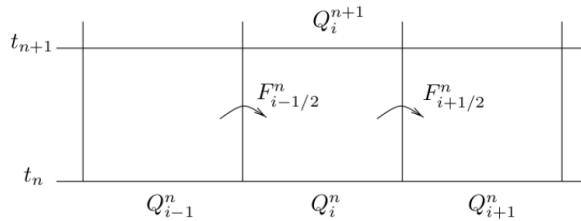


Figure 4.2: The concept of finite volume approximation.

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 (4.16) 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 Δx :

$$\frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \mathbf{u}(x, t_{n+1}) dx = \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \mathbf{u}(x, t_n) dx - \frac{\Delta t}{\Delta x} \left[\int_{t_n}^{t_{n+1}} \mathbf{f}(\mathbf{u}(x_{i-\frac{1}{2}}, t)) dt - \int_{t_n}^{t_{n+1}} \mathbf{f}(\mathbf{u}(x_{i+\frac{1}{2}}, t)) dt \right].$$

According to the definition of mean values on cell \mathbf{U} and flux function \mathbf{F} , we reach to

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{\Delta x} \left(F_{i+\frac{1}{2}}^n - F_{i-\frac{1}{2}}^n \right), \quad (4.17)$$

where $\mathbf{F}_{i-\frac{1}{2}} \approx \frac{1}{\Delta t} \int_{t_n}^{t_{n+1}} \mathbf{f}(\mathbf{u}(x_{i-1/2}, t)) dt$.

For the purpose of evaluate (4.17) 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

$$\mathbf{F}_{i-1/2}^n = \mathcal{F}(U_{i-1}, U_i), \quad (4.18)$$

where \mathcal{F} is referred as *numerical flux function*. Finally, the numerical methods reads

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{\Delta x} (\mathcal{F}(U_{i-1}^n, U_i^n) - \mathcal{F}(U_i^n, U_{i+1}^n)). \quad (4.19)$$

Practically, there some different choice for the numerical flux, $\mathcal{F}(U_{i-1}^n, U_{i+1}^n) = \frac{1}{2}[f(u_{i-1}^n) + f(u_i^n)]$ 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

$$U_i^{n+1} = U_i^n - \lambda \frac{\Delta t}{\Delta x} (U_i^n - U_{i-1}^n). \quad (4.20)$$

例4.3.1 (此处应有一例). content

4.3.2 Riemann Solver

线性化 例: the advection equation

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

$$w_t^p + \lambda^p w_x^p = 0, \quad p = 1 \dots m.$$

$$\begin{cases} \omega_t + \lambda \omega_x = 0, \\ \omega(x, 0) = \omega_0(x) \end{cases}$$

solved with the method of characteristics $\omega(x, t) = \omega_0(x - \lambda t)$.

Boundary condition for IVP($a \leq x \leq b$)?

- 依赖域(Domain of dependence) 左图
 - 影响域(Range of Influence) 右图
- The hyperbolic equation with initial data

$$q_0(x) = \begin{cases} q_l & x < 0 \\ q_r & x > 0 \end{cases}$$

is known as the Riemann problem.

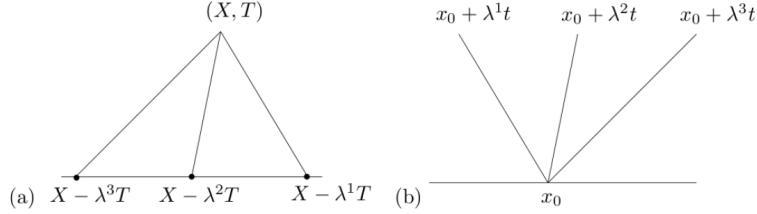


Fig. 3.2. For a typical hyperbolic system of three equations with $\lambda^1 < 0 < \lambda^2 < \lambda^3$, (a) shows the domain of dependence of the point (X, T) , and (b) shows the range of influence of the point x_0 .

(R. Leveque, 2002)

For the linear constant-coefficient system, the solution is

$$\begin{aligned} q(x, t) &= q_l + \sum_{p: \lambda^p < x/t} [l^p(q_r - q_l)] r^p \\ &= q_r - \sum_{p: \lambda^p \geq x/t} [l^p(q_r - q_l)] r^p \end{aligned}$$

Roe 的方案 Recall the numerical method for Conservation Law

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x} [\mathcal{F}(Q_i^n, Q_{i+1}^n) - \mathcal{F}(Q_i^n, Q_{i+1}^n)],$$

A linearized choice of the numerical flux based on the Godunov's method for the nonlinear problems. Define $|A| = R|\Sigma|R^{-1}$, where $|\Sigma| = \text{diag}(|\lambda^p|)$, then we can derive the Roe's flux as

$$F_{i-\frac{1}{2}}^n = \frac{1}{2} [f(Q_{i-1}) + f(Q_i)] - \frac{1}{2}|A|[Q_{i-1} + Q_i]$$

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.

Recall the solution to the Riemann problem form a linear system

$$Q_i - Q_{i-1} = \sum_{p=1}^m [l^p(Q_{i+1} - Q_i)] r^p = \sum_{p=1}^m \mathcal{W}_{i-\frac{1}{2}}^p$$

If Δt is small enough, waves from adjacent cells do not interact!

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

$$F_{i-\frac{1}{2}}^n = f(Q_{i-1}) + \sum_{p=1}^m (\lambda^p)^- \mathcal{W}_{i-\frac{1}{2}}^p,$$

or

$$F_{i-\frac{1}{2}}^n = f(Q_i) + \sum_{p=1}^m (\lambda^p)^+ \mathcal{W}_{i-\frac{1}{2}}^p,$$

where $\lambda^+ = \max(\lambda, 0)$ and $\lambda^- = \min(\lambda, 0)$ is an upwind choice.

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

$$Q_i^{n+1} = \frac{1}{\Delta x} \int_{C_i} \tilde{q}^n(x, t_{n+1}) dx.$$

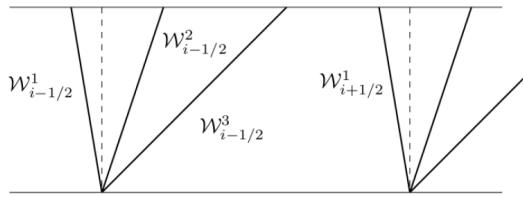


Fig. 4.7. An illustration of the process of Algorithm 4.1 for the case of a linear system of three equations. The Riemann problem is solved at each cell interface, and the wave structure is used to determine the exact solution time Δt later. The wave $W_{i-1/2}^2$, for example, has moved a distance $\lambda^2 \Delta t$ into the cell.

4.3.3 Total Variation Diminision(TVD)

Recall the numerical method for Conservation Law

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x} [\mathcal{F}(Q_i^n, Q_{i+1}^n) - \mathcal{F}(Q_i^n, Q_{i+1}^n)],$$

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

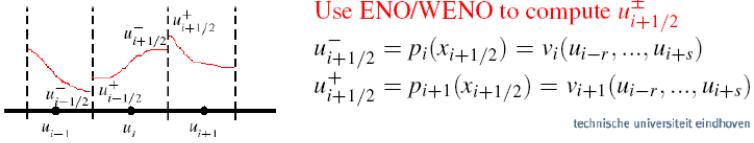
$$h(a, b) = 0.5(f(a) + f(b) - \alpha(b - a)),$$

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 $q(x)$.

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

$$p_i(x) = q(x) + \mathcal{O}(\Delta^k) \quad \forall x \in C_i, i = 0, 1, \dots, N.$$



Finally, one can evaluate u at each cell interface ($u_{i+1/2}^-$ and $u_{i+1/2}^+$).

例4.3.2 (此处应再有一例). content

4.4 Elliptic Equations for BVPs

4.4.1 Five-point scheme for elliptic equations

Let us introduce the basic two dimensional model problem of elliptic equations at first.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + f(x, y) = 0, (x, y) \in \Omega, \quad (4.21)$$

$$u = 0, (x, y) \in \partial\Omega, \quad (4.22)$$

Let us further compare FEM with FDM. Denote $u_{i,j}$ as the function values on grid points, then

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{-u_{i-1,j} + 2u_{i,j} - u_{i+1,j}}{h_x^2}, \quad \frac{\partial^2 u}{\partial y^2} \approx \frac{-u_{i,j-1} + 2u_{i,j} - u_{i,j+1}}{h_y^2}.$$

Assuming that $h_x = h_y = h$, there yields

$$-\Delta u := -(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) \approx \frac{1}{h^2} \begin{pmatrix} & -u_{i,j+1} & \\ -u_{i-1,j} & 4u_{i,j} & -u_{i+1,j} \\ & -u_{i,j-1} & \end{pmatrix} \quad (4.23)$$

Sparse Laplace operator in Matlab:

There are two different ways computing Sparse Laplace operator in Matlab

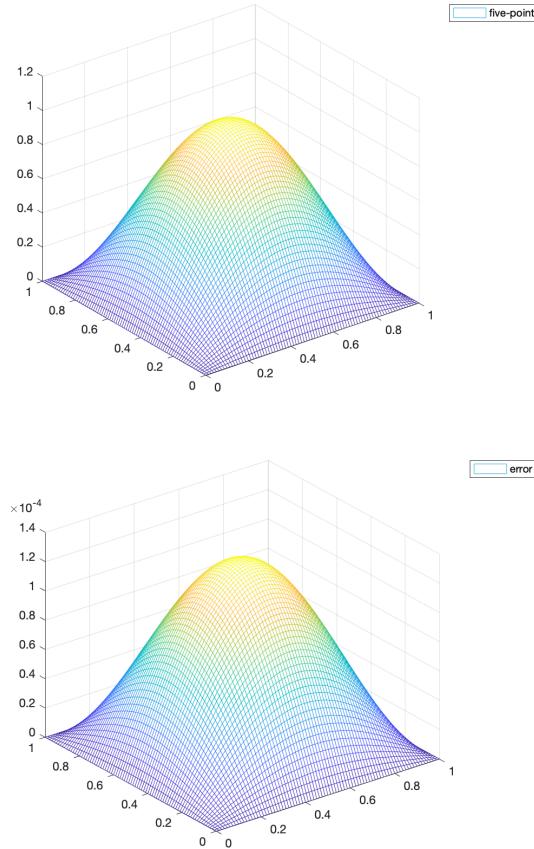
- kron — based on the inner unknown points(the number is $(n - 1) \times (n - 1)$);
- spLaplacian.m — based on all points(the number is $(n + 1) \times (n + 1)$).

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4.4.1 (Five-point differential scheme). Using the above five-point differential scheme to solve the following poisson equation

$$\begin{aligned}-\Delta u &= \pi^2 \sin(\pi x) \sin(\pi y), (x, y) \in [0, 1] \times [0, 1], \\ u &= 0, (x, y) \in \partial[0, 1] \times [0, 1]\end{aligned}$$

Using the above schemes, and m file of spLaplacian.m, the numerical result and error are shown in the following figure:



4.4.2 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4.4.1 (Convergent). Using fixed boundary values, and let $h \rightarrow 0$. If on any given position $(x^*, y^*) \in (0, 1) \times (0, 1)$,

$$u_{ij} \rightarrow u(x_i, y_j), \forall x_i \rightarrow x_i^*, y_j \rightarrow y_j^*.$$

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:

$$L_h e_{ij} = -T_{ij}. \quad (4.24)$$

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

$$L_h \phi_{ij} = L_h e_{ij} + \frac{1}{4}T L_h \psi_{ij} = -T_{ij} + T \geq 0 \quad (4.25)$$

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

$$e_{ij} \leq \phi_{ij} \leq \frac{1}{8}T = \frac{1}{9}6h^2(M_{xxxx} + M_{yyyy}). \quad (4.26)$$

□

4.5 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

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

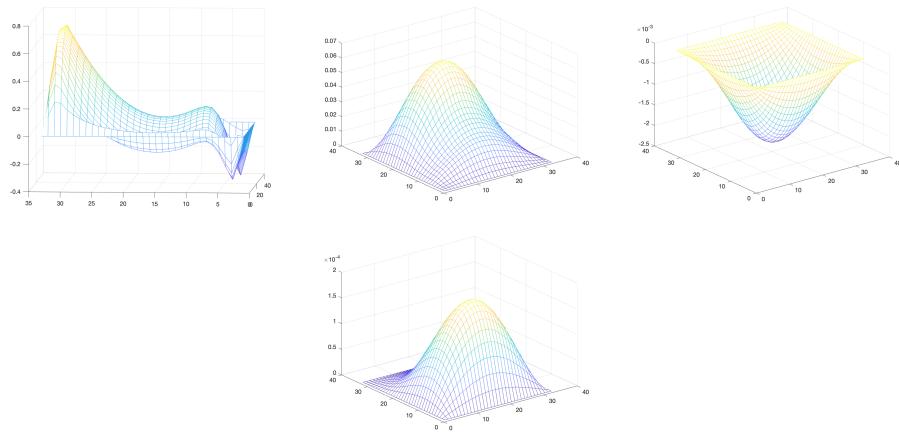
Considering a transfer matrix P and construct a new systems of equations on coarse mesh as

$$P^T A P \mathbf{x}_c = P^T \mathbf{b}_c$$

By solving this new problem, the corresponding solution \mathbf{x}_c^k and error \mathbf{e}^k are given as:

$$r^k = \mathbf{b} - AP\mathbf{x}_c^k; \quad Ae^k = r^k \quad (4.27)$$

例4.5.1 (Multigrid method). Using multigrid method to solve the problem in example 4.4.1. The iteration is convergent after 14 steps, here we show four errors(step 1, 5, 10, 14) in those steps in the following figure.



4.6 Mathematical Framework for FEM

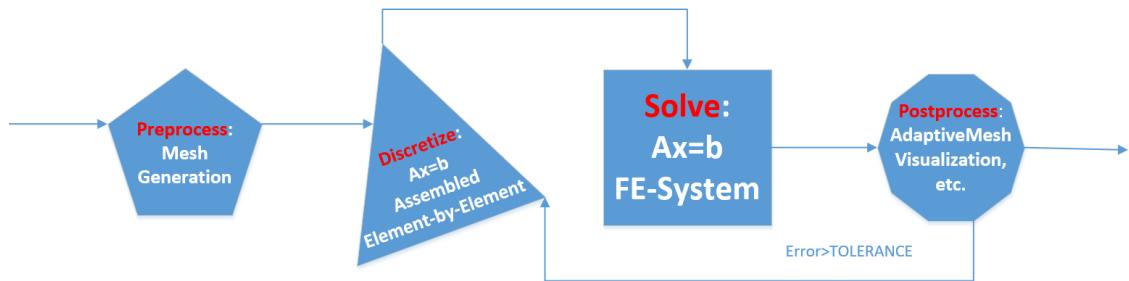
In the previous section, we present the Galerkin method for two-point boundary value problems for ODEs. Considering PDE-based BVPs, the shape of the spatial domain should be discretized by quadrilaterals or triangles in two dimensions and hexahedra or tetrahedra in three dimensions. Practical applications in scientific and engineering computing result this methodology 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。有限元方法的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4.6.1 (Elasticity equation). d

例4.6.2. 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

$$u = 2 \ln \left(\frac{B+1}{B(x^2+y^2)+1} \right),$$

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4.6.3 (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 $\mathbf{u} = (u, v) = (1, 0)$, $\forall 0 \leq x \leq 1$ 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 emphasize 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.

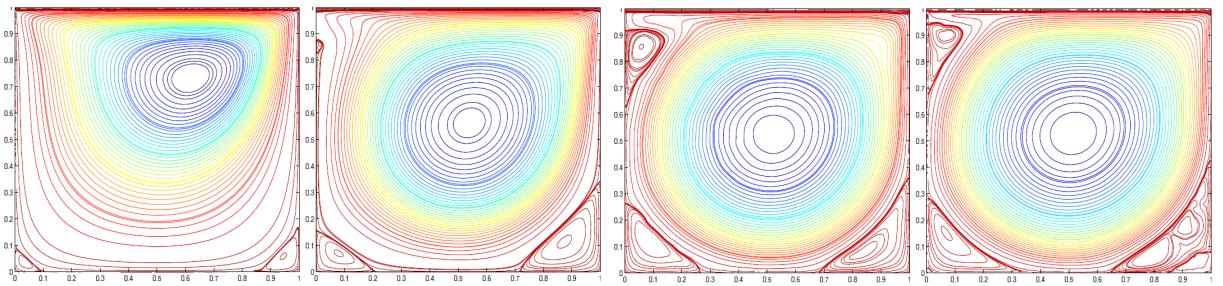


Figure 4.3: The contours of stream function when Reynolds number equals to 100, 1000, 5000 and 10000.

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 pricion. 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.

4.7 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 $100 \times 100 \times 100$ 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

$$Ax = b \quad (4.28)$$

以及稀疏矩阵特征值问题

$$Bx = \lambda x \quad (4.29)$$

where A and B are sparse matrix, which is defined as the following

定义4.7.1 (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。

4.7.1 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稳定性(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

```

1 for  $j = 0, 1, \dots, \text{until convergence}$  do
2    $\mathbf{r}_j = \mathbf{b} - A\mathbf{x}_j;$ 
3    $\alpha_j = (\mathbf{r}_j, \mathbf{r}_j) / (A\mathbf{r}_j, \mathbf{r}_j);$ 
4    $\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha_j \mathbf{r}_j;$ 
5 end

```

极小残量(Minimal Residual)法

```

1 for  $j = 0, 1, \dots$ , until convergence do
2   |  $\mathbf{r}_j = \mathbf{b} - A\mathbf{x}_j;$ 
3   |  $\alpha_j = (\mathbf{A}\mathbf{r}_j, \mathbf{r}_j) / (\mathbf{A}\mathbf{r}_j, \mathbf{A}\mathbf{r}_j);$ 
4   |  $\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha_j \mathbf{r}_j;$ 
5 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

$$\mathcal{K}_m(A, \mathbf{r}_0) = \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, \dots, A^{m-1}\mathbf{r}_0\}.$$

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:

```

1 Choose a unit vector  $\mathbf{v}_1$ ;
2 for  $j = 1, 2, \dots, m$  do
3   | Compute  $h_{ij} = (A\mathbf{v}_j, \mathbf{v}_i)$ ,  $\forall i = 1, 2, \dots, j$ ;
4   | Compute  $\mathbf{u}_j = A\mathbf{v}_j - \sum_{i=1}^j h_{ij} \mathbf{v}_i$ ;
5   |  $h_{j+1,j} = \|\mathbf{u}_j\|_2$ ;
6   | if  $h_{j+1,j} == 0$  then
7     |   | Stop
8   | else
9     |   |  $\mathbf{v}_{j+1} = \mathbf{u}_j / h_{j+1,j}$ 
10  | end
11 end

```

接着实施Arnoldi-Modified Gram-Schmidt正交化过程:

```

1 Choose a unit vector  $\mathbf{v}_1$ ;
2 for  $j = 1, 2, \dots, m$  do
3   Compute  $\mathbf{u}_j = A\mathbf{v}_j$ ;
4   for  $i = 1, \dots, j$  do
5      $h_{i,j} = (\mathbf{u}_j, \mathbf{v}_i)$ ;
6      $\mathbf{u}_j = \mathbf{u}_j - h_{i,j}\mathbf{v}_i$ ;
7   end
8    $h_{j+1,j} = \|\mathbf{u}_j\|_2$ ;
9   if  $h_{j+1,j} == 0$  then
10    | Stop
11   else
12    |  $\mathbf{v}_{j+1} = \mathbf{u}_j / h_{j+1,j}$ 
13   end
14 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  $H_m = \{h_{i,j}\}_{i,j=1,2,\dots,m}$ , set  $H_m = 0$ ;
3 for  $j = 1, 2, \dots, m$  do
4   Compute  $\mathbf{u}_j = A\mathbf{v}_j$ ;
5   for  $i = 1, \dots, j$  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    | Compute  $\mathbf{v}_{j+1} = \mathbf{u}_j / h_{j+1,j}$ 
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  $H_m = \{h_{i,j}\}_{1 \leq i \leq (m+1)}^{1 \leq j \leq m}$ , set  $H_m = 0$ ;
3 for  $j = 1, 2, \dots, m$  do
4   Compute  $\mathbf{u}_j = A\mathbf{v}_j$ ;
5   for  $i = 1, \dots, j$  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    | Compute  $\mathbf{v}_{j+1} = \mathbf{u}_j/h_{j+1,j}$ 
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:

```

1 Compute  $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$ ,  $\mathbf{p}_0 = \mathbf{r}_0$ ;
2 for  $j = 0, 1, \dots, until convergence$  do
3    $\alpha_j = (\mathbf{r}_j, \mathbf{r}_j)/(A\mathbf{p}_j, \mathbf{p}_j)$ ;
4    $\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha_j\mathbf{p}_j$ ;
5    $\mathbf{r}_{j+1} = \mathbf{r}_j - \alpha_j A\mathbf{p}_j$ ;
6    $\beta_j = (\mathbf{r}_{j+1}, \mathbf{r}_{j+1})/(\mathbf{r}_j, \mathbf{r}_j)$ ;
7    $\mathbf{p}_{j+1} = \mathbf{r}_{j+1} + \beta_j\mathbf{p}_j$ ;
8 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

4.7.2 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 $M^{-1}A$ 的条件数相对较小, 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4.7.1.

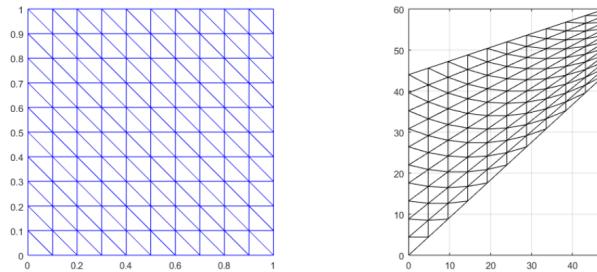
4.8 Finite element mesh - generation and adaptivity

In finite element applications, algorithm for mesh generation is essential for practical using. Finite element meshes with good quality will not only improve 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 challdge for various application. For the purpose of generality, only the triangular mesh generation and adaptation are concerned.

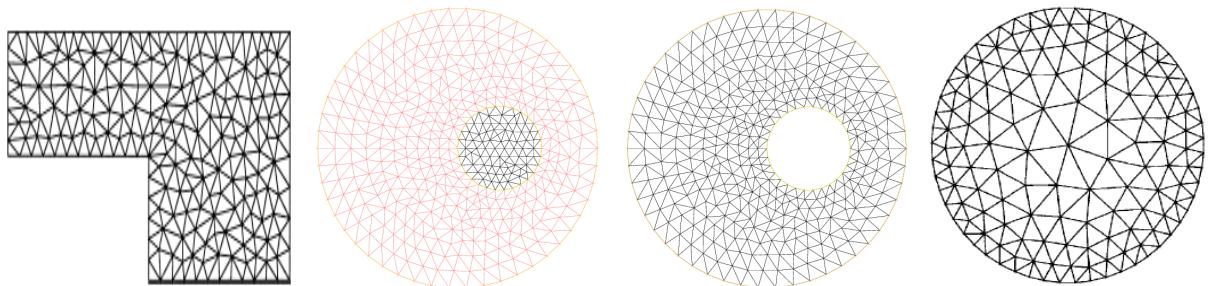
The conformity of the finite element mesh is essential for a legal approximation. It requires that 合法性:一个单元的结点不能落入其他单元内部, 在单元边界上的结点均应作为单元的结点, 不可丢弃。相容性:单元必须落在待分区域内部, 不能落入外部, 且单元并集等于待分区域。逼近精确性:待分区域的顶点(包括特殊点)必须是单元的结点, 待分区域的边界(包括特殊边及面)被单元边界所逼近。

Mesh quality is evaluated based on the quality of each element shape. The best element shape is the equilateral triangle. secondly, a smooth transition between the

neighboured element is important for continuous 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 $P_i(i=1,2,\dots,N)$ 构成的点集为S，称满足下列条件的点集 V_i 为Voronoi多边形。其中， V_i 满足下列条件： $V_i = \{X : X - P_i \geq X - P_j, \forall j \neq i\}$ ， V_i 为凸多边形，称 V_i 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 T_k ，称集合 T_k 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4.8.1 (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

$$\nabla \cdot \left(\frac{1}{\omega} \nabla \xi(x, y) \right) = 0, \quad \nabla \cdot \left(\frac{1}{\omega} \nabla \eta(x, y) \right) = 0,$$

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

$$\nabla \cdot (\omega \nabla \mathbf{x}) = 0, \quad \nabla \cdot (\omega \nabla \mathbf{y}) = 0, \quad (4.30)$$

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

$$X_{i,j} = \frac{(\omega_{i-\frac{1}{2},j} X_{i-1,j} + \omega_{i+\frac{1}{2},j} X_{i+1,j} + \omega_{i,j-\frac{1}{2}} X_{i,j-1} + \omega_{i,j+\frac{1}{2}} X_{i,j+1})}{(\omega_{i,j-\frac{1}{2}} + \omega_{i,j+\frac{1}{2}} + \omega_{i,j-\frac{1}{2}} + \omega_{i,j+\frac{1}{2}})}, \quad (4.31)$$

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 (4.31) 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 (4.31) 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

$$\omega(u) = \sqrt{\alpha + \beta |\nabla u|^2 + \gamma u^2}, \quad (4.32)$$

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) :

$$u^* = u + \nabla_x u(x^{n+1} - x^n) + \nabla_y u(y^{n+1} - y^n), \quad (4.33)$$

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 (4.33) is accurate enough.

4.9 Mixed FEM and Navier-Stokes equation solver

参考NotesZou.pdf的8-9章，写3-5页

Exercises

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.