重庆大学
CHONGQING UNIVERSITY

数值分析第一章

1.1 (1) 有限次四则运算

(2) 尽量避免两个相近数作减法计算

分母绝对值远小于分子绝对值

(3) 数值分析处理截断误差和舍入误差

(4) 有效数字越多相对误差越小

1.3 $x_1 = 0.3040$ → 有效数位 4

→ $\Sigma(x_1)$ 误差限 0.5×10^{-4}

→ $\Sigma_r(x_1)$ 相对误差限 $\frac{\Sigma(x_1)}{x_1} = 1.64 \times 10^{-4}$

1.5 正方形面积误差算法 $\Delta S = 2L \cdot \Delta L$

1.6 跑步成绩误差 $t' = t \times \frac{L+\Delta L}{L}$

1.11 精确度改进方法

① $y = \frac{1}{1+2x} - \frac{1-x}{1+x} \Rightarrow \frac{2x^2}{(1+2x)(1+x)}$ 二合一

② $y = \sqrt{x+\frac{1}{x}} - \sqrt{x-\frac{1}{x}} \Rightarrow \frac{2}{x(\sqrt{x+\frac{1}{x}} + \sqrt{x-\frac{1}{x}})}$ 二合一

③ $y = \frac{1-\cos 2x}{x} = \frac{2\sin^2 x}{x}$ 升幂

④ $y = \sin(x+\varepsilon) - \sin x = 2\cos(x+\frac{\varepsilon}{2})\sin\frac{\varepsilon}{2}$ 升幂

重庆大学 CHONGQING UNIVERSITY

数值分析第二章

2.1(4)

$A=\begin{pmatrix}1&1\\0&2\end{pmatrix}$ $\|A\|_1$-范数 $=\max\{1,3\}=3$ 列绝对值和的最大值

$\|A\|_2$ 二范数 $A^TA=\begin{pmatrix}1&1\\1&5\end{pmatrix}$

$\det(\lambda I-A^TA)=\begin{pmatrix}\lambda-1&-1\\-1&\lambda-5\end{pmatrix}=0$

求特征值 $\lambda=3\pm\sqrt{5}$

$\|A\|_2=\sqrt{\lambda_{max}}=2.287$

$\rho(A)$ 谱范数 $\det(\lambda I-A)=(\lambda-1)(\lambda-2)=0$

$\lambda_{max}=2$ $\rho(A)=2$

$\mathrm{cond}(A)_2=\|A\|_2\|A^{-1}\|_2$

$\|A\|_\infty$ 行范数 行绝对值和的最大值

2.1(5)(6) 2.11(1) 略

2.14 $\begin{pmatrix}2&1&1\\3&1&2\\1&2&2\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}=\begin{pmatrix}4\\6\\5\end{pmatrix}$ 右端扰动 $\|\delta_b\|=\frac{1}{2}\times10^{-5}$ 求相对误差

A b $A^{-1}=\begin{vmatrix}\frac{2}{3}&0&-\frac{1}{3}\\-\frac{4}{3}&-1&\frac{1}{3}\\-\frac{5}{3}&1&\frac{1}{3}\end{vmatrix}$

$\|A\|_\infty=6$ $\|A^{-1}\|_\infty=3$ $\|b\|_\infty=6$

$$\frac{1}{\|A\|\cdot\|A^{-1}\|}\frac{\|\delta_b\|}{\|b\|}\le\frac{\|\delta_x\|}{\|x\|}\le\|A\|\cdot\|A^{-1}\|\frac{\|\delta_b\|}{\|b\|}$$

数值分析 2 前

2.1 (1) Gauss消去 主元素为0 计算中断

主元素太小误差增大

(2) Gauss计算工作量乘除法次 $(\frac{n^3}{3})$

平方根法角矩对称正定结性 $(\frac{n^3}{6})$

(3) 直接LU法 $(\frac{n^3}{3})$

追赶法对角占优三对角 $(5n-3)$

→高斯法

$$\left(\begin{array}{ccc|c} 2 & 3 & 1 & 1 \\ 4 & 12 & 2 & 2 \\ 3 & 23 & 3 & 3 \end{array}\right) \Rightarrow \left(\begin{array}{ccc|c} 2 & 3 & 1 & 1 \\ 0 & -5 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{3}{2} & \frac{3}{2} \end{array}\right) \Rightarrow \left(\begin{array}{ccc|c} 2 & 3 & 1 & 1 \\ 0 & -5 & 0 & 0 \\ 0 & 0 & \frac{3}{4} & \frac{3}{4} \end{array}\right) \Rightarrow \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$$

→平方根法(对称正定)

$A = LL^T$ 解L

→LU分解

$A = LU$ $Ly = b \to y$ $Ux = y \to x$

→追赶法

将三对角矩阵分解为上三角和下三角的乘积

数值分析第三章

3.2 Jacobi 迭代法 Gauss-Seidei 迭代法

3.4 略

3.6. $A=\begin{pmatrix}1 & a & a\\ a & 1 & a\\ a & a & 1\end{pmatrix}$ ① 正定 ② Jacobi 收敛

① i 所有主子矩阵行列式为正

ii 所有特征值为正

iii 可 Cholesky 分解

② i 谱半径小于 1

→ 特征值绝对值的最大值

ii 严格对角占优

→ $|A_{ii}| > \sum |A_{ij}|$

数值分析第四章)

4.2 乘幂法
4.7 Jacobi法
> 特征值，特征向量

2.2 GAUSS消去法
2.3 GAUSS 主元素法 (列主元 全主元)
2.4 GAUSS-JORDAN消去法
2.5 LU分解
- Doolittle分解
- Crout分解
- 列主元三角分解

2.6 LDU分解
- Cholesky分解
- 平方根法
- 改进平方根法
- Gauss全主元法行列式

2.7 追赶法
2.8 范数 谱半径 条件数

3.2 Jacobi迭代
Gauss Seidel迭代
SOR法
3.3 迭代矩阵判收敛
当数矩阵判收敛
*3.4 共轭梯度

4.1 乘幂法/改进
反幂法
4.2 Jacobi方法
*4.3 QR法
Householder变换
LR分解

```python
#2.2高斯消去法
def gauss_elimination(A, b):
    n = len(b)
    # 将A扩展成增广矩阵
    for i in range(n):
        A[i].append(b[i])

    # 前向消去过程
    for i in range(n):
        for k in range(i+1, n):
            factor = A[k][i] / A[i][i]
            for j in range(i, n+1):
                A[k][j] -= factor * A[i][j]

    # 回代过程
    x = [0 for i in range(n)]
    for i in range(n-1, -1, -1):
        x[i] = A[i][n] / A[i][i]
        for k in range(i-1, -1, -1):
            A[k][n] -= A[k][i] * x[i]
    return x

```

```python
#2.3列主元消去法
def pivot_gauss_elimination(A, b):
    n = len(b)
    # 将A扩展成增广矩阵
    for i in range(n):
        A[i].append(b[i])

    for i in range(n):
        # 列主元选择
        max_row = max(range(i, n), key=lambda k: abs(A[k][i]))
        A[i], A[max_row] = A[max_row], A[i]

        for k in range(i+1, n):
            factor = A[k][i] / A[i][i]
            for j in range(i, n+1):
                A[k][j] -= factor * A[i][j]

    x = [0 for i in range(n)]
    for i in range(n-1, -1, -1):
        x[i] = A[i][n] / A[i][i]
        for k in range(i-1, -1, -1):
            A[k][n] -= A[k][i] * x[i]
    return x

```

```python
#2.4GAUSS-JORDAN消去法
def gauss_jordan_elimination(A, b):
    n = len(b)
    # 将A扩展成增广矩阵
    for i in range(n):
        A[i].append(b[i])

    for i in range(n):
        # 归一化主对角线上的元素
        pivot = A[i][i]
        for j in range(i, n+1):
            A[i][j] /= pivot

        for k in range(n):
            if k != i:
                factor = A[k][i]
                for j in range(i, n+1):
                    A[k][j] -= factor * A[i][j]

    x = [A[i][n] for i in range(n)]
    return x

```

```python
#2.5LU分解法
def lu_decomposition(A):
    n = len(A)
    L = [[0.0] * n for i in range(n)]
    U = [[0.0] * n for i in range(n)]

    for i in range(n):
        L[i][i] = 1.0
        for j in range(i, n):
            sum_U = sum(L[i][k] * U[k][j] for k in range(i))
            U[i][j] = A[i][j] - sum_U

        for j in range(i+1, n):
            sum_L = sum(L[j][k] * U[k][i] for k in range(i))
            L[j][i] = (A[j][i] - sum_L) / U[i][i]

    return L, U

```

```python
#2.6平方根分解法
def cholesky_decomposition(A):
    n = len(A)
    L = [[0.0] * n for i in range(n)]

    for i in range(n):
        for j in range(i+1):
            sum_L = sum(L[i][k] * L[j][k] for k in range(j))
            if i == j:
                L[i][j] = (A[i][i] - sum_L) ** 0.5
            else:
                L[i][j] = (A[i][j] - sum_L) / L[j][j]

    return L

```

```
#2.7LDLT分解法
def ldlt_decomposition(A):
    n = len(A)
    L = [[0.0] * n for i in range(n)]
    D = [0.0] * n

    for i in range(n):
        for j in range(i):
            sum_LDLT = sum(L[i][k] * L[j][k] * D[k] for k in range(j))
            L[i][j] = (A[i][j] - sum_LDLT) / D[j]

        sum_D = sum(L[i][k] * L[i][k] * D[k] for k in range(i))
        D[i] = A[i][i] - sum_D
        L[i][i] = 1.0

    return L, D

```

```python
#2.8追赶法
def tridiagonal_matrix_algorithm(a, b, c, d):
    n = len(d)
    c_prime = [0] * n
    d_prime = [0] * n

    c_prime[0] = c[0] / b[0]
    d_prime[0] = d[0] / b[0]

    for i in range(1, n):
        denominator = b[i] - a[i] * c_prime[i-1]
        c_prime[i] = c[i] / denominator
        d_prime[i] = (d[i] - a[i] * d_prime[i-1]) / denominator

    x = [0] * n
    x[-1] = d_prime[-1]

    for i in range(n-2, -1, -1):
        x[i] = d_prime[i] - c_prime[i] * x[i+1]

    return x

```

```python
#3.2JACOBI迭代法
def jacobi_method(A, b, tolerance=1e-10, max_iterations=1000):
    import numpy as np
    n = len(A)
    x = np.zeros_like(b, dtype=np.double)
    x_new = np.zeros_like(x, dtype=np.double)

    for iteration in range(max_iterations):
        for i in range(n):
            s1 = np.dot(A[i, :i], x[:i])
            s2 = np.dot(A[i, i + 1:], x[i + 1:])
            x_new[i] = (b[i] - s1 - s2) / A[i, i]

        if np.linalg.norm(x_new - x, ord=np.inf) < tolerance:
            break
        x = x_new.copy()

    return x

```

```python
#3.2GAUSS-SEIDEI迭代法
def gauss_seidel_method(A, b, tolerance=1e-10, max_iterations=1000):
    import numpy as np
    n = len(A)
    x = np.zeros_like(b, dtype=np.double)

    for iteration in range(max_iterations):
        x_new = np.copy(x)
        for i in range(n):
            s1 = np.dot(A[i, :i], x_new[:i])
            s2 = np.dot(A[i, i + 1:], x[i + 1:])
            x_new[i] = (b[i] - s1 - s2) / A[i, i]

        if np.linalg.norm(x_new - x, ord=np.inf) < tolerance:
            break
        x = x_new

    return x

```

```python
#4.2乘幂法
def power_method(A, tol=1e-10, max_iter=1000):
    n = len(A)
    b_k = [1.0 for _ in range(n)]

    for _ in range(max_iter):
        # 计算A * b_k
        b_k1 = [sum(A[i][j] * b_k[j] for j in range(n)) for i in range(n)]

        # 计算新的b_k1的模长
        b_k1_norm = sum(x**2 for x in b_k1) ** 0.5

        # 归一化b_k1
        b_k = [x / b_k1_norm for x in b_k1]

        # 检查收敛性
        if sum((b_k1[i] / b_k1_norm - b_k[i])**2 for i in range(n)) < tol:
            break

    # 计算特征值
    eigenvalue = sum(A[i][j] * b_k[j] for i in range(n) for j in range(n)) / sum(b_k)

    return eigenvalue, b_k

# 示例矩阵
A = [
    [4, 1],
    [2, 3]
]
eigenvalue, eigenvector = power_method(A)
print("最大特征值:", eigenvalue)
print("对应特征向量:", eigenvector)
```

```python
#4.7JACOBI法
def jacobi_method(A, tol=1e-10, max_iter=1000):
    import math

    def max_off_diag(A):
        n = len(A)
        max_val = 0
        k = 0
        l = 0
        for i in range(n):
            for j in range(i + 1, n):
                if abs(A[i][j]) > max_val:
                    max_val = abs(A[i][j])
                    k = i
                    l = j
        return max_val, k, l

    def rotate(A, P, k, l):
        n = len(A)
        if A[k][k] != A[l][l]:
            tau = (A[l][l] - A[k][k]) / (2 * A[k][l])
            t = math.copysign(1 / (abs(tau) + math.sqrt(1 + tau * tau)), tau)
        else:
            t = 1 if A[k][l] > 0 else -1
        c = 1 / math.sqrt(1 + t * t)
        s = t * c

        for i in range(n):
            if i != k and i != l:
                A_ik = A[i][k]
                A_il = A[i][l]
                A[i][k] = c * A_ik - s * A_il
                A[k][i] = A[i][k]
                A[i][l] = c * A_il + s * A_ik
                A[l][i] = A[i][l]
        A_kk = A[k][k]
        A_ll = A[l][l]
        A_kl = A[k][l]
        A[k][k] = c * c * A_kk - 2 * c * s * A_kl + s * s * A_ll
        A[l][l] = s * s * A_kk + 2 * c * s * A_kl + c * c * A_ll
        A[k][l] = 0
        A[l][k] = 0

        for i in range(n):
            P_ik = P[i][k]
            P_il = P[i][l]
            P[i][k] = c * P_ik - s * P_il
            P[i][l] = c * P_il + s * P_ik

    n = len(A)
    P = [[1 if i == j else 0 for j in range(n)] for i in range(n)]
    for i in range(max_iter):
        max_val, k, l = max_off_diag(A)
        if max_val < tol:
            break
        rotate(A, P, k, l)

    eigenvalues = [A[i][i] for i in range(n)]
    eigenvectors = [P[i] for i in range(n)]
    return eigenvalues, eigenvectors

# 示例矩阵
A = [
    [4, -2, 2],
    [-2, 4, -2],
    [2, -2, 4]
]
eigenvalues, eigenvectors = jacobi_method(A)
print("特征值:", eigenvalues)
print("特征向量:", eigenvectors)

```