# 计算物理第一次作业

刘茁
1500011438

# 1. 数据误差的避免

## (a)

- 近考虑舍入误差计算机对两个数的相加可以表示为

$$a \oplus b = (a+b)(1+\delta), |\delta| < \epsilon_M/2$$

- 记机器计算的平均值为 $\overline{x}'$

$$\overline{x}' = \frac{1}{N}(((x_1+x_2)(1+\delta_1)+x_3)(1+\delta_2)+...)$$

- 忽略高阶小量，有

$$\begin{aligned}\overline{x}' &= \frac{1}{N}(\Sigma_{i=1}^N x_i + (x_1+x_2)\delta_1 + (x_1+x_2+x_3)\delta_2 + ... + \Sigma_{i=1}^N x_i\delta_N) \\ &= \overline{x} + \frac{1}{N}((x_1+x_2)\delta_1 + (x_1+x_2+x_3)\delta_2 + ... + \Sigma_{i=1}^N x_i\delta_N)\end{aligned}$$

- 利用$|\delta_i| < \epsilon_M/2$，相对误差可以表达为

$$\begin{aligned}\frac{\overline{x}'-\overline{x}}{\overline{x}} &\le \frac{\epsilon_M}{2\Sigma_{i=1}^N x_i}((N-1)(x_1+x_2)+(N-2)x_3+(N-3)x_4+...x_N) \\ &= \frac{\epsilon_M}{2\Sigma_{i=1}^N x_i}((N-1)\Sigma_{i=1}^N x_i - x_3 - 2x_4 - (N-2)x_N) \\ &\approx \frac{1}{2}(N+1)\epsilon_M\end{aligned}$$

- 注：以上仅考虑无符号数，否则无上限

## (b)

- 对第一种方法，乘法运算带来的舍入误差也可以表示为

$$a \text{✖} b = ab(1+\delta)$$

  - 先计算第一项，用与(a)中相同的方法，忽略一阶以上的高阶小量第一项最大的误差可以估计为

$$\Delta_{11} = (N+1)\epsilon_M \Sigma_{i=1}^N x_i^2$$

- 对第二项，舍入误差可估计为(a)的两倍

$$\Delta_{12} = (N+1)\epsilon_M(\Sigma_{i=1}^N x_i)^2/N^2 \approx \epsilon_M(\Sigma_{i=1}^N x_i)^2/N$$

- 考虑两项做差的舍入误差，第一钟方法的舍入误差可以估计为

$$\Delta_1 \approx \epsilon_M((\Sigma_{i=1}^N x_i)^2 + (\Sigma_{i=1}^N x_i)^2/N + S^2)$$

- 对第二种计算方法每一项计算中都含有$\overline{x}$ 的误差和减法运算的误差
  - 对于一次减法运算

$$x_i \bigtriangledown \overline{x} = (x_i - \overline{x}(1+\delta))(1+\delta_i) \approx (x_i - \overline{x})(1+\delta_i) - \delta\overline{x}$$

  - 因此

$$\begin{gathered}(x_i-\overline{x})(x_i-\overline{x}) = (x_i-\overline{x})(1+\delta_i)-\delta\overline{x})(x_i-\overline{x})(1+\delta_i)-\delta\overline{x})(1+\epsilon_i)\\ =((x_i-\overline{x})^2(1+2\delta_i)-2\delta\overline{x}(x_i-\overline{x}))(1+\epsilon_i)\\ =(x_i-\overline{x})^2(1+\epsilon_i+2\delta_i)-\delta\overline{x}(x_i-\overline{x}) \le (x_i-\overline{x})^2\frac{3}{2}\epsilon_M - \frac{1}{2}\epsilon_M\overline{x}(x_i-\overline{x})\end{gathered}$$

  - 求和的运算与(a)相同，$\overline{x}$的效应被抵消了

$$\Delta_2 \approx \frac{3}{2}(N+1)\epsilon_M S^2$$

- 可以看出来，第一种方法的误差上限可以非常大，因此第二种方法好。

## (c)

- 首先显然有

$$\int_0^1 \frac{1}{x+5}dx = ln(\frac{6}{5})$$

- 其次

$$I_k + 5I_{k-1} = \int_0^1 \frac{x^k+5x^{k-1}}{x+5} = \int_0^1 x^{k-1}dx = \frac{1}{k}$$

- 因此满足递推公式。考虑误差后，按照递推公式有

$$I_k + 5I_{k-1} + \epsilon_k + 5\epsilon_{k-1} = \frac{1}{k}$$

$$\therefore \epsilon_k = -5\epsilon_{k-1}$$

$$|\epsilon_k| = |5|^k|\epsilon|$$

- 不稳定性指数扩大，因此该算法是不稳定的

# 2. 矩阵的模与条件数

## (a)

- 只要将矩阵按第一列展开就可以发现，子矩阵的行列式不变，因此该矩阵的行列式与平凡矩阵(1)的行列式相同，即

$$det = 1$$

## (b)

- 用数学归纳法，首先

$$A_1^{-1} = (1)_{1\times 1}$$

- 假设第n阶矩阵的可以写为

$$A_n^{-1} = \begin{pmatrix} \alpha & \beta \\ \gamma & A_{n-1}^{-1} \end{pmatrix}$$

$$A_n A_n^{-1} = \begin{pmatrix} \alpha - \Sigma\gamma_i & \beta + (-1, ..., -1)A_{n-1}^{-1} \\ A_{n-1}\gamma & I_{n-1} \end{pmatrix} = I_n$$

- 容易判断$\gamma_i = 0$以及$\alpha = 1$，右上角的等式可以写为

$$\beta_i = \Sigma_j A_{n-1_{i,j}}^{-1}$$

- 于是最后的矩阵形式为

$$A_n^{-1} = \begin{pmatrix} 1 & 2^0 & 2^1 & ... & 2^{n-2} \\ 0 & 1 & 2^0 & ... & 2^{n-3} \\ 0 & 0 & 1 & ... & 2^{n-4} \\ ... & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & 1 \end{pmatrix}$$

## (c)

- 对于矢量，在$p \to \infty$时给出的是最大模，因此

$$||A||_\infty = sup_{x=\emptyset} \frac{max_i|\Sigma_{j=1}^n a_{ij}x_j|}{max_n|x_n|}$$

$$\frac{|\Sigma_{j=1}^n a_{ij}x_j|}{max_n|x_n|} \le \Sigma_{j=1}^n |a_{ij}| \frac{|x_j|}{max_n|x_n|} \le \Sigma_{j=1}^n |a_{ij}|$$

- 注意到当$x_j$与$a_{ij}$符号相同时，等号是可以取到的

$$\therefore ||A||_\infty = max_i \Sigma_{j=1}^n |a_{ij}|$$

## (d)

- 对幺正矩阵

$$UU^\dagger = I$$

$$||Ux||_2 = (x^\dagger U^\dagger U x)^{1/2} = ||x||_2$$

$$\therefore ||U||_2 = sup_{x=\emptyset} \frac{||Ux||_2}{||x||_2} = 1$$

同理

$$||U^\dagger||_2 = 1$$

- 对任意的A满足条件，同样地

$$||UA||_2 = sup_{x=\emptyset} \frac{||UAx||_2}{||x||_2} = sup_{x=\emptyset} \frac{||Ax||_2}{||x||_2} = ||A||_2$$

## (e)

$$||A||_\infty = n$$

$$||A^{-1}||_\infty = 2^{n-1}$$

$$K_\infty(A) = 2^{n-1} n$$

# 3.Hilbert矩阵

## (a)

$$\frac{\partial D}{\partial c_i} = \int_0^1 2(P_n(x) - f(x)) x^{i-1} dx = 0$$

$$\therefore \int_0^1 x^{i-1} P_n dx = \int_0^1 x^{i-1} f(x) dx$$

$$P_n = \Sigma_{j=1}^n c_j x^{j-1}$$

$$\Sigma_{j=1}^n \int_0^1 c_j x^{i+j-2} dx = \int_0^1 x^{i-1} f(x) dx$$

$$\Sigma_{j=1}^{n}\frac{1}{i+j-1}=\int_0^1 x^{i-1}f(x)dx$$

因此可以写成

$$H_nC=B,(H_n)_{ij}C_j=B_i$$

$$B_i=\int_0^1 x^{i-1}f(x)dx$$

$$(H_n)_{ij}=\frac{1}{i+j-1}$$

## (b)

$$\begin{aligned}c^TH_nc&=\int_0^1(\Sigma_{ij}c_ix^{i-1}c_jx^{j-1})dx\\&=\int_0^1(\Sigma_ic_ix^{i-1})^2dx\geq=0\end{aligned}$$

当且仅当对所有i，$c_i=0$时等号成立。从上述可以看出，$H_n$是对称正定的矩阵。

## (c)

- 取对数后

$$\begin{aligned}ln(det(H_n))&=4lnc_n-lnc_{2n}\\&=4ln\prod_{i=1}^{n-1}i!-ln\prod_{i=1}^{2n-1}i!\\&=4\Sigma_{i=1}^{n-1}(n-i)ln(i)-\Sigma_{i=1}^{2n-1}(2n-i)ln(i)\end{aligned}$$

- 编写python3程序 calculate.py，进行计算计算结果为

| n | $det(H_n)$ |
|---|---|
| 1 | 1.000000000000000 |
| 2 | 0.083333333333333 |
| 3 | 0.000462962962962 |
| 4 | 1.653439153439155e-07 |
| 5 | 3.749295132515107e-12 |
| 6 | 5.367299887358753e-18 |
| 7 | 4.835802623926094e-25 |

| | |
|---|---|
| 8 | 2.737050113791512e-33 |
| 9 | 9.720234311924803e-43 |
| 10 | 2.164179226431516e-53 |

## (d)

- 编写python3程序 Solve.py，取n=1-12的情况，求解结果详见附件

# 附件

## Caculate.py

```
import numpy as np

def det(n):
    sum1 = 0
    sum2 = 0
    for i in range(1,n):
        sum1 = sum1 + (n-i)*np.log(i)
    for i in range(1,2*n):
        sum2 = sum2 + (2*n-i)*np.log(i)

    sum = 4.0 * sum1 - sum2
    det = np.exp(sum)
    return det

file1 = open(r'HW_1_source_code/det.txt','w')
for i in range(1,11):
    de = det(i)
    string = "|{}|".format(i)
    s = str(de)
    file1.write(string)
    file1.write(s)
    file1.write("| \n")

file1.close()
```

## Solve.py

```
import numpy as np
import sys

def GEM(A,b):
    rows = A.shape[0]
    if rows != len(b):
        print("incompatible matrix A and b")
        exit()

    for i in range(0,rows-1):
        # select the pivot by the max
        firstcol=[]
        for ii in range(i,rows):
            firstcol.append(A[ii][i])

        pivot = firstcol.index(max(firstcol)) + i
        A[[i,pivot],:] = A[[pivot,i],:]
        b[i],b[pivot] = b[pivot],b[i]
        for j in range(i+1,rows):
            if A[j,i]!=0.0:
                fro = A[j,i] / A[i,i]
                A[j,i] = 0.0
                A[j,i+1:rows] = - fro * A[i,i+1:rows] + A[j,i+1:rows]
                b[j] = - fro * b[i] + b[j]

    x = np.zeros(rows)

    for k in range(rows-1,-1,-1):
        x[k] = (b[k] - np.dot(A[k,k+1:rows],x[k+1:rows]))/A[k,k]

    return x

def choleskey(A,b):
    rows,cols = A.shape
    if rows != len(b):
        print("incompatible matrix A and b")
        exit(0)
    if rows != cols:
```

```
        print("not correct matirx!")
        exit(0)

    H = np.zeros((rows,cols))

    H[0,0] = np.sqrt(A[0][0])
    for i in range(1,rows):
        for j in range (0,i):
            H[i,j] = (A[i,j] - np.dot(H[i,0:j],H[j,0:j]))/H[j,j]
        H[i,i] = np.sqrt(A[i,i] - np.dot(H[i,0:i],H[i,0:i]))

    x1 = np.zeros(rows)
    x2 = np.zeros(cols)

    for m in range(rows):
        x1[m] = (b[m] - np.dot(H[m,0:m],x1[0:m]))/H[m,m]

    for n in range(rows-1,-1,-1):
        x2[n] = (x1[n] - np.dot(H[n+1:rows,n],x2[n+1:rows]))/H[n,n]

    return x2

def create_hilbert(n):
    mat = []
    for i in range(1,n+1):
        line = []
        for j in range(1,n+1):
            h = 1.0 / (i+j-1)
            line.append(h)
        mat.append(line)
    hilbert = np.array(mat)
    return hilbert

def create_vector(n):
    return np.array([1 for i in range (1,n+1)],dtype=np.float)

file1 = open(r'HW_1_source_code/GEM.txt','w')
file2 = open(r'HW_1_source_code/Choleskey.txt','w')

for n in range (1,13):
    A1 = create_hilbert(n)
```

```
    b1 = create_vector(n)
    A2 = create_hilbert(n)
    b2 = create_vector(n)

    result_1 = GEM(A1,b1)
    result_2 = choleskey(A2,b2)
    string = "n = {} \n".format(n)

    file1.write(string)
    r1 = str(result_1) + "\n \n"
    file1.write(str(r1))

    file2.write(string)
    r2 = str(result_2) + "\n \n"
    file2.write(str(r2))

file1.close()
file2.close()
```

## 运行结果

Cholesky

n = 1
[1.]

n = 2
[-2. 6.]

n = 3
[ 3. -24. 30.]

n = 4
[ -4. 60. -180. 140.]

n = 5
[ 5. -120. 630. -1120. 630.]

n = 6
[-6.000e+00 2.100e+02 -1.680e+03 5.040e+03 -6.300e+03 2.772e+03]

n = 7
[ 7.00000001e+00 -3.36000000e+02 3.78000000e+03 -1.68000000e+04 3.46500000e+04

-3.32640000e+04 1.20120000e+04]

n = 8
[-7.99999935e+00 5.03999963e+02 -7.55999950e+03 4.61999972e+04 -1.38599992e+05 2.16215989e+05 -1.68167992e+05 5.14799977e+04]

n = 9
[ 8.99992463e+00 -7.19994699e+02 1.38599090e+04 -1.10879343e+05 4.50447568e+05 -1.00900300e+06 1.26125422e+06 -8.23676489e+05 2.18789129e+05]

n = 10
[-9.99647239e+00 9.89693166e+02 -2.37534315e+04 2.40180050e+05 -1.26097314e+06 3.78298940e+06 -6.72542008e+06 7.00002152e+06 -3.93755829e+06 9.23634288e+05]

n = 11
[ 1.09013685e+01 -1.30959927e+03 3.83385824e+04 -4.77430486e+05 3.13490714e+06 -1.20437288e+07 2.84479818e+07 -4.18155209e+07 3.72530997e+07 -1.84014238e+07 3.86519637e+06]

n = 12
[-1.06586186e+01 1.54967606e+03 -5.49242901e+04 8.32010641e+05 -6.70895715e+06 3.21428166e+07 -9.69534472e+07 1.88838275e+08 -2.36982407e+08 1.84952571e+08 -8.16237546e+07 1.55564209e+07]

GEM

n = 1
[1.]

n = 2
[-2. 6.]

n = 3
[ 3. -24. 30.]

n = 4
[ -4. 60. -180. 140.]

n = 5
[ 5. -120. 630. -1120. 630.]

n = 6
[-6.000e+00 2.100e+02 -1.680e+03 5.040e+03 -6.300e+03 2.772e+03]

n = 7
[ 7.00000004e+00 -3.36000002e+02 3.78000002e+03 -1.68000001e+04 3.46500001e+04

-3.32640001e+04 1.20120000e+04]

n = 8
[-7.99999982e+00 5.03999988e+02 -7.55999982e+03 4.61999989e+04 -1.38599997e+05 2.16215995e+05 -1.68167996e+05 5.14799990e+04]

n = 9
[ 8.99994489e+00 -7.19996133e+02 1.38599337e+04 -1.10879522e+05 4.50448234e+05 -1.00900437e+06 1.26125581e+06 -8.23677458e+05 2.18789370e+05]

n = 10
[-9.99807268e+00 9.89833697e+02 -2.37564615e+04 2.40207859e+05 -1.26110680e+06 3.78335912e+06 -6.72602984e+06 7.00061336e+06 -3.93787017e+06 9.23703099e+05]

n = 11
[ 1.09475741e+01 -1.31447626e+03 3.84658805e+04 -4.78860520e+05 3.14345886e+06 -1.20738884e+07 2.85138169e+07 -4.19054647e+07 3.73279469e+07 -1.84361073e+07 3.87205691e+06]

n = 12
[-1.03948050e+01 1.51178142e+03 -5.36181293e+04 8.12917880e+05 -6.56096500e+06 3.14624032e+07 -9.49849286e+07 1.85160064e+08 -2.32551511e+08 1.81630379e+08 -8.02138627e+07 1.52977616e+07]