**北京邮电大学软件学院**

**\_\_2019-2020\_\_学年第\_1\_学期实验报告**

**课程名称： \_\_\_\_\_\_\_\_\_\_\_数值分析\_\_\_\_\_\_\_\_\_\_\_\_\_**

**实验名称： \_\_\_\_\_\_实验一：Hilbert矩阵求解\_\_\_\_**

**实验完成人：**

**姓名：\_\_\_\_\_\_\_\_\_\_\_\_**隋小雨\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**学号：**\_\_\_\_\_\_\_\_\_\_2017211948\_\_\_\_\_\_\_\_\_\_\_

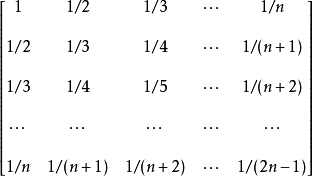
**成绩：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**指导教师：** \_\_\_\_\_\_\_\_\_\_\_\_\_\_漆涛\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**日 期： 2019 年 12 月 23 日**

# 问题描述

Hilbert矩阵是一种数学[变换矩阵](https://baike.baidu.com/item/%E5%8F%98%E6%8D%A2%E7%9F%A9%E9%98%B5/9035701" \t "/Users/xiaoyu/Documents\\x/_blank)，正定，且高度病态，其元素A（i,j）=1/(i+j-1)，i,j分别为其行标和列标。Hilbert矩阵病态程度和阶数相关。



Hn为n x n Hilbert矩阵，对于n=10, 20 , 30 求解



b的取值为使得方程的准确解为



求解过程采用列选主元高斯消去，完全选主元高斯消去，Householder变换三种方法，以python语言实现。每种算法实现过程中分别采用有理数，IEEE单精度浮点数，IEEE双精度浮点数，自定义精度小数四种精确度类型。对比不同算法，不同精度结果之间的区别并进行分析。

在此基础上对Hilbert矩阵进行行平衡变换并求其变换后的矩阵条件数。分析条件数的变化。

cond(A)=‖A‖·‖A^(-1)‖

方程求解的误差由一范数计算



# 算法思路分析

## 列选主元高斯消去

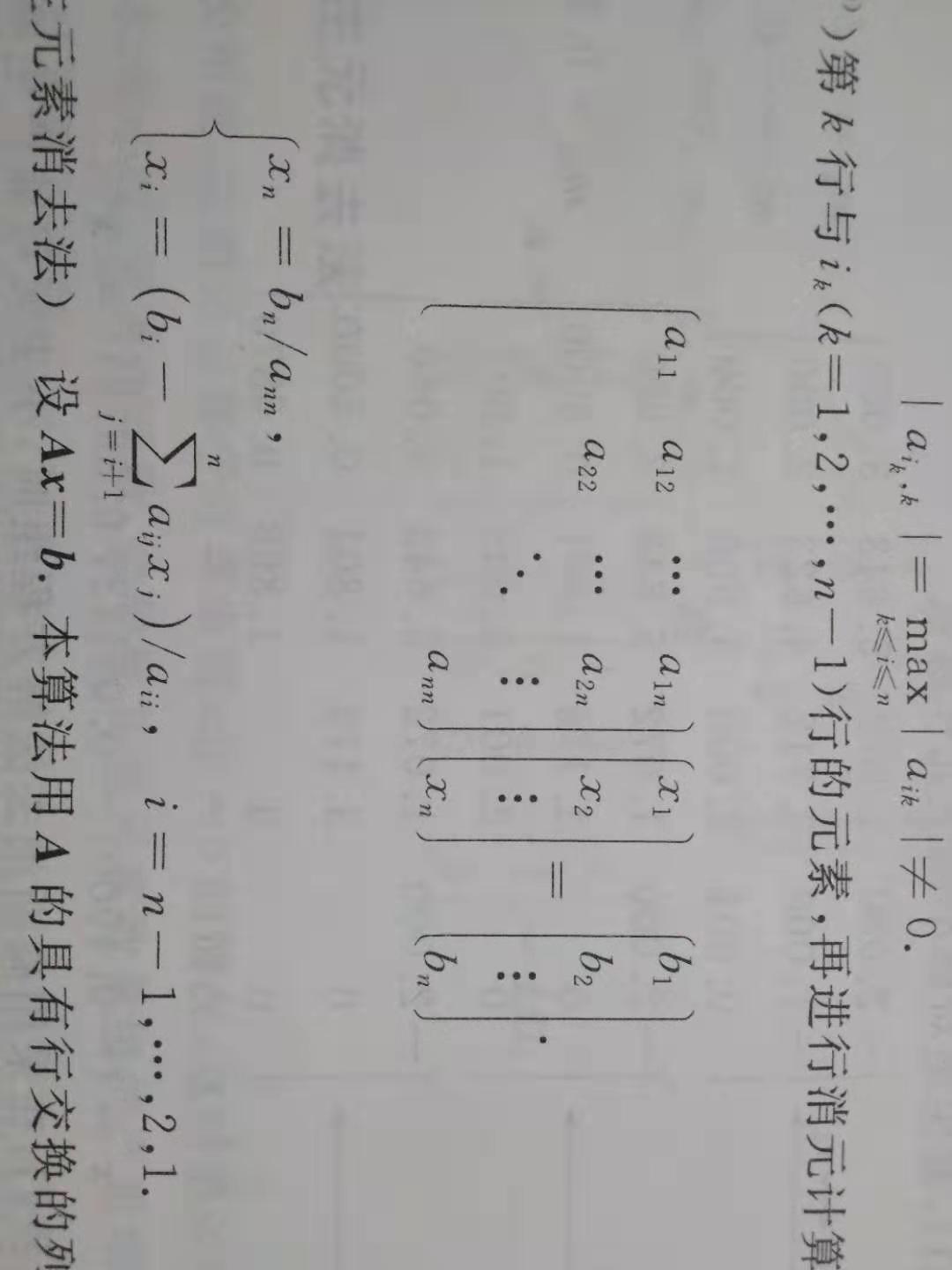
### 算法分析

列主元素消去法是为控制舍入误差而提出来的一种算法，列主元素消去法计算基本上能控制舍入误差的影响，其基本思想是：在进行第 k(k=1,2,...,n-1)步消元时，从第k列的 akk及其以下的各元素中选取绝对值最大的元素，然后通过行变换将它交换到主元素akk的位置上，再进行消元。

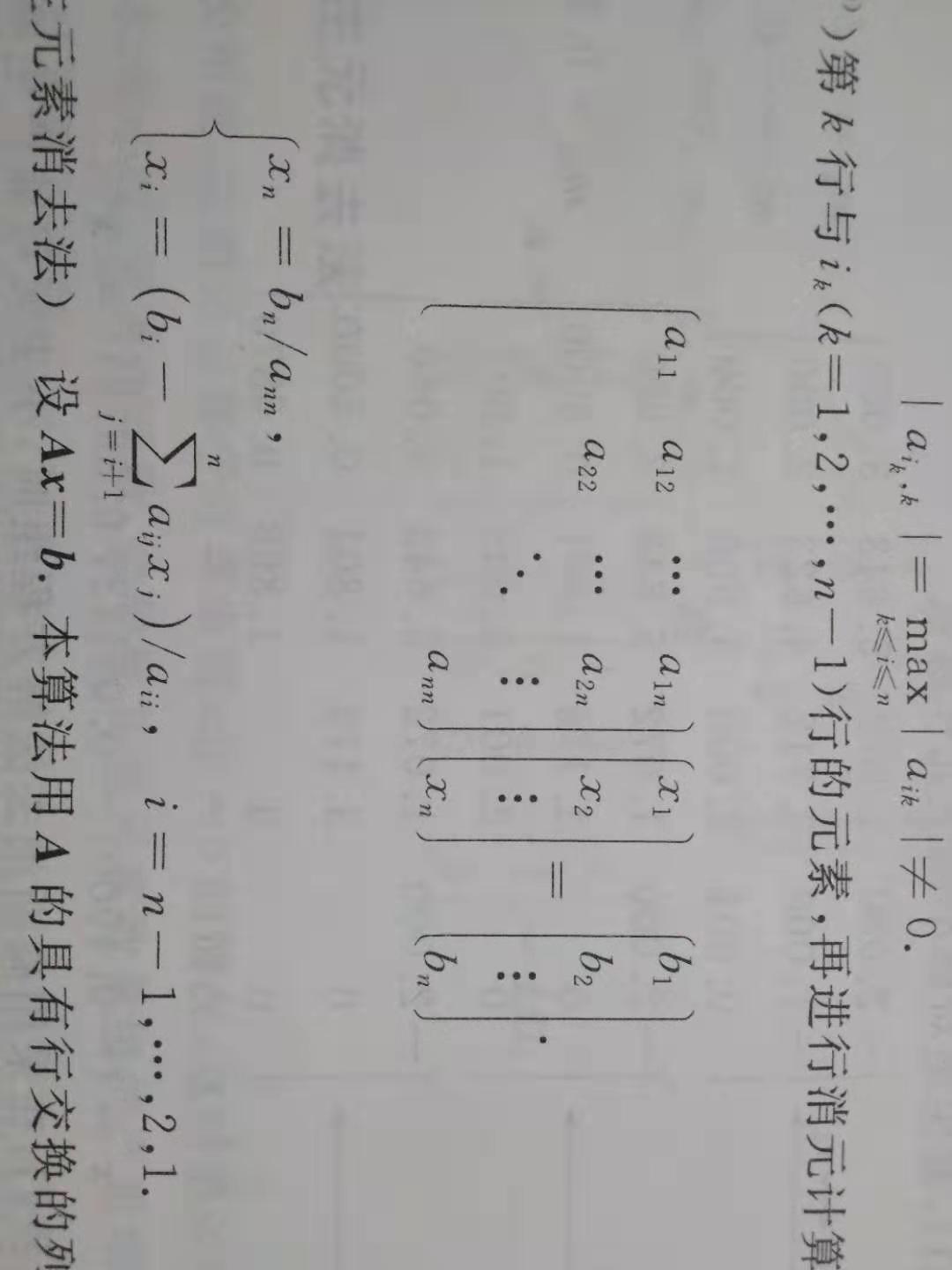
从矩阵第一列开始，寻找此列中绝对值最大的元素，将其所在行与矩阵第一行位置互换。之后对第2行到第n行，将每一行减去第一行乘以本行第一列元素与第一行第一列元素比值的乘积。第一次消去完成后矩阵第一列除第一行元素外全部为0。

第二次消元时，在第二列第二行到最后一行所有元素中选择绝对值最大的元素。将该元素所在行与第二行调换。之后对矩阵第3行到第n行进行消元。得到结果矩阵第一列第2行到最后一行以及矩阵第二列第3行到最后一行为0

以此类推，消去完成后，矩阵变为上三角矩阵。即此时



利用回带函数计算（x1,x2,...xn)T



### 算法实现

列选主元高斯消去

**for** i **in** range(n):  
 *#找到矩阵augment第i列中最大绝对值索引* t=findColMax(augment[:,i],i)  
 *#交换i行和最大值所在行* exchangeRow(augment,i,t,type)  
 *#消元* **for** k **in** range(i+1 , n):  
 m = augment[k, i] / augment[i, i]  
 augment[k, :] = augment[k, :] - m \* augment[i, :]

回带函数求解

*#增广矩阵回带函数***def** solution(a,type):  
 n=a.shape[0]  
 *#构造n x 1精度为type的矩阵* x=createTempMat(n,1,type)  
 *#求x最后一个元素* x[n-1]=a[n-1,n]/a[n-1,n-1]  
 *#求剩下的X* sum\_fra = 0  
 *#从倒数第二行到0行递减* **for** i **in** range(n-2,-1,-1):  
 **for** j **in** range (i+1,n):  
 sum\_fra += a[i, j] \* x[j]  
 b=a[i,n]-sum\_fra  
 x[i]=b/a[i,i]  
 sum\_fra=0  
 **return** x;

## **全选主元高斯消去**

### 算法分析

全选主元高斯消去与列选主元高斯消去非常类似，同样也是通矩阵的行列变换消元将原矩阵转化为上三角矩阵。与列选主元消去不同的是，全选主元消去过程中每次选择的不再是该列第i 行到第n行中绝对值最大的元素，而是矩阵第i 行到第n行，第i列到第n列中绝对值最大的元素。

第i 次消去时，取得全选主元最大值的行列索引值，将其所在行与第i 行互换，其所在列与第i列互换。由于行交换不会对解产生影响但是列交换会改变得到的解中（x1,x2,x3...xn)的顺序，所以需要记录列交换情况，在求解完成后对应恢复X中元素位置。

交换结束后主元位于（i，i）。之后进行与列选主元相同的消元步骤使得第i列的（i+1）行到第n行值为0。

消元完成后采用与列选主元消去采用同样的回带函数求解。得到的解再根据记录的交换顺序恢复元素顺序。

### **算法实现**

全选主元高斯消去

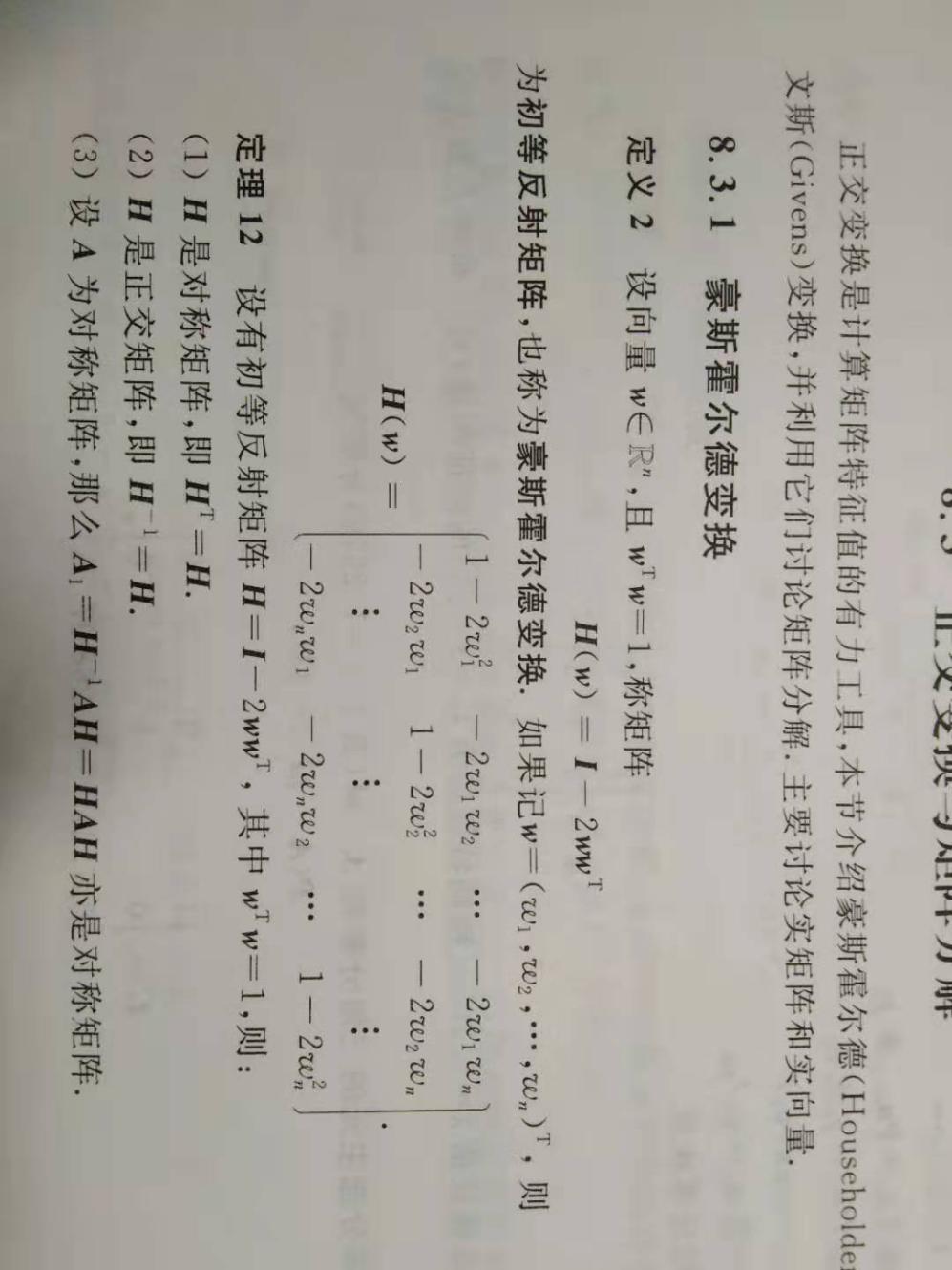
*#记录位置*pos=zeros(n)  
**for** i **in** range (n):  
 pos[i]=i  
**for** i **in** range(n):  
 *#取矩阵切片* temp=augment[i:n,i:n]  
 *#到切片绝对值最大值位置* x,y=findMatMax(temp)  
 *#行列交换* exchangeRow(augment,x+i,i,type)  
 exchangeCol(augment,y+i,i,type)  
 *#记录列交换位置* temp=pos[y+i]  
 pos[y+i]=pos[i]  
 pos[i]=temp  
 *#消元* **for** k **in** range(i+1,n):  
 m=augment[k,i]/augment[i,i]  
 augment[k,:]=augment[k,:]-m\*augment[i,:] *#由增广矩阵回带求X*sol=solution(augment,type)  
*#结果列顺序还原***for** l **in** range(n):  
 num=int(pos[l])  
 X[num]=sol[l]

## **Householder矩阵变换**

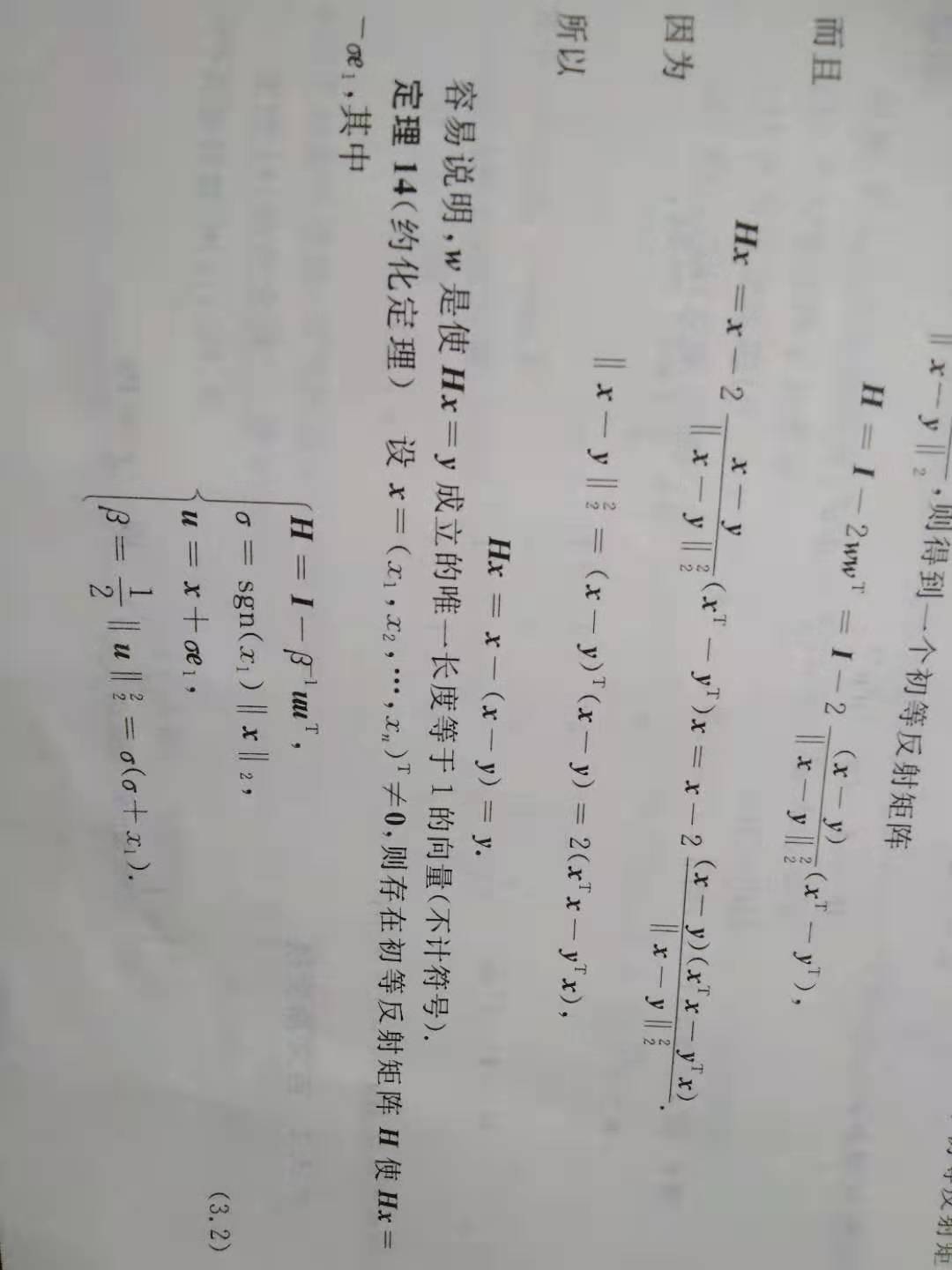
### 算法分析

豪斯霍尔德法(Householder method)，是[正交分解法](https://baike.baidu.com/item/%E6%AD%A3%E4%BA%A4%E5%88%86%E8%A7%A3%E6%B3%95/9487764" \t "/Users/xiaoyu/Documents\\x/_blank)的一种。它先在线性方程组Ax=b的两边施行n-1次豪斯霍尔德变换（[Householder变换](https://baike.baidu.com/item/Householder%E5%8F%98%E6%8D%A2" \t "/Users/xiaoyu/Documents\\x/_blank)）将其变为等价的上三角方程组Rx=b'，然后求解Rx=b'，即可得到原方程组的解。这一方法的运算量是[高斯消去法](https://baike.baidu.com/item/%E9%AB%98%E6%96%AF%E6%B6%88%E5%8E%BB%E6%B3%95/7939730" \t "/Users/xiaoyu/Documents\\x/_blank)的两倍，但其数值稳定性较好。

Householder矩阵定义如下：



Householder矩阵具有以下的性质



因此对于



可以通过对其进行Householder变换，对于第i列求得矩阵Hi(n-i，n-i)使得Hi左乘Hn可以使得Hn的第i列 (i+1) 行到第n行元素为0。

因此可以求得

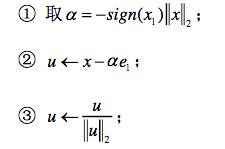
Hk ·Hk-1·。。。。H2·H1·Hilber(n) · x =Hk ·Hk-1·。。。。H2·H1·b

其中

Hk ·Hk-1·。。。。H2·H1·Hilber(n)

为上三角矩阵。特别地，对于





IMG_256 

由于Householder矩阵具有性质是n+r阶Householder矩阵，所以对于第

i列求得的（n-i）阶矩阵Hi可以利用这种方式扩充为n阶矩阵，从而进行矩阵乘法。

Householder变换后，x的左侧为上三角矩阵。因此可以使用回带函数求解得到x

### 算法实现

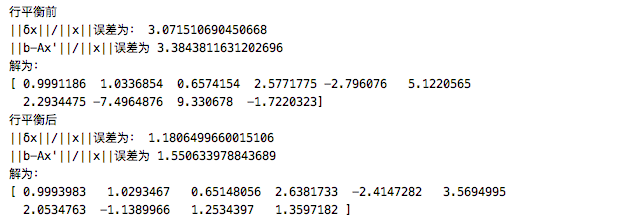
for i in range(n):  
 #求二范数  
 k=norm(a[i:n,i],type)  
 if(a[i,i]<0):  
 k=-k  
 #求u  
 u=createTempMat(n-i,1,type)  
 u=a[i:n,i].copy()  
 u[0]+=k  
 u=u.reshape(n-i,1)  
 #u的转制  
 uT=u.reshape(1,n-i)  
 #求U的二范数  
 u2=0  
 for j in range (len(u)):  
 u2+=u[j]\*u[j]  
 #计算beta  
 if(type=="decimal"):  
 beta=Decimal(0.5)\*u2  
 else:  
 if(type=="fraction"):  
 beta = Fraction(1, 2) \* u2  
 else:  
 beta = 0.5 \* u2  
 #计算Householder矩阵  
 H=dot(u,uT)  
 beta=beta[0]  
 if(type=="fraction"):  
 H=Fraction(1/beta)\*H  
 else:  
 if(type=="decimal"):  
 H=Decimal(1/beta)\*H  
 else:  
 H=(1/beta)\*H  
 H=IminusH(H)  
 #将H补足成可以相乘的大小  
 H=createH(H,n,type)  
 #Hilbert矩阵左乘Houseder矩阵  
 a=dot(H,a)  
 #b左乘Householder矩阵  
 b=dot(H,b)

# 实验结果

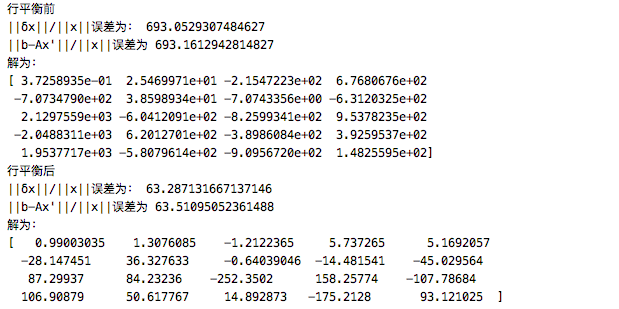
## 列选主元高斯消去

### 单精度类型

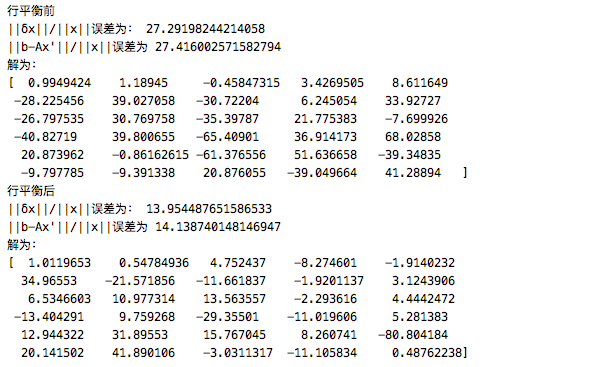
### N=10



### N=20

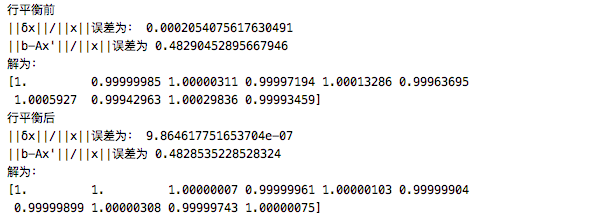


### N=30

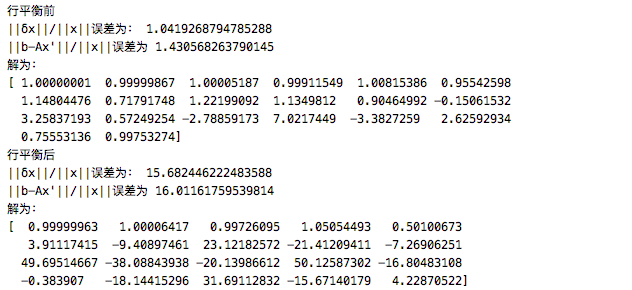


### 双精度类型

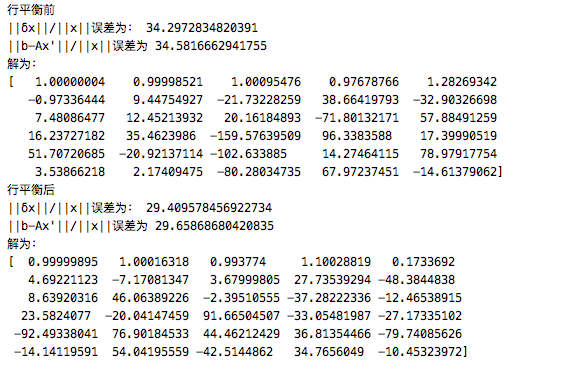
### N=10



### N=20

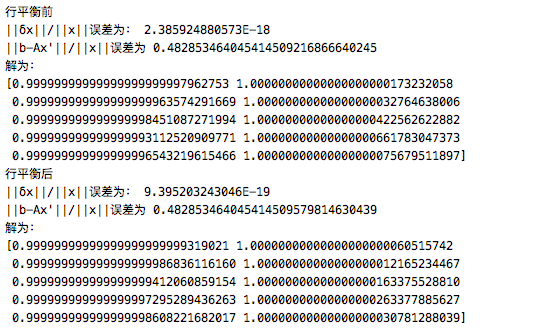


### N=30

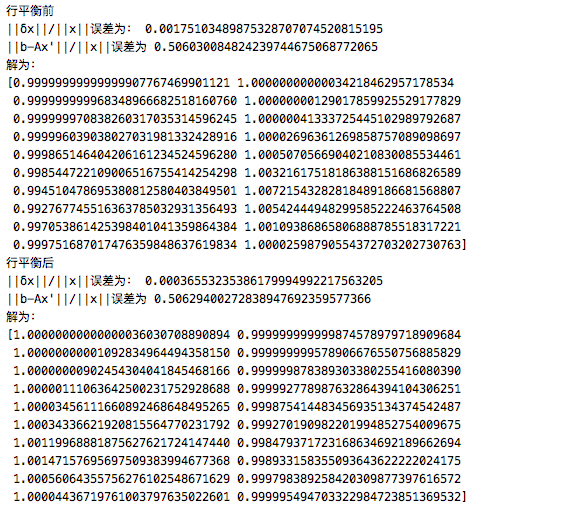


### 自定义精度Decimal30

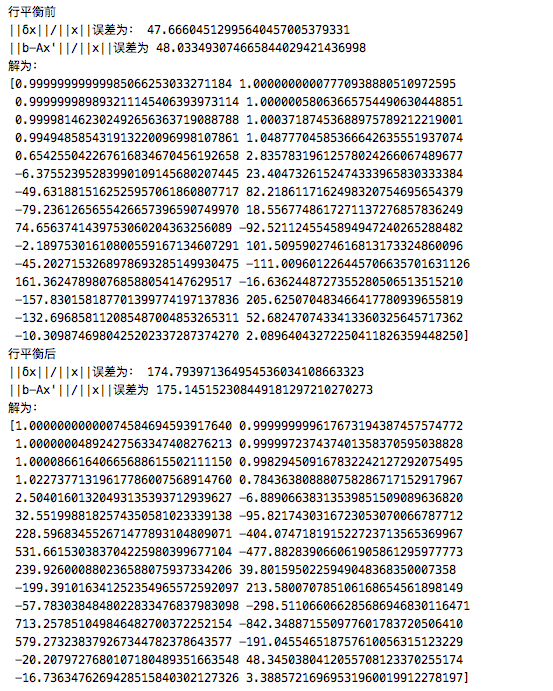
### N=10



### N=20

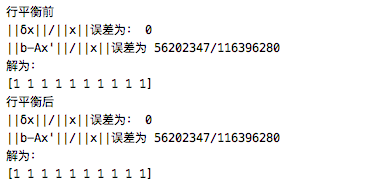


### N=30

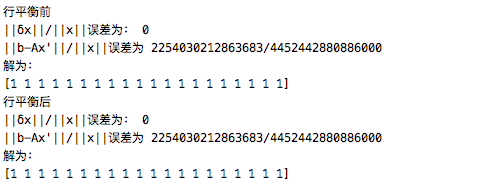


### 有理数

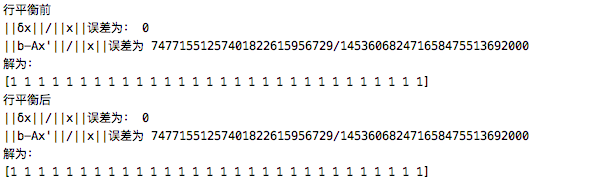
### N=10



### N=20



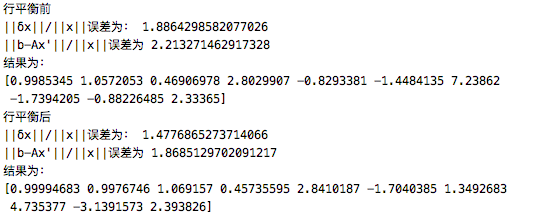
### N=30



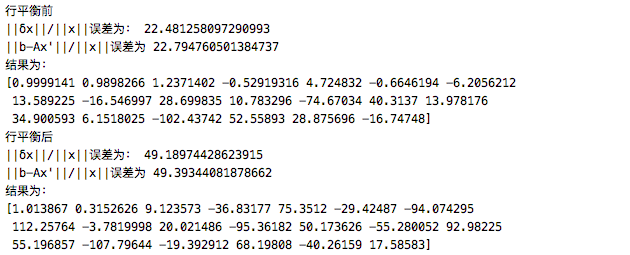
## 全选主元高斯消去

### 单精度类型

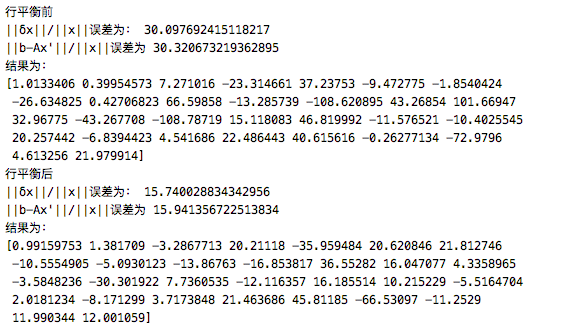
### N=10



### N=20

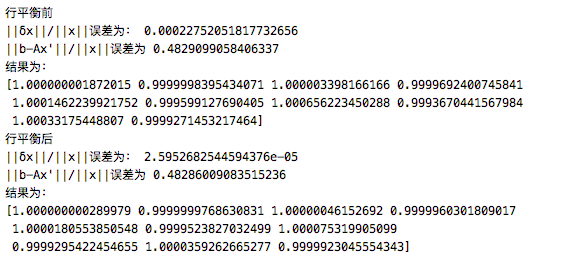


### N=30

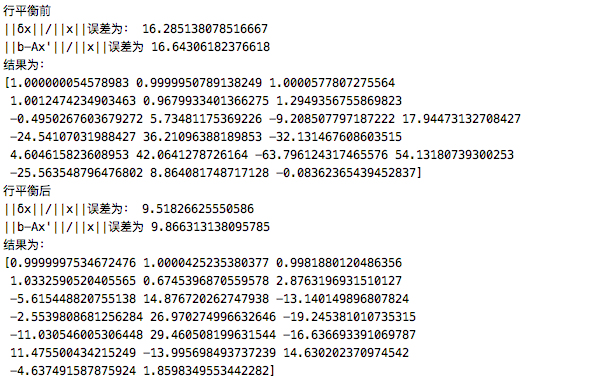


### 双精度类型

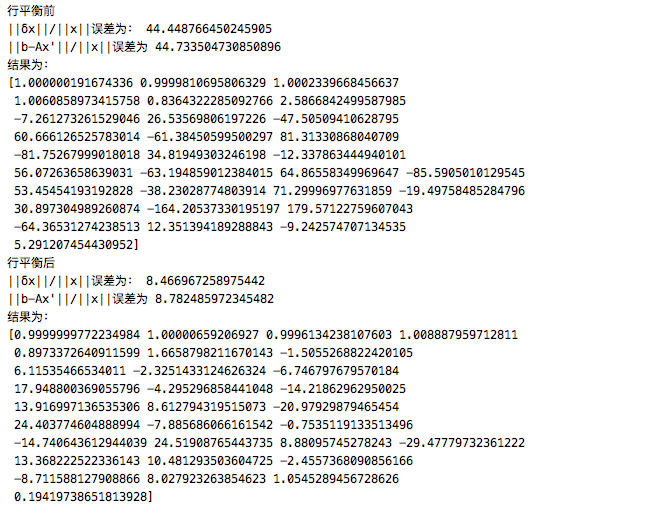
### N=10



### N=20

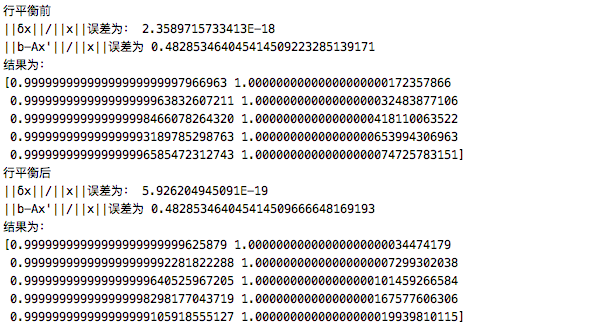


### N=30

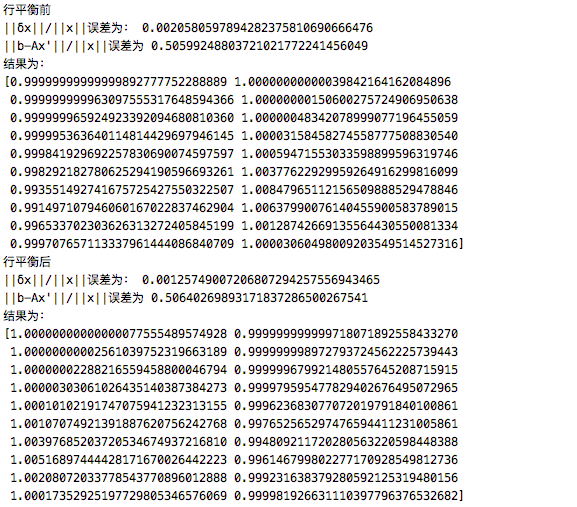


### 自定义精度Decimal30

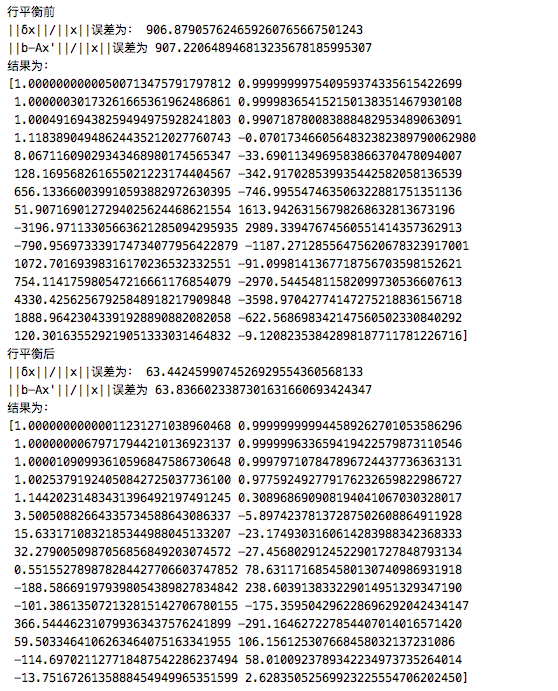
### N=10



### N=20

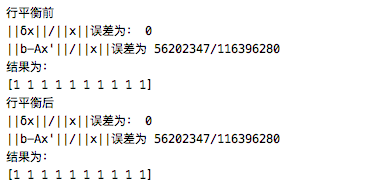


### N=30

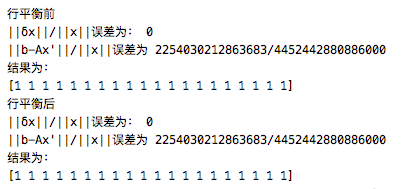


### 有理数

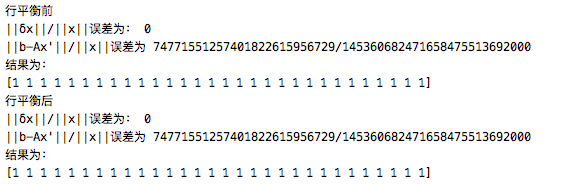
### N=10



### N=20



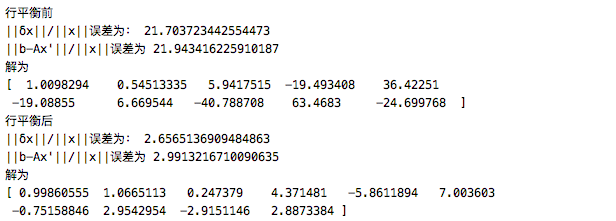
### N=30



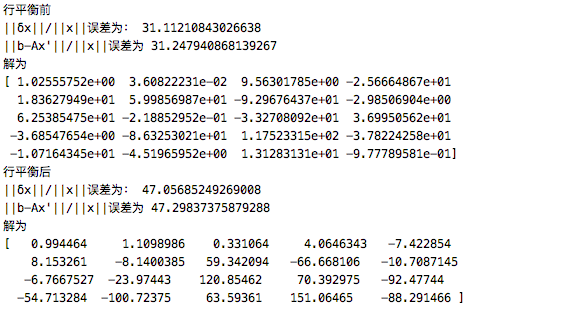
## Householder变换

### 单精度类型

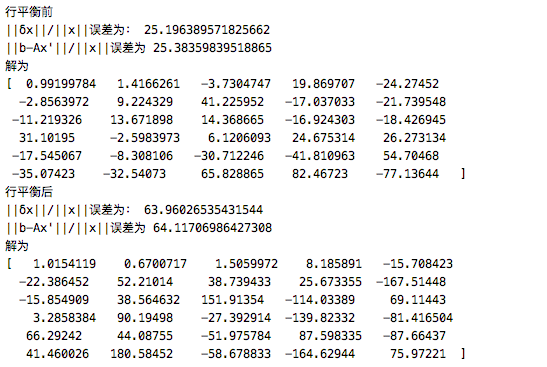
### N=10



### N=20

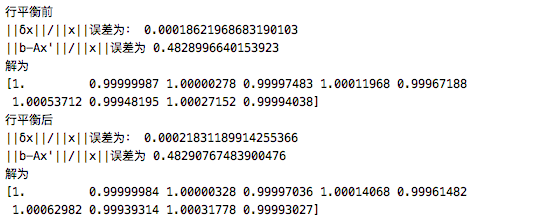


### N=30

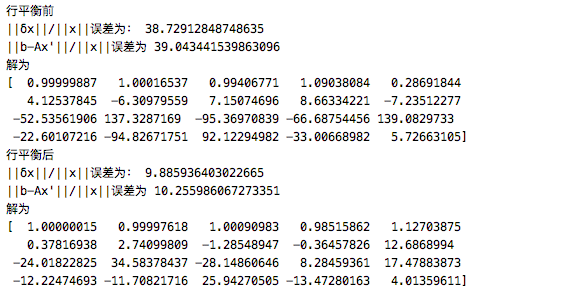


### 双精度类型

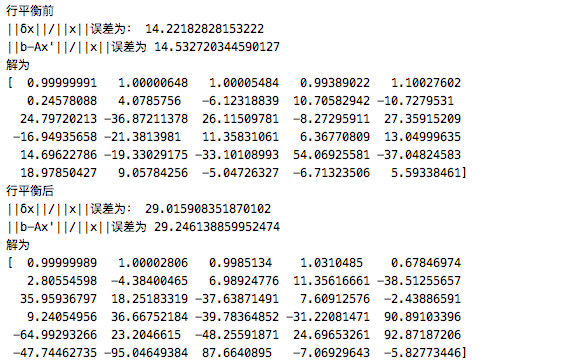
### N=10



### N=20

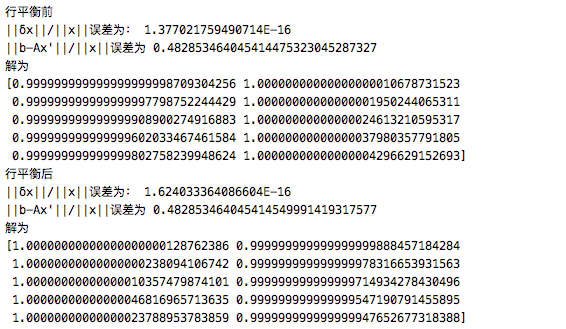


### N=30

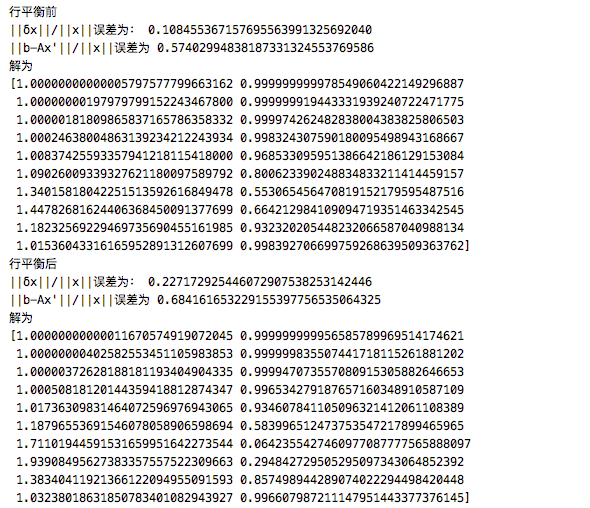


### 自定义精度Decimal30

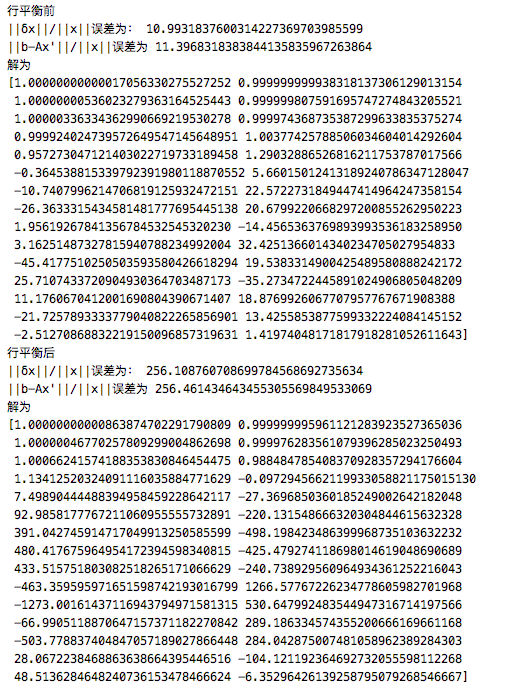
### N=10



### N=20

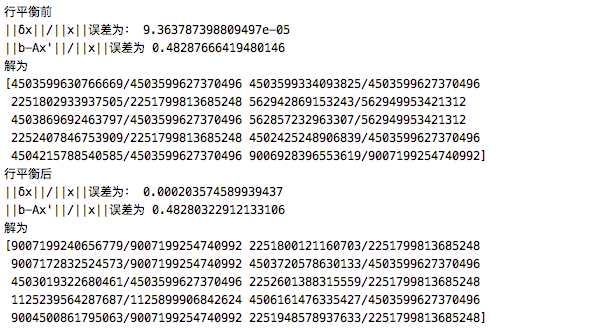


### N=30

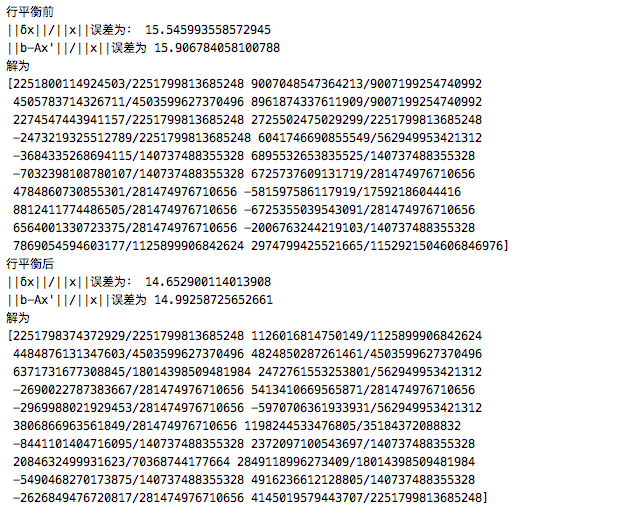


### 有理数

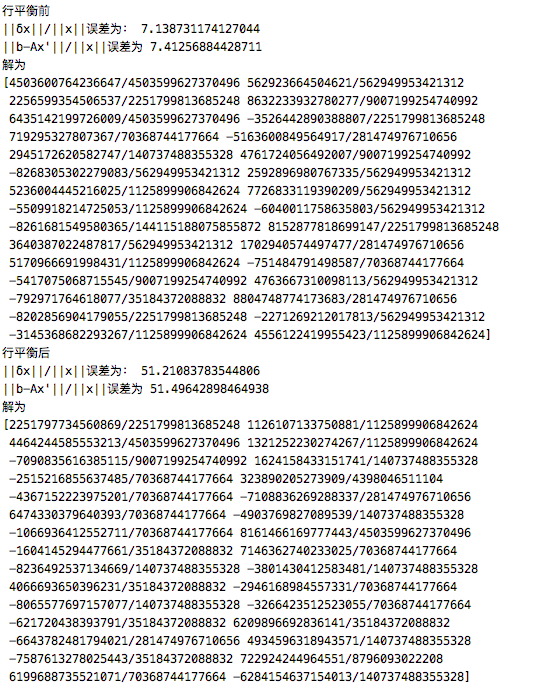
### N=10



### N=20



### N=30



# 实验结果分析

## 矩阵条件数分析

不同阶数的Hilbert矩阵行平衡前后条件数如下

|  |  |  |
| --- | --- | --- |
|  | 平衡前 | 平衡后 |
| N=10 | 971107180215.232 | 477102450126.83246 |
| N=20 | 1.4286302298176482e+17 | 8841509881088502.0 |
| N=30 | 1.8157597007240733e+17 | 2.2146828605199037e+18 |

当N=10，20，行平衡后矩阵条件数降低，印证了行平衡可以降低条件数。而当N=30，进行行平衡后矩阵条件数反而增大。这也说明行平衡对条件数的减小作用不是绝对的，存在随机性

## **误差分析**

以下是四种精度数据的实验结果统计表，误差采用公式||δx||/||x||进行计算，且数据均取到小数点后四位。

**单精度float32误差统计表**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 列选主元 | | 完全选主元 | | Householder | |
|  | 行平衡前 | 行平衡后 | 行平衡前 | 行平衡后 | 行平衡前 | 行平衡后 |
| N=10 | 3.0715 | 1.1806 | 1.8864 | 1.4776 | 21.7037 | 2.6565 |
| N=20 | 693.0529 | 63.2871 | 22.4812 | 49.1897 | 31.1121 | 47.0568 |
| N=30 | 27.2919 | 13.9544 | 30.0976 | 15.7400 | 25.1963 | 63.9602 |

**双精度float64误差统计表**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 列选主元 | | 完全选主元 | | Householder | |
|  | 行平衡前 | 行平衡后 | 行平衡前 | 行平衡后 | 行平衡前 | 行平衡后 |
| N=10 | 0.0002 | 9.864e-7 | 0.0002 | 2.5952e-5 | 0.0002 | 0.0002 |
| N=20 | 1.0419 | 15.6824 | 16.2851 | 9.5182 | 38.7291 | 9.8859 |
| N=30 | 34.2972 | 29.4096 | 44.4488 | 8.4670 | 14.2218 | 29.0159 |

**自定义精度Decimal(30)误差统计表**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 列选主元 | | 完全选主元 | | Householder | |
|  | 行平衡前 | 行平衡后 | 行平衡前 | 行平衡后 | 行平衡前 | 行平衡后 |
| N=10 | 2.3859e-18 | 9.3952e-19 | 2.3589e-18 | 5.9262e-19 | 1.3770e-16 | 1.6240e-16 |
| N=20 | 0.0017 | 0.0004 | 0.0021 | 0.0013 | 0.1085 | 0.2271 |
| N=30 | 44.6660 | 174.7939 | 960.8791 | 63.4425 | 10.9932 | 256.1088 |

**有理数Fraction误差统计表**

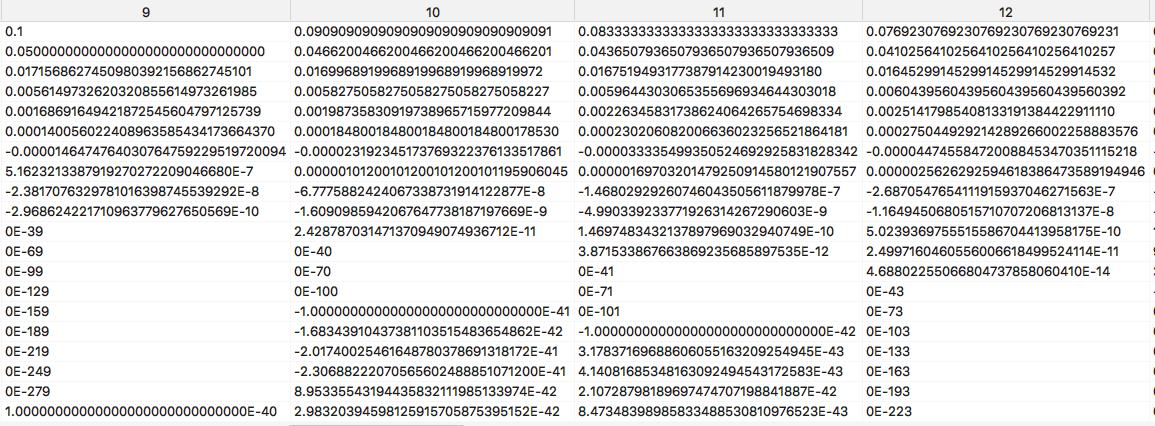
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 列选主元 | | 完全选主元 | | Householder | |
|  | 行平衡前 | 行平衡后 | 行平衡前 | 行平衡后 | 行平衡前 | 行平衡后 |
| N=10 | 0 | 0 | 0 | 0 | 9.3637e-05 | 0.0002 |
| N=20 | 0 | 0 | 0 | 0 | 15.5460 | 14.6529 |
| N=30 | 0 | 0 | 0 | 0 | 7.1387 | 51.2108 |

对比以上四张表作出下面规律总结与分析

### 数据类型对结果的影响

实验中采用的四种数据类型：单精度float32，双精度float64,自定义精度decimal30,有理数Fraction，精度逐渐增加。因此纵向对比四张表的每一列发现四个对于同一个阶数的矩阵，采用同一种算法，结果误差逐渐减小。数据类型精度越高，计算过程中因为精度不足而产生的舍入误差越小，自然结果误差会更小。但是对于decimal30精度N=30的误差相较单精度和双精度大很多。观察Decimal(30)的误差统计表，发现从N=20到N=30误差突然骤增。

查看自定义精度类型列选主元消去后矩阵的状态发现由于精度太高消去的时候被消元的部分不能被消去为0。从计算原理来看消去之后应该可以准确得到一个下三角矩阵，对于浮点类型，由于精度不够所以被消元部分舍入后为0，而自定义类型由于精度更高反而记录到了误差。虽然下三角部分的数据不对结果产生影响但是可以据此推断在消元过程中可能因为精度高但又不完全准确所以记录到了更多的误差。当N=30时累计的误差显现出来。



使用有理数数据类型，采用高斯消去方式求解不产生误差，但采用Householder方法会产生误差。原因是HIlbert矩阵中的元素都是分数，采用有理数类型存储可以不损失任何精度。并且高斯消去过程中对于数据的计算只涉及加减乘除，所以计算过程中矩阵中的数据也仍然全部是分数，用有理数类型不会产生舍入误差。而采用householder变换由于需要计算向量的二范数，开放后数据不再能保证是有理数所以Fraction数据类型会产生舍入误差，因此最终结果存在误差。

### 矩阵阶数对于结果的影响

总体来看随着n的增大，对于任意一种算法其误差都逐渐增大。这是因为计算过程中的误差来自于精度不足产生的舍入误差。随着矩阵阶数增加，计算步骤

增加，计算过程中出现舍入误差的步骤增多，所以总体的误差呈现增大的趋势。然而对于单精度数据类型采用列选主元消去和单精度行平衡后全选主元消去以及单精度行平衡前Householder变换求解，随着n的增大，结果误差先增大再减小。猜测这可能是因为误差的求法采用向量的一范式。当矩阵阶数很大时，||∆x||的增大非常有限，因为Hibert矩阵在行列数越大的位置的元素，数值越小，逐渐趋近于0。所以30阶时，对比20阶，||∆x||变化微弱但是分子n从20增大到30。所以计算得到的误差值变小。

### 行平衡产生的影响

观察实验数据，总体来看采用行平衡可以使结果误差减小。根据上一部分中给出的条件数计算结果可知进行行平衡可以使得Hilbert矩阵条件数降低。这意味着矩阵的变态程度得到降低，微小扰动对结果的影响减小，因而结果误差降低。同时当N=30，行平衡后矩阵的条件数增大，这一点在四种数据类型N=30采用Householder变换求解时都有体现。在这几个情况下行平衡后结果误差不降反增。

### 算法对结果的影响

通常高斯消去法有两个过程:消元过程和回代过程. 如果不选主元,消元过程第1步需要进行下面运算,a22-(a21/a11)\*a12。这是消元后所得新的第2个方程的x2的系数,如果a11较a21小的多,则a21/a11就很大,由于在计算机上编程计算或手算时,[舍入误差](https://www.baidu.com/s?wd=%E8%88%8D%E5%85%A5%E8%AF%AF%E5%B7%AE&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "/Users/xiaoyu/Documents\\x/_blank)难以避免,如a12=1/3,计算时需舍入为有限小数,比如保留8位[有效数字](https://www.baidu.com/s?wd=%E6%9C%89%E6%95%88%E6%95%B0%E5%AD%97&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "/Users/xiaoyu/Documents\\x/_blank),1/3用0.33333333代替,误差很小,但是当a21/a11很大,比如a21=10,a11=0.000001,则a21/a11=10000000,此时计算a22-(a21/a11)\*a12的值,由于a12有10^(-8)的误差,则(a21/a11)\*a12的误差却变为了10^(-1),误差被大大的放大了。这在[数值分析](https://www.baidu.com/s?wd=%E6%95%B0%E5%80%BC%E5%88%86%E6%9E%90&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "/Users/xiaoyu/Documents\\x/_blank)中称这种现象是算法不稳定,不稳定的算法计算出的结果就不可靠了。

列选主元消去通过寻找列中绝对值最大的数，调换行的位置使得其位于分母的位置。此时方程的a11较a21大,a21/a11的绝对值小于1,这样做的结果,在以后的计算中,误差不但不被放大,反而缩小。因此列选主元消去法是数值稳定的算法。

全选主元高斯消去与列选主元高斯消去非常相似，不同点是为了使消元时用来消元的行乘以的系数尽可能小，也就是使当前进行到的第i行的第1个不为0元素，H（i，i）尽可能大，搜索右下角的整个矩阵找到其中绝对值最大的元素，通过行交换和列交换使其位于矩阵（i，i）位置。因此对于同一个矩阵，采用列选主元消去时矩阵（i，i）位置的数值一定小于等于采用全主元消去时该位置的值。但是观察实验数据，完全选主元消去并不一定结果误差小于列选主元消去。

Householder的原理比高斯消去算法的数值稳定性更高，但是实验中其误差均大于高斯消去得到的结果。推断这是因为Householder变换求解在求二范数过程中开方操作计算过程中产生了较大的误差。

# 实验源代码

### ChooseCol.py 列选主元高斯消去

**from** fractions **import** Fraction  
**from** decimal **import** Decimal  
**from** numpy **import** \*  
**from** Tools **import**\*  
  
*#返回最大值的索引***def** findMax(a):  
 max=0  
 num=0  
 len=a.shape[1]  
 **for** i **in** range(a.shape[1]):  
 **if**(a[i]>max):  
 max=a[i]  
 num=i  
 **return** num  
  
*#找列向量v,第n行之后最大的元素,返回索引***def** findColMax(v,i):  
 max=0  
 num=0  
 **for** i **in** range(i,v.size):  
 **if**(abs(v[i])>max):  
 max=abs(v[i])  
 num=i  
 **return** num  
  
**def** colChoose(n,type,balance):  
 *#构造Hilbert矩阵* H = createHilbert(n)  
 *#构造X向量* X = createX(n)  
 *#计算b* b = dot(H, X)  
 *#构造增广矩阵* augment = Combine(H, b, n)  
 *#行平衡* **if**(balance==1):  
 augment = rowBalance(augment)  
 *#设置精度* **if**(type==**"decimal"**):  
 set\_printoptions(floatmode=**'unique'**, formatter={**'all'**: **lambda** x: str(Decimal(x))})  
 getcontext().prec = 30  
 **for** i **in** range(n):  
 **for** j **in** range(n+1):  
 augment[i, j] = Decimal(augment[i, j].numerator) / Decimal(augment[i, j].denominator)  
 **if**(type==**"float32"**):  
 augment=float32(augment)  
 **if**(type==**"float64"**):  
 augment=float64(augment)  
 **if**(type==**"fraction"**):  
 set\_printoptions(formatter={**'all'**: **lambda** x: str(Fraction(x))})  
  
 **for** i **in** range(n):  
 *#找到列中最大绝对值索引* t=findColMax(augment[:,i],i)  
 *#交换i行和最大值所在行* exchangeRow(augment,i,t,type)  
 *#消元* **for** k **in** range(i+1 , n):  
 m = augment[k, i] / augment[i, i]  
 augment[k, :] = augment[k, :] - m \* augment[i, :]  
  
 *#回带求解* res=solution(augment,type)  
 print(**"||δx||/||x||误差为："**,error(res))  
 print(**"||b-Ax'||/||x||误差为"**,balanceError(res,type))  
 print(**"解为："**)  
 print(res)  
 **return** res  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 type = **"fraction"  
 for** n **in** range(1, 4):  
 print(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* n ="**, n,  
 **" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**)  
 *# print("单精度")* print(**"行平衡前"**)  
 colChoose(10 \* n, type, 0)  
 print(**"行平衡后"**)  
 colChoose(10 \* n, type, 1)

### ChooseMat.py完全选主元高斯消去

**from** fractions **import** Fraction  
**from** decimal **import** Decimal  
**from** numpy **import** \*  
**from** Tools **import** \*  
*#找矩阵最大绝对值位置***def** findMatMax(m):  
 row=shape(m)[0]  
 col=shape(m)[1]  
 max=0  
 x=0  
 y=0  
 **for** i **in** range(row):  
 **for** j **in** range(col):  
 **if**(abs(m[i,j])>max):  
 max=abs(m[i,j])  
 x=i  
 y=j  
 **return** x ,y  
  
*#交换列i，t***def** exchangeCol(mat,i,t,type):  
 n=mat.shape[0]  
 temp = createTempMat(n, 1, type)  
 *#temp=zeros(n, dtype=fraction)* temp=mat[:,i].copy()  
 mat[:,i]=mat[:,t].copy()  
 mat[:,t]=temp.copy()  
  
*#完全选主元消去***def** completeChoose(n,type,balance):  
 *#创建Hilbert矩阵* H=createHilbert(n)  
 *#创建X* X=createX(n)  
 *#创建B* b=dot(H,X)  
 *#创建增广矩阵* augment=Combine(H,b,n)  
 *#行平衡* **if**(balance==1):  
 augment = rowBalance(augment)  
 *#设置精度和对应输出方式* **if**(type==**"float32"**):  
 augment=float32(augment)  
 **if**(type==**"float64"**):  
 augment=float64(augment)  
 **if**(type==**"decimal"**):  
 set\_printoptions(floatmode=**'unique'**, formatter={**'all'**: **lambda** x: str(Decimal(x))})  
 getcontext().prec = 30  
 **for** i **in** range(n):  
 **for** j **in** range(n + 1):  
 augment[i, j] = Decimal(augment[i, j].numerator) / Decimal(augment[i, j].denominator)  
 **if**(type==**"fraction"**):  
 set\_printoptions(formatter={**'all'**: **lambda** x: str(Fraction(x))})  
  
 *#记录位置* pos=zeros(n)  
 **for** i **in** range (n):  
 pos[i]=i  
 **for** i **in** range(n):  
 *#取矩阵切片* temp=augment[i:n,i:n]  
 *#到切片绝对值最大值位置* x,y=findMatMax(temp)  
 *#行列交换* exchangeRow(augment,x+i,i,type)  
 exchangeCol(augment,y+i,i,type)  
 *#记录列交换位置* temp=pos[y+i]  
 pos[y+i]=pos[i]  
 pos[i]=temp  
 *#消元* **for** k **in** range(i+1,n):  
 m=augment[k,i]/augment[i,i]  
 augment[k,:]=augment[k,:]-m\*augment[i,:]  
 *#return augment  
 #由增广矩阵回带求X* sol=solution(augment,type)  
 *#结果列顺序还原* **for** l **in** range(n):  
 num=int(pos[l])  
 X[num]=sol[l]  
 print(**"||δx||/||x||误差为："**, error(X))  
 print(**"||b-Ax'||/||x||误差为"**, balanceError(X, type))  
 print(**"结果为："**)  
 print(X)  
 **return** X  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 type = **"fraction"  
 for** n **in** range(1, 4):  
 print(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* n ="**, n,  
 **" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**)  
 *# print("单精度")* print(**"行平衡前"**)  
 completeChoose(10 \* n, type, 0)  
 print(**"行平衡后"**)  
 completeChoose(10 \* n, type, 1)

### Householder.py 豪斯霍尔德变换

**import** math  
**from** numpy **import** \*  
**from** Tools **import**\*  
  
**def** createH(h,n,type):  
 H=createTempMat(n,2,type)  
 k=h.shape[0]  
 H[n-k:n,n-k:n]=h.copy()  
 **for** i **in** range(0,n-k):  
 H[i,i]=1  
 **return** H  
  
*#回带求解***def** H\_solution(a, b):  
 n=size(b)  
 d = a[n - 1, n - 1]  
 *# B[n-1]/d的值为解x[n-1]* b[n - 1] = b[n - 1] / d  
 *# 对x[0..n-2]的值进行回代* **for** i **in** range(n - 2, -1, -1):  
 t = 0  
 **for** j **in** range(i + 1, n):  
 t = t + a[i, j] \* b[j]  
 b[i] = (b[i] - t) / a[i, i]  
 **return** b  
  
  
  
*#单位向量减去H***def** IminusH(h):  
 n=h.shape[0]  
 **for** i **in** range(n):  
 **for** j **in** range(n):  
 **if**(i==j):  
 h[i,j]=1-h[i,j]  
 **else**:  
 h[i,j]=-h[i,j]  
 **return** h;  
  
  
*#求向量范数***def** norm(v,type):  
 n=v.size  
 sum=0  
 **for** i **in** range(n):  
 sum+=v[i]\*v[i]  
 **if**(type==**"float32"**):  
 **return** sqrt(sum.astype(float32))  
 **if**(type==**"float64"**):  
 **return** sqrt(sum.astype(float64))  
 **if**(type==**"fraction"**):  
 **return** sum \*\*0.5  
 **if**(type==**"decimal"**):  
 sum = Decimal(sum)  
 **return** sum \*\* Decimal(0.5)  
  
*#行平衡***def** H\_rowBalance(a,b):  
 n=a.shape[0]  
 **for** i **in** range(n):  
 a[i,:]=a[i,:]/a[i,0]  
 b[i]=b[i]/a[i,0]  
 **return** a,b  
  
  
*#矩阵H左乘向量B***def** left\_multi(H, b, s):  
 n = size(b)  
 m = size(H[0])  
 res = createTempMat(n,1,s)  
 **for** i **in** range(n-m):  
 res[i]=b[i]  
 **for** i **in** range(m):  
 **for** j **in** range(m):  
 res[n-m+i] += H[i,j] \* b[n - m + j]  
 **return** res  
  
  
  
*#两个n\*n矩阵相乘***def** matrixMul(a, b,type):  
 n=a.shape[0]  
 res = createTempMat(n,2,type)  
 **for** i **in** range(len(a)):  
 **for** j **in** range(len(b[0])):  
 **for** k **in** range(len(b)):  
 res[i][j] += a[i][k] \* b[k][j]  
 **return** res  
  
  
  
**def** householder(n,type,balance):  
 *# 创建Hilbert矩阵* a = createHilbert(n)  
 *# 创建X* X = createX(n)  
 *#行平衡* **if**(balance==1):  
 a=rowBalance(a)  
 *# 创建B* b = transpose(dot(a, X))  
  
  
 *#设置精度* **if** (type == **"float32"**):  
 a=float32(a)  
 b=float32(b)  
 **if** (type == **"float64"**):  
 a=float64(a)  
 b=float64(b)  
 **if** (type == **"decimal"**):  
 set\_printoptions(floatmode=**'unique'**, formatter={**'all'**: **lambda** x: str(Decimal(x))})  
 getcontext().prec = 28  
 **for** i **in** range(n):  
 **for** j **in** range(n):  
 a[i, j] = Decimal(a[i, j].numerator) / Decimal(a[i, j].denominator)  
 b[i]=Decimal(b[i].numerator) / Decimal(b[i].denominator)  
 **if** (type == **"fraction"**):  
 set\_printoptions(formatter={**'all'**: **lambda** x: str(Fraction(x))})  
  
  
 *#对于矩阵每一列* **for** i **in** range(n):  
 *#求二范数* k=norm(a[i:n,i],type)  
 **if**(a[i,i]<0):  
 k=-k  
 *#求u* u=createTempMat(n-i,1,type)  
 **for** col **in** range(i,n):  
 u[col-i]=a[col,i]  
 u[0]+=k  
 u=u.reshape(n-i,1)  
 *#u的转制* uT=u.reshape(1,n-i)  
 *#求U的二范数* u2=0  
 **for** j **in** range (len(u)):  
 u2+=pow(u[j],2)  
 *#计算beta* **if**(type==**"decimal"**):  
 beta=Decimal(0.5)\*u2  
 **else**:  
 **if**(type==**"fraction"**):  
 beta = Fraction(1, 2) \* u2  
 **else**:  
 beta = 0.5 \* u2  
 *#计算Householder矩阵* H=matrixMul(u,uT,type)  
 beta=beta[0]  
 **if**(type==**"fraction"**):  
 H=Fraction(1/beta)\*H  
 **else**:  
 **if**(type==**"decimal"**):  
 H=Decimal(1/beta)\*H  
 **else**:  
 H=(1/beta)\*H  
 *#单位矩阵减去H* H=IminusH(H)  
 *#将H补足成可以相乘的大小* H=createH(H,n,type)  
 *#Hilbert矩阵左乘Houseder矩阵* a=matrixMul(H,a,type)  
 *#b左乘Householder矩阵* b=left\_multi(H,b,type)  
 res = H\_solution(a,b)  
 print(**"||δx||/||x||误差为："**, error(res))  
 print(**"||b-Ax'||/||x||误差为"**, balanceError(res, type))  
 print(**"解为"**)  
 print(res)  
 **return** a,b  
  
  
type=**"fraction"  
for** n **in** range(1,4):  
 print(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* n ="**,n,**" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**)  
 *#print("单精度")* print(**"行平衡前"**)  
 householder(10\*n,type,0)  
 print(**"行平衡后"**)  
 householder(10\*n,type,1)

### Tools.py 存放通用的工具函数

**from** decimal **import** \*  
  
  
**from** numpy **import**\*  
**from** fractions **import** Fraction  
fraction=Fraction  
dec=30  
  
*#构造增广矩阵***def** Combine(a,b,n):  
 c = zeros((n, n + 1), dtype=fraction)  
 **for** i **in** range(n):  
 **for** j **in** range(n):  
 c[i,j]=a[i,j]  
 **for** k **in** range(n):  
 c[k,n]=b[k]  
 **return** c  
  
*#构造hilbert矩阵***def** createHilbert(n):  
 a = zeros((n, n), dtype=fraction)  
 **for** i **in** range(n):  
 **for** j **in** range(n):  
 a[i,j]=Fraction(1,(i+j+1))  
 **return** a  
  
  
*#创建相应精度和纬度的矩阵***def** createTempMat(n,i,type):  
 **if**(i==1):  
 **if** (type == **"float32"**):  
 a = zeros(n, dtype=float32)  
 **if** (type == **"float64"**):  
 a = zeros(n, dtype=float64)  
 **if** (type == **"decimal"**):  
 getcontext().prec = dec  
 a = zeros(n, dtype=Decimal)  
 **if** (type == **"fraction"**):  
 a = zeros(n, dtype=fraction)  
 **return** a  
 **if**(i==2):  
 **if** (type == **"float32"**):  
 a = zeros((n, n), dtype=float32)  
 **if** (type == **"float64"**):  
 a = zeros((n, n), dtype=float64)  
 **if** (type == **"decimal"**):  
 getcontext().prec = dec  
 a = zeros((n, n), dtype=Decimal)  
 **if** (type == **"fraction"**):  
 a = zeros((n, n), dtype=fraction)  
 **return** a  
  
  
  
*#交换mat第t和i行，行元素个数为n***def** exchangeRow(mat,t,i,type):  
 n=mat.shape[0]  
 temp=createTempMat(n,1,type)  
 temp=mat[t,:].copy()  
 mat[t,:]=mat[i,:].copy()  
 mat[i,:]=temp.copy()  
  
  
*#增广矩阵回带函数***def** solution(a,type):  
 n=a.shape[0]  
 *#构造n x 1精度为type的矩阵* x=createTempMat(n,1,type)  
 *#求x最后一个元素* x[n-1]=a[n-1,n]/a[n-1,n-1]  
 *#求剩下的X* sum\_fra = 0  
 *#从倒数第二行到0行递减* **for** i **in** range(n-2,-1,-1):  
 **for** j **in** range (i+1,n):  
 sum\_fra += a[i, j] \* x[j]  
 b=a[i,n]-sum\_fra  
 x[i]=b/a[i,i]  
 sum\_fra=0  
 **return** x;  
  
*#创建fraction精度的X***def** createX(n):  
 *"""""  
 if (type == "float32"):  
 a = zeros(n, dtype=float32).T  
 if (type == "float64"):  
 a = zeros(n, dtype=float64).T  
 if (type == "decimal"):  
 getcontext().prec = 28  
 a = zeros(n, dtype=Decimal).T  
 if (type == "fraction"):  
 a = zeros(n, dtype=fraction).T  
 """* a = zeros(n, dtype=fraction).T  
 **for** i **in** range(n):  
 a[i]=1  
 **return** a  
  
  
*#行平衡***def** rowBalance(m):  
 n=m.shape[0]  
 **for** i **in** range(n):  
 m[i,:]=m[i,:]/m[i,0]  
 **return** m  
  
*#向量一范式,误差||δx||/||x||***def** error(s):  
 n=len(s)  
 sum=0  
 **for** i **in** range(n):  
 sum+=abs(s[i]-1)  
 **return** sum/n  
  
*#误差***def** balanceError(x,type):  
 n=len(x)  
 error=0  
 H=createHilbert(n)  
 b=dot(H,createX(n))  
 **if**(type==**"float32"**):  
 H=float32(H)  
 b=float32(b)  
 **if**(type==**"float64"**):  
 H=float32(H)  
 b=float32(b)  
 **if** (type == **"decimal"**):  
 getcontext().prec = 30  
 **for** i **in** range(n):  
 **for** j **in** range(n):  
 H[i, j] = Decimal(H[i, j].numerator) / Decimal(H[i, j].denominator)  
 b[i] = Decimal(b[i].numerator) / Decimal(b[i].denominator)  
 **for** i **in** range(n):  
 error += abs(b[i] - x[i])  
 error = error / n  
 **return** error  
  
*#矩阵一范式***def** paradigm(m):  
 n=m.shape[0]  
 max=Fraction(0)  
 sum=zeros(n,dtype=Fraction)  
 *#求列向量绝对值之和* **for** i **in** range(n):  
 **for** j **in** range(n):  
 sum[i]+=abs(m[j,i])  
 *#求和的最大值* **for** k **in** range(n):  
 **if**(sum[i]>max):  
 max=sum[i]  
 **return** max  
  
*#向量一范式***def** condition(H):  
 *#rH=linalg.pinv(H)* H=float64(H)  
 rH=linalg.inv(H)  
 a=paradigm(H)  
 b=paradigm(rH)  
 con=a\*b  
 print(con)  
 **return** con