运用雅克比和高斯赛德尔公式的求解方程组

姓名:陈扬,石晓晨

实验目的：

a. 比较两种方法的收敛速度；

b. 验证收敛条件的正确性。

雅克比算法

在数值线性代数中，**雅可比法**（Jacobi Method）是一种解对角元素几乎都是各行和各列的绝对值最大的值的线性方程组的算法。求解出每个对角元素并插入近似值。不断迭代直至收敛[1]。这个算法是雅可比矩阵的精简版。方法的名字来源于德国数学家卡尔·雅可比。

给定一个*n*×*n*的线性方程组

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

其中:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

A 可以分解成对角部分*D*和剩余部分*R*:

$$A = D + R \qquad \text{其中} \qquad D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}, \qquad R = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}$$

线性方程组可以重写为:

$$D\mathbf{x} = \mathbf{b} - R\mathbf{x}$$

雅可比法是一种迭代方法。在每一次迭代中，上一次算出的**x**被用在右侧，用来算出左侧的新的**x**。这个过程可以如下表示：

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - R\mathbf{x}^{(k)}).$$

对每个元素可以用以下公式：

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n.$$

注意计算 $x_i^{(k+1)}$需要$\mathbf{x}^{(k)}$中除了自己之外的每个元素。 不像高斯–赛德尔迭代，我们不能用 $x_i^{(k+1)}$覆盖$x_i^{(k)}$，因为在接下来的计算中还要用到这些值。这是雅可比和高斯–塞德尔方法最显著的差别，也是为什么前者可以用并行算法而后者不能的原因。最小需要的存储空间是两个长度为*n*的向量。

```python
# -*- coding: utf-8 -*-

#Jacobi迭代法 输入系数矩阵mx、值矩阵mr、迭代次数n、误差c(以list模拟矩阵 行优先)

def Jacobi(mx,mr,n=100,c=0.0001):
    if len(mx) == len(mr):  #若mx和mr长度相等则开始迭代 否则方程无解
        x = [] #迭代初值 初始化为单行全0矩阵
        for i in range(len(mr)):
            x.append([0])
        count = 0 #迭代次数计数
        while count < n:
            nx = [] #保存单次迭代后的值的集合
            for i in range(len(x)):
                nxi = mr[i][0]
                for j in range(len(mx[i])):
                    if j!=i:
                        nxi = nxi+(-mx[i][j])*x[j][0]
                nxi = nxi/mx[i][i]
                nx.append([nxi]) #迭代计算得到的下一个xi值
            lc = [] #存储两次迭代结果之间的误差的集合
            for i in range(len(x)):
                lc.append(abs(x[i][0]-nx[i][0]))
            if max(lc) < c:
                print(count)
                return nx #当误差满足要求时 返回计算结果
            x = nx
            count = count + 1
        return False #若达到设定的迭代结果仍不满足精度要求 则方程无解
    else:
        return False

#调用 Jacobi(mx,mr,n=100,c=0.001) 示例
mx = [[8,-3,2],[4,11,-1],[6,3,12]]

mr = [[20],[33],[36]]
print(Jacobi(mx,mr,100,0.000001))

```

```
迭代次数 15
[[2.9999999884363877], [1.999999749136996], [0.9999998711422083]]
```

高斯－赛德尔迭代（Gauss–Seidel method）

是数值线性代数中的一个迭代法，可用来求出线性方程组解的近似值。该方法以卡尔·弗里德里希·高斯和路德维希·赛德尔命名。同雅可比法一样，高斯－赛德尔迭代是基于矩阵分解原理。

算法[编辑]

对于一个含有n个未知量及n个等式的如下线性方程组

$$\begin{aligned} a_{11} \cdot x_1 + a_{12} \cdot x_2 + \ldots + a_{1n} \cdot x_n &= b_1, \\ a_{21} \cdot x_1 + a_{22} \cdot x_2 + \ldots + a_{2n} \cdot x_n &= b_2, \\ \vdots \qquad\qquad\qquad\qquad &= \vdots \\ a_{n1} \cdot x_1 + a_{n2} \cdot x_2 + \ldots + a_{nn} \cdot x_n &= b_n. \end{aligned}$$

为了求这个方程组的解$\vec{x}$，我们使用迭代法。k用来计量迭代步数。给定该方程组解的一个近似值$\vec{x}^{\,k} \in \mathbb{R}^n$。在求k+1步近似值时，我们利用第m个方程求解第m个未知量。在求解过程中，所有已解出的k+1步元素都被直接使用。这一点与雅可比法不同。对于每个元素可以使用如下公式

$$x_m^{k+1} = \frac{1}{a_{mm}} \left(b_m - \sum_{j=1}^{m-1} a_{mj} \cdot x_j^{k+1} - \sum_{j=m+1}^{n} a_{mj} \cdot x_j^k \right), \quad 1 \leq m \leq n.$$

重复上述的求解过程，可以得到一个线性方程组解的近似值数列：$\vec{x}^0, \vec{x}^1, \vec{x}^2, \ldots$。在该方法收敛的前提下，此数列收敛于$\vec{x}$.

为了保证该方法可以进行，主对角线元素a_{mm}需非零。

```
迭代次数：  15
[[2.9999999884363877], [1.999999749136996], [0.999999871142208]]
```

```python
def gaussSeidel(A, b, x, N, tol):
    maxIterations = 10000
    xprev = [0.0 for i in range(N)]
    for i in range(maxIterations):
        for j in range(N):
            xprev[j] = x[j]
        for j in range(N):
            summ = 0.0
            for k in range(N):
                if (k != j):
                    summ = summ + A[j][k] * x[k]
            x[j] = (b[j] - summ) / A[j][j]
        diff1norm = 0.0
        oldnorm = 0.0
        for j in range(N):
            diff1norm = diff1norm + abs(x[j] - xprev[j])
            oldnorm = oldnorm + abs(xprev[j])
```

```
        if oldnorm == 0.0:
            oldnorm = 1.0
        norm = diff1norm / oldnorm
        if (norm < tol) and i != 0:
            print("Sequence converges to [", end="")
            for j in range(N - 1):
                print(x[j], ",", end="")
            print(x[N - 1], "]. \n迭代次数: ", i + 1 )
            return
    print("Doesn't converge.")

guess = [0.0, 0.0 , 0.0 ]
mx = [[8,-3,2],[4,11,-1],[6,3,12]]
mr = [20,33,36]

gaussSeidel(mx,mr, guess,3, 0.000001)
```

```
Sequence converges to [2.9999996835259037 ,2.0000000530966062
,1.0000001449628966 ].
迭代次数:  8
```

结论:高斯-赛德尔迭代算法比雅克比迭代算法收敛速度更快