数值实验报告 I

实验名称	线性代数方程组的解法				实验时间	2021 年 10 月 16 日	
姓名	孙百乐	班级	本研 AI2001	学号	2007010218	成绩	

一、实验目的

1．理解掌握线性方程组求解和迭代思想。

2．代码实现三角分解法、高斯列主元，Jacobi 迭代，Gauss-Sedial 迭代。

二、实验内容

三角分解法：

三角分解法亦称因子分解法，由消元法演变而来的解线性方程组的一类方法。设方程组的矩阵形式为 Ax=b，三角分解法就是将系数矩阵 A 分解为一个下三角矩阵 L 和一个上三角矩阵 U 之积：A=LU，然后依次解两个三角形方程组 Ly=b 和 Ux=y，而得到原方程组的解。

高斯列主元：

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。

Jacobi 迭代法：

对于一般的线性方程组，从中分离出未知数 x1,x2,x3……，则原式可写为一个有迭代形式的矩阵乘积，若产生的雅可比迭代公式收敛于方程组的解，则任意定一个初始值，经过足够次数的迭代，都能得到预期精度的解。

Gauss-Sedial 迭代法：

在雅克比迭代法中，并没有对新算出的分量进行充分利用，一般来说，这些新算出计算的结果要比上一步计算的结果精确。对上式第二个方程组，第一行式子算出的 x 值立即投入第二行方程里，第二行式子的结果算出后投入第三行方程中，直到第 n 个方程。根据这种思路建立的迭代格式，就是高斯-赛戴尔迭代法。

三、程序代码

三角分解法解线性方程组

```python
import numpy as np
def LUFact(A):# LU Factorization, 分解为L,U矩阵
    n = len(A)
    L = np.eye(n)
    U = np.zeros(A.shape)
    U[0,:] = A[0,:]
    L[1:,0] = A[1:,0]/U[0,0]
    for r in range(1,n):
        lt = L[r,:r].reshape(r,1)
        U[r,r:] = A[r,r:] - np.sum(lt*U[:r,r:],axis=0)
#         print('\n#U%d:'%(r))
#         print(U)
        if r==n-1:
            continue
        ut = U[:r,r].reshape(r,1)
        L[r+1:,r] = (A[r+1:,r] - np.sum(ut*L[r+1:,:r].T,axis=0))/U[r,r]
#         print('\n#L%d:'%(r))
#         print(L)
    return L,U
```

```python
def solveLineq(L,U,b):#根据分解后的矩阵计算
    # Ly = b
    rows = len(b)
    y = np.zeros(rows)
    y[0] = b[0]/L[0,0]
    for k in range(1,rows):
        y[k] = (b[k] - np.sum(L[k,:k]*y[:k]))/L[k,k]
    # Ux = y (back substitution)
    x = np.zeros(rows)
    k = rows-1
    x[k] = y[k]/U[k,k]
    for k in range(rows-2,-1,-1):
        x[k] = (y[k] - np.sum(x[k+1:]*U[k,k+1:]))/U[k,k]
    return x

def SanJiao(A, b):#封装三角分解法
    """
    输入：A：numpy矩阵
          b：numpy矩阵，无需转置
    输出：x：numpy矩阵
    """
    L,U = LUFact(A)
    print("分解后的L矩阵：",L)
    print("分解后的U矩阵：",U)

    x = solveLineq(L,U,b)
    print("计算结果：",x)
```

高斯列主元

```python
import numpy as np

def getInput(A,b):
    matrix_a = np.mat(A, dtype=float)
    matrix_b = np.mat(b)
    # 答案：-2 0 1 1
    return matrix_a, matrix_b



# 交换
def swap(mat, num):
    print(num)
    print("调换前")
    print(mat)
    maxid = num
    for j in range(num, mat.shape[0]):
        if mat[j, num] > mat[num, num]:
            maxid = j
    if maxid is not num:
        mat[[maxid,num],:] = mat[[num,maxid],:]
    else:pass
    print("调换后")
    print(maxid)
    print(mat)
    return mat

```

```
def SequentialGauss(mat_a):
    for i in range(0, (mat_a.shape[0])-1):
        swap(mat_a, i)
        if mat_a[i, i] == 0:
            print("终断运算：")
            print(mat_a)
            break
        else:
            for j in range(i+1, mat_a.shape[0]):
                mat_a[j:j+1 , :] = mat_a[j:j+1,:] - \
                                                    (mat_a[j,i]/mat_a[i,i])*mat_a[i, :]
    return mat_a

# 回带过程
def revert(new_mat):
    # 创建矩阵存放答案 初始化为0
    x = np.mat(np.zeros(new_mat.shape[0], dtype=float))
    number = x.shape[1]-1
    # print(number)
    b = number+1
    x[0,number] = new_mat[number,b]/new_mat[number, number]
    for i in range(number-1,-1,-1):
        try:
            x[0, i] = (new_mat[i,b]-np.sum(np.multiply(new_mat[i,i+1:b],x[0,i+1:b])))/(new_mat[i,i])
        except:
            print("错误")
    print(x)
```

```
def GSLZY(A, b):
    A,b = getInput(A, b)
    # 合并两个矩阵
    print("原矩阵")
    print(np.hstack((A, b.T)))
    new_mat = SequentialGauss(np.hstack((A, b.T)))
    print("三角矩阵")
    print(new_mat)
    print("方程的解")
    revert(new_mat)
```

Jacobi迭代法

```
#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):
    mr = [[i] for i in mr]
    print("mr:",mr)
    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("满足误差要求的结果：")
                print(nx)
                print("迭代次数：",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
```

Gauss-Seidal迭代法

```python
import numpy as np

def G_S(a, b, x=np.array([0,0,0]), g=1e-6):   # a为系数矩阵  b增广的一列  x迭代初始值  g计算精度
    x = x.astype(float)  #设置x的精度，让x计算中能显示多位小数
    m, n = a.shape
    times = 0           #迭代次数
    if (m < n):
        print("There is a 解空间。")  # 保证方程个数大于未知数个数
    else:
        while True:
            for i in range(n):
                s1 = 0
                tempx = x.copy()    #记录上一次的迭代答案
                for j in range(n):
                    if i != j:
                        s1 += x[j] * a[i][j]
                x[i] = (b[i] - s1) / a[i][i]
                times += 1                  #迭代次数加一
            gap = max(abs(x - tempx))       #与上一次答案模的差

            if gap < g:             #精度满足要求，结束
                break

            elif times > 10000:     #如果迭代超过10000次，结束
                break
                print("10000次迭代仍不收敛")

    print("迭代次数：",times)
    print("计算结果：",x)
```

四、数值结果

A 矩阵：

```
[[10 -2 -1]
 [-2 10 -1]
 [-1 -2  5]]
```

b 矩阵：

```
[ 3 15 10]
```

```
===== 三角分解法 =====
```

```
#U1:
[[10.  -2.  -1. ]
 [ 0.   9.6 -1.2]
 [ 0.   0.   0. ]]

#L1:
[[ 1.         0.         0.       ]
 [-0.2        1.         0.       ]
 [-0.1       -0.22916667  1.       ]]

#U2:
[[10.    -2.    -1.   ]
 [ 0.     9.6   -1.2  ]
 [ 0.     0.     4.625]]
```

```
===== 高斯列主元 =====
```

```
原矩阵
[[10. -2. -1.  3.]
 [-2. 10. -1. 15.]
 [-1. -2.  5. 10.]]
0
```

```
调换前
[[10. -2. -1.  3.]
 [-2. 10. -1. 15.]
 [-1. -2.  5. 10.]]
调换后
0
[[10. -2. -1.  3.]
 [-2. 10. -1. 15.]
 [-1. -2.  5. 10.]]
1
调换前
[[10.  -2.  -1.   3. ]
 [ 0.   9.6 -1.2 15.6]
 [ 0.  -2.2  4.9 10.3]]
调换后
1
[[10.  -2.  -1.   3. ]
 [ 0.   9.6 -1.2 15.6]
 [ 0.  -2.2  4.9 10.3]]
三角矩阵
[[10.    -2.    -1.     3.   ]
 [ 0.     9.6   -1.2   15.6  ]
 [ 0.     0.     4.625 13.875]]
方程的解
[[1. 2. 3.]]

===== Jacobi 迭代法 =====

mr: [[3], [15], [10]]
满足误差要求的结果：
[[0.9999967155648001], [1.9999967163839998], [2.999994593216]]

迭代次数： 12

===== Gauss-Seidal 迭代法 =====

迭代次数： 27

计算结果： [0.99999989 1.99999995 2.99999996]
```

五、计算结果分析

四种方法都可以对线性代数方程组求解，其中三角分解法和高斯列主元消去法求得精确解，而 Jacobi 迭代法和 Gauss-seidal 迭代法求得近似解。Gauss-seidal 迭代法是 Jacobi 迭代法的改进，其效率更高。

六、感想体会

在实际问题中，对于一个数学问题，我们往往不需要求其精确解，只要在误差允许范围内求一个近似的解即可。这是数学家和工程师的区别。

Python 相比于 MATLAB 更加灵活，但是代码复杂性稍微高于 MATLAB。

算法是可以不断优化的，程序也是可以不断优化的。

教师评语	指导教师：　　年　月　日