In the name of God

درس : نظریه ریاضی

استاد : دکتر ابراهیمی

دانشجو : توحید حقیقی سیس (۸۳۰۵۹۸۰۲۱)

تمرین اول :

من در برنامه از ۳ کلاس استفاده کردم که کلاس اول توابع مربوط به Jacobi و GausssSeidel و SuccessiveORelax است و یک فایل main.py است که در ان کلاس ها فراخوانی شده و برنامه اجرا میشود

نمایش ماتریس تغییر وضعیت

در ابتدای برنامه در کلاس main.py من اومدم و مقادیر را از ورودی گرفتم و طول و عرض ماتریس رو هم گرفتم از کاربر این تابع در کلاس jacobi و Gausssidel وجود دارند

```
    def makematrix(self,R,C):
        matrix=[]
        for i in range(C):           # A for loop for row entries
            a =[]
            for j in range(R):      # A for loop for column entries
                 a.append(float(input("Enter newNumber row {} column {} ".format(
i,j))))
            matrix.append(a)
        return np.array(matrix)
```

و در تابع main.py برای فراخوانی این تابع به شکل زیر عمل میکنیم

```
from GaussSeidelModel import GausssSeidel
from JacobiModel import Jacobi
from SuccessiveOverRelaxiation import SuccessiveORelax
import numpy as np

R = int(input("Enter the number of rows:"))
C = int(input("Enter the number of columns:"))

jacob=Jacobi([])
gaussseidel=GausssSeidel([])
successoverrelax=SuccessiveORelax([])
```

که در بالای صفحه هر ۳ کلاس را import میکنیم به صفحه و بعد rowو column را از ورودی میگیریم و به تابع زیر پاس میدهیم تا ماتریس را از ورودی بگیرد

```
matrix=jacob.makematrix(R,C)
```

خروجی به شکل زیر خواهد بود

```
Enter the number of rows:3
Enter the number of columns:3
Enter newNumber row 0 column 0 1
Enter newNumber row 0 column 1 4
Enter newNumber row 0 column 2 5
Enter newNumber row 1 column 0 1
Enter newNumber row 1 column 1 7
Enter newNumber row 1 column 2 8
Enter newNumber row 2 column 0 5
Enter newNumber row 2 column 1 2
Enter newNumber row 2 column 2 4
```

در ادامه توابع مورد استفاده را برای همگرایی گراف استفاده میکنیم اولین تابع jacobi است

کلاس جاکوبی به شکل زیر است که توضیحات ان را در ادامه میدهیم

```
import numpy as np

class Jacobi:
    #make object from class
    def __init__(self,matrix):
        self.matrix=matrix

    def makematrix(self,R,C):
        matrix=[]
        for i in range(C):          # A for loop for row entries
            a =[]
            for j in range(R):      # A for loop for column entries
                 a.append(float(input("Enter newNumber row {} column {} ".format(
i,j))))
            matrix.append(a)
        return np.array(matrix)

    def jacobi_func(self,c,x,d):
        # Jacobi
        # تعریفepsilon برای شرط پایان
```

```python
epsilon = 0.001
# درست میکنیم ۱یک ماتریس تمام
x_d = np.ones(d.shape)

x_new = x.copy()
a=1
while(not(x_d<epsilon).all()):
    x = x_new.copy()
    # را پیدا میکنیم Cx+D در هر حلقه مقدار
    # جدید قرار میدهیم x و بعد در
    x_new = c@x+d
    print(a)
    print(x_new)
    a+=1
    #قدر نسبت فاصله را در نظر میگیریم
    x_d = abs(x_new - x)
```

در تابع jacobi func

چند تا متغیر اصلی داریم وقتی ما از ورودی Ax=B

را میگیریم برای محاسبه ان آن را تبدیل به x=Cx+D میکنیم

که در زیر یک مثال نمونه از C و D خواهیم داشت

$$\boxed{\begin{array}{c} Ax = b \quad \Rightarrow \\ x = Cx + d \;, \quad C_{ii} = 0 \end{array}}$$

- **Jacobi method**
- **Gauss-Seidel Method***
- **Successive Over Relaxation (SOR)**

$$Ax = b \iff x^j = Cx^{j-1} + d\,;\; C_{ii} = 0$$

- x and d are column vectors, and C is a square matrix

$$C = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} & -\frac{a_{13}}{a_{11}} & -\frac{a_{14}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & -\frac{a_{23}}{a_{22}} & -\frac{a_{24}}{a_{22}} \\ -\frac{a_{31}}{a_{33}} & -\frac{a_{32}}{a_{33}} & 0 & -\frac{a_{34}}{a_{33}} \\ -\frac{a_{41}}{a_{44}} & -\frac{a_{42}}{a_{44}} & -\frac{a_{43}}{a_{44}} & 0 \end{bmatrix};\quad d = \begin{bmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ \frac{b_3}{a_{33}} \\ \frac{b_4}{a_{44}} \end{bmatrix}$$

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 = b_3 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 = b_4 \end{cases}$$

Can be converted to

$$\begin{cases} x_1 = (b_1 - a_{12}x_2 - a_{13}x_3 - a_{14}x_4)/a_{11} \\ x_2 = (b_2 - a_{21}x_1 - a_{23}x_3 - a_{24}x_4)/a_{22} \\ x_3 = (b_3 - a_{31}x_1 - a_{32}x_2 - a_{34}x_4)/a_{33} \\ x_4 = (b_4 - a_{41}x_1 - a_{42}x_2 - a_{43}x_3)/a_{44} \end{cases}$$

که در زیر مراحل ساخت C و D را مثل عکس های بالا خواهیم داشت

که در main.py قرار دارد

```
A = np.array([[ 4., -1., -1.],
       [ 6.,  8.,  0.],
       [-5.,  0., 12.]])

b = np.array([-2, 45, 80])

# سطر های ماتریس را بر aii تقسیم میکنیم
c = -A/(A.diagonal()).reshape((b.size, -1))
# بعد قطر اصلی را صفر میکنیم
np.fill_diagonal(c, 0)
#d را هم بر قطر اصلی تقسیم میکنیم
d = b/A.diagonal()
# بعد ان را به سطری ستونی تبدیل میکنیم
d = d.reshape((d.size, -1))
# x در ابتدا صفر است یک ماتریس تمام صفر تعریف میکنیم چون

x = np.zeros(d.shape)
```

بعد ۲ تابع jacobi و Gausssidel را از روی این ماتریس های اولیه انتخاب میکنیم

تفاوت این ۲ تابع فقط در این قسمت است که در jacobi در هر مرحله کل x های جدید یافت میشود و در مرحله بعد استفاده میشود ولی در gausssidel در هر مرحله x جدید یافت میشود و از جدیدا در همان مرحله استفاده میشود

کلاس gausssidel به صورت زیر خواهد بود :

```
import numpy as np

class GausssSeidel:
    #make object from class
    def __init__(self,matrix):
        self.matrix=matrix

    def makematrix(self,R,C):
```

```
        matrix=[]
        for i in range(C):          # A for loop for row entries
            a =[]
            for j in range(R):      # A for loop for column entries
                 a.append(float(input("Enter newNumber row {} column {} ".format(
i,j))))
            matrix.append(a)
        return np.array(matrix)

    def gauss_func(self,c,x,d):
        # Gauss-Seidel
        epsilon = 0.001
        x_d = np.ones(d.shape)
        x_new = x.copy()
        a=1
        while(not(x_d<epsilon).all()):
            x = x_new.copy()
            x_new = x.copy()
            for r in range(x.shape[0]):
                x_new[r] = c[r,:] @ x_new + d[r]
            print(a)
            a+=1
            print(x_new)
            x_d = abs(x_new - x)
```

و توضیحات ان مشابه jacobi است

و در قسمت اخر سوال که Successive Over Relaxation است ما باید مقدار لاندا داده شده را اجرا کنیم ببینیم در چه مرحله ای به جواب میرسد

برای مثال مثالی که استاد در اسلاید خود قرار داده ماتریس A و B را در اینجا اجرا کردم و همه اون جواب هایی که در اسلاید بود را به دست اوردم

کلاس Successive Over Relaxation به شکل زیر است که فقط در لاندا با Gausssidel تفاوت دارد

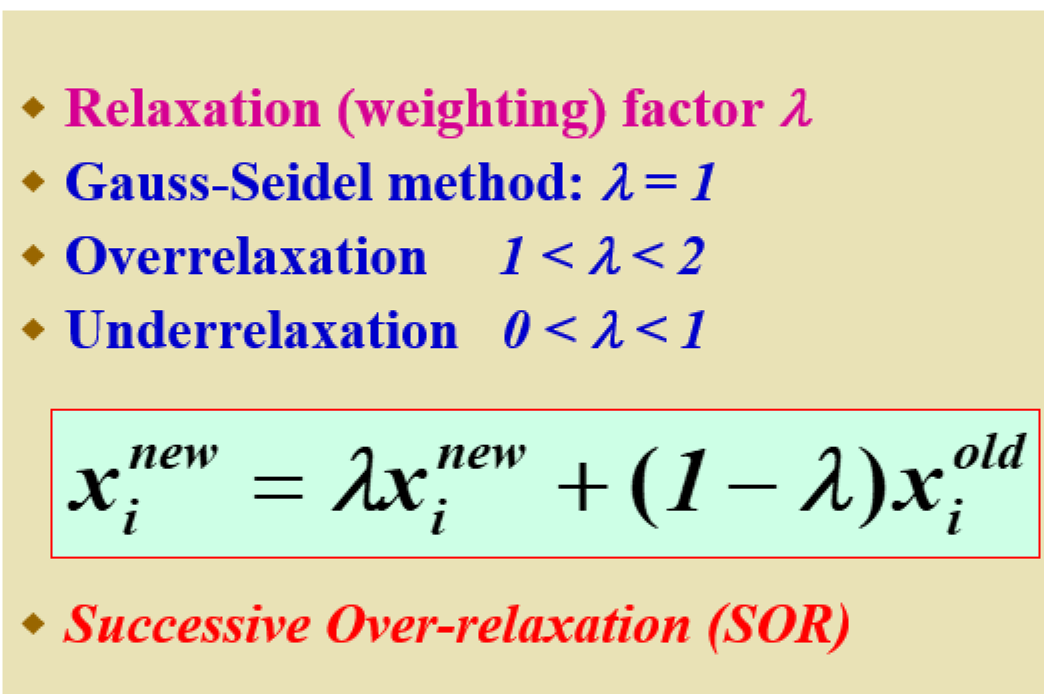
```
import numpy as np

class SuccessiveORelax:
```

```python
    #make object from class
    def __init__(self,matrix):
        self.matrix=matrix

    def SuccessOverRelaxiation(self,c,x,d):
        epsilon = 0.001
        lam = 1.5
        x_d = np.ones(d.shape)
        x_new = x.copy()
        a = 1
        while(not(x_d<epsilon).all()):
            x = x_new.copy()
            x_new = x.copy()
            for r in range(x.shape[0]):
                x_new[r] = ((1-lam) * x_new[r]) + lam * (c[r,:] @ x_new + d[r])
            print(a)
            a += 1
            print(x_new)
            x_d = abs(x_new - x)
```

با استفاده از تابع موجود در اسلاید استاد یافت شده است



خروجی برنامه در هر ۳ قسمت همان خروجی اسلاید های استاد است

به شکل زیر

```
[[-0.5       ]
 [ 5.625     ]
 [ 6.66666667]]

[[2.57291667]
 [6.        ]
 [6.45833333]]

[[2.61458333]
 [3.6953125 ]
 [7.73871528]]

[[2.35850694]
 [3.6640625 ]
 [7.75607639]]

[[2.35503472]
 [3.85611979]
 [7.64937789]]

[[2.37637442]
 [3.85872396]
 [7.64793113]]

[[2.37666377]
 [3.84271918]
 [7.65682268]]

[[2.37488546]
 [3.84250217]
 [7.65694324]]
```

```
1
[[-0.5       ]
 [ 6.        ]
 [ 6.45833333]]
2
[[2.61458333]
 [3.6640625 ]
 [7.75607639]]
3
[[2.35503472]
 [3.85872396]
 [7.64793113]]
4
[[2.37666377]
 [3.84250217]
 [7.65694324]]
5
[[2.37486135]
 [3.84385399]
 [7.65619223]]
6
[[2.37501155]
 [3.84374133]
 [7.65625481]]
```

```
1
[[-0.75   ]
 [ 9.28125]
 [ 9.53125]]
2
[[ 6.6796875 ]
 [-3.71777344]
 [ 9.40917969]]
3
[[-1.95556641]
 [12.49639893]
 [ 4.07318115]]
4
[[ 6.44137573]
 [-5.05724716]
 [11.98926926]]
5
[[-1.37117958]
 [12.50870061]
 [ 3.14837813]]
6
[[ 5.80699432]
 [-4.34971891]
 [12.05518238]]
7
[[-0.76394836]
 [11.47180136]
 [ 3.49494108]]
8
[[ 5.2445026 ]
 [-3.1984661 ]
```