Numerical Analysis assignment No. 2

B6TB1505 Daichi HAYASHI (Ohnishi Lab.)

Oct. 18th, 2019.

1 Assignment Content

To make script of SOR (Successive Over–Relaxation) method. The equation is below.

$$A\boldsymbol{x} = \begin{bmatrix} 4 & -1 & 0 & 1 & 0 \\ -1 & 4 & -1 & 0 & 1 \\ 0 & -1 & 4 & -1 & 0 \\ 1 & 0 & -1 & 4 & -1 \\ 0 & 1 & 0 & -1 & 4 \end{bmatrix} \boldsymbol{x} = \boldsymbol{b} = \begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \\ 100 \end{bmatrix} \tag{1}$$

SOR method is iterative method and the next step equation is expressed in following equation.

$$x_i^{(k+1)} = x^{(k)} + \omega \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij} x_j^{(k)} \right) \tag{2}$$

here, k is step, $i,\ j$ are index, n is the dimension of equation and ω is acceleration of SOR method. If $\omega = 1.0$, this numerical scheme become same as Gauss–Saidel method.

2 Source Code and Result

The Python source code is shown below. The Python version is 3.6.0.

Listing 1: Script source code

```
import numpy as np

def sor(a,b,eps,imax):
  n = b.size # dimension of equation
  x = np.zeros(n, float) # zero matrix

  for it in range(0,imax):
    dxmax = 0.0
    omega = 1.1 # acceleration coefficient
    for i in range(0,n):
      res = b[i] # residual
      for j in range(0,n):
        res -= a[i,j]*x[j]
      dxmax = max(abs(res),dxmax)
      x[i] = x[i] + omega*res/a[i,i]
    print(it, dxmax)
    if dxmax < eps: return x

def main():
  a = np.array([[ 4.0,-1.0, 0.0, 1.0, 0.0],
                [-1.0, 4.0,-1.0, 0.0, 1.0],
                [ 0.0,-1.0, 4.0,-1.0, 0.0],
                [ 1.0, 0.0,-1.0, 4.0,-1.0],
                [ 0.0, 1.0, 0.0,-1.0, 4.0]])
  b = np.array([100.0,100.0,100.0,100.0,100.0])
  imax = 100
  eps = 1e-13

```

```
  print("A, b = ")
  print(a,",",b)
  sol = sor(a,b,eps,imax)
  print(sol)

if __name__=='__main__':
  main()
```

The result output text is shown below. From the result, the iteration number is 24, and numerical solution is

$$\boldsymbol{x} = \begin{bmatrix} 25.00000000 \\ 35.71428571 \\ 42.85714286 \\ 35.71428571 \\ 25.00000000 \end{bmatrix} \tag{3}$$

Listing 2: Output text

```
A, b =
[[ 4. -1.  0.  1.  0.]
 [-1.  4. -1.  0.  1.]
 [ 0. -1.  4. -1.  0.]
 [ 1.  0. -1.  4. -1.]
 [ 0.  1.  0. -1.  4.]] , [100. 100. 100. 100. 100.]
0 135.0625
1 20.92352440063475
2 6.11318410275269
3 1.680628226002007
4 0.48131245986569837
5 0.06447102227491541
6 0.018678017554982773
7 0.005626917202128823
8 0.000929087823124064
9 0.00018554246854307 62
10 5.397835669640472e-05
11 1.273083384489837e-05
12 1.586474542847327e-06
13 4.559077950716528e-07
14 1.5174517642435603e-07
15 2.5687505456062354e-08
16 4.985437840332452e-09
17 1.5905925465631299e-09
18 3.803251047429512e-10
19 4.7130299662967445e-11
20 1.502442614764732e-11
21 4.860112312599085e-12
22 5.968558980384842e-13
23 1.9184653865522705e-13
24 5.684341886080802e-14
[25.         35.71428571 42.85714286 35.71428571 25.        ]
```

3 Discussion

Solution of WolframAlpha [1] is shown below. From this, solution is

$$\boldsymbol{x} = \begin{bmatrix} 25 \\ 250/7 \\ 300/7 \\ 250/7 \\ 25 \end{bmatrix} \tag{4}$$

Compare with this, the numerical solution with SOR is matching up to 8 digits after the decimal point. Therefore, this numerical solution is reasonable.

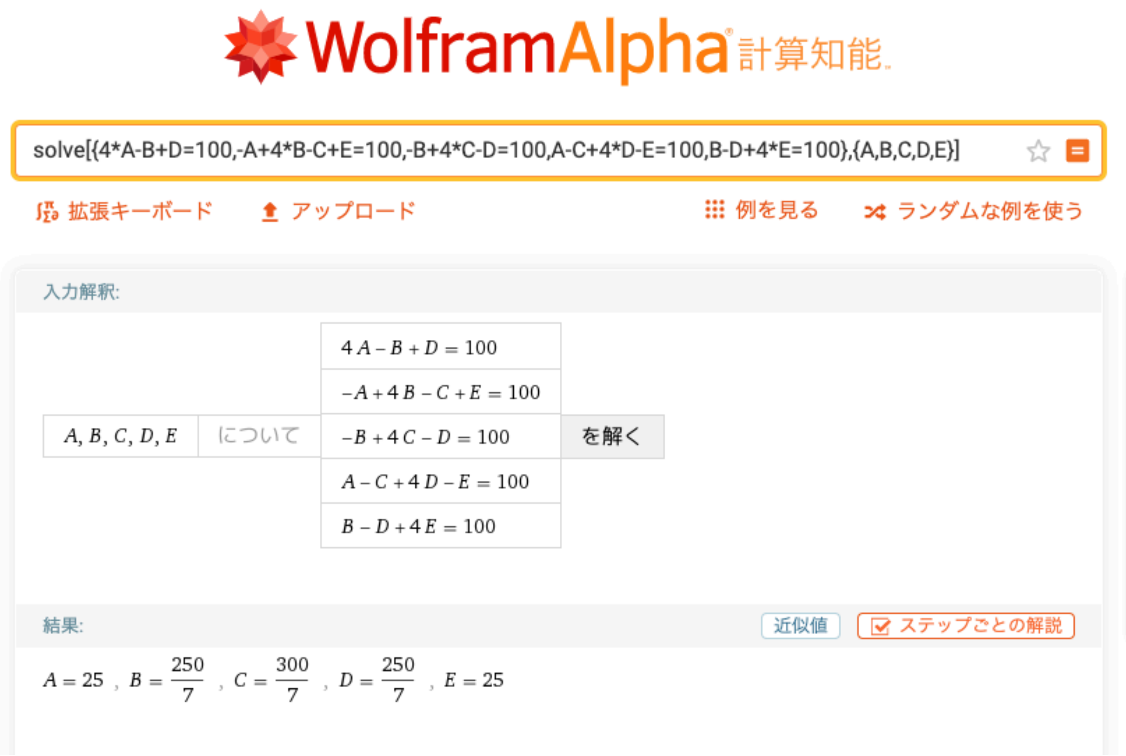

Figure 1 Solution of WalframAlpha [1]

References

[1] WolframAlpha, `https://www.wolframalpha.com`, viewing date: Oct. 16th, 2019.