Computer Application for Scientific Computing

2015-17231
박우정

1. 질문 : 이 행렬의 iterative method들은 수렴하나요? 한다면 이유는 무엇인가요?
Jacobi iteration method와 Gauss-Seidel iteration method의 경우, 주어진 행렬 A의 대각원소는 모두 1.5이고, (대각원소를 제외한 것들의 row별 원소의 합)=(대각원소를 제외한 것들의 column별 원소의 합)=-0.25-0.25=-0.5이므로, 1.5>0.5에서 행렬 A는 diagonally dominated matrix이다. 따라서 두 iteration method는 수렴한다.
한편, SOR로 확대해석하면, $w=0, w=1$인 경우는 이미 증명하였고, 일반적으로는 maximum eigenvalue of $(D+wL)^{-1}((1-w)L+U)$가 1보다 작으면 충분한 조건으로 수렴하게 되는데, $\det(D+wL)^{-1}((1-w)L+U)=(1-w)^{1000}=\prod_{i=1}^{1000}\lambda_i$(단, λ_i는 주어진 1000*1000 행렬의 eigenvalue)임을 이용하면 위 행렬의 maximum eigenvalue의 조건으로부터 $|1-w|<1$, 즉 $0<w<2$를 얻는다. 실제로 2번에서 w가 음수이거나 2를 넘으면 $l2-norm$이 발산하여 수렴하지 않는 것을 알 수 있다.

2. 3가지 iterative method 와 기본 LU의 해를 구하는 시간을 각각 구하고 표로 정리하시오. (표는 깔끔해야 합니다.)

방법	시간
LU 분해	1.6027560000000001
Jacobi	7.1988999999999997E-002
Gauss-Seidel	0.16397499999999998
SOR(w=0.1)	0.77688299999999999
SOR(w=0.5)	0.15797600000000001
SOR(w=0.9)	6.3989999999999991E-002
SOR(w=1.1)	4.6993000000000000E-002
SOR(w=1.5)	0.11898199999999999
SOR(w=1.9)	0.66789799999999999

3. 주어진 3x3 행렬과 RHS 자료에 대해서도 비슷한 계산을 시행해 보고 LU에 의한 sol 과 iter sol 을 비교해본 뒤 iterative method의 장,단점을 서술하시오. iterative method들 간의 차이점도 생각해봅니다. (단 이때 3x3 에 대한 error바운드는 본인이 주고 싶은 값을 줄 것 - 본인이 생각하기에 iterative를 돌리는데 의미가 있어 보이는 값. 초기 값은 (1,1,1))
error를 동일하게 주고, 해를 구하면 다음과 같다. (시간은 모두 0.0000000000000000)

방법	해
LU 분해	-1.0000000000000000 3.0000000000000000 2.0000000000000000
Jacobi	-1.0000000301649803 2.9999999794414380 1.9999999716844545
Gauss-Seidel	-0.99999999284273278 3.0000000038313290 2.0000000003525682
SOR(w=0.1)	0.16009838410360178 0.30281178372552975 0.33014701771288268
SOR(w=0.5)	6.0068264269519389E-002 1.6935153581034239 1.3774744033892978
SOR(w=0.9)	-0.75786493216809081 2.8023770654925362 1.9211781306584486
SOR(w=1.1)	-1.2444998664164761 3.1667757969987802 2.0639675204734726
SOR(w=1.5)	-2.1897810408790734 3.5693430421104977 2.2335766484831927
SOR(w=1.9)	-3.0409626586739482 3.6516477536012872 2.3662437126889575

결과를 보면 SOR을 제외한 방법들은 (-1, 3, 2)에 아주 가까운 수치해를 얻을 수 있었으나, SOR의 경우 w=1.1인 경우를 제외하고는 매우 동떨어져 있음을 확인할 수 있다. 이는 l2 norm을 계산하는 데에 있어서 다른 방법들에 비해 큰 영향을 받기 때문으로 보인다. 허용오차 err을 줄이면 줄일수록 다른 w에 대한 SOR iteration method도 가까운 수치해를 얻을 수 있으나, 수렴하기 위한 iteration number가 너무 커지므로, 적절한 w를 골라서 활용하면 아주 강력한 iterative method가 되지만, 그러한 w를 찾기 위해서는 1번처럼 적절한 bounded condition이나 행렬에 따른 좋은 w를 잘 선택하지 못하면 해의 정확도가 매우 떨어지는 단점이 있다.

```
iter=          65 l2 norm=   1.0601731732604717E-005
iter=          66 l2 norm=   7.9010285879584973E-006
iter=          67 l2 norm=   7.2418807392959200E-006
iter=          68 l2 norm=   5.1812535388134994E-006
iter=          69 l2 norm=   4.8934146777955581E-006
iter=          70 l2 norm=   3.4478271524922130E-006
iter=          71 l2 norm=   3.2668833161013171E-006
iter=          72 l2 norm=   2.3315526963595780E-006
iter=          73 l2 norm=   2.1553787651208704E-006
iter=          74 l2 norm=   1.5976580513747133E-006
iter=          75 l2 norm=   1.4077781333417099E-006
iter=          76 l2 norm=   1.1028331762951727E-006
iter=          77 l2 norm=   9.1333296928098747E-007
iter=          78 l2 norm=   7.6180472512461140E-007
iter=          79 l2 norm=   5.9158691253008819E-007
iter=          80 l2 norm=   5.2362162888829420E-007
iter=          81 l2 norm=   3.8502439277624496E-007
iter=          82 l2 norm=   3.5668217475613767E-007
iter=          83 l2 norm=   2.5341188491298452E-007
iter=          84 l2 norm=   2.4022656024257429E-007
iter=          85 l2 norm=   1.6938934741846236E-007
iter=          86 l2 norm=   1.5984125030278227E-007
iter=          87 l2 norm=   1.1501827394324067E-007
iter=          88 l2 norm=   1.0513954017475638E-007
iter=          89 l2 norm=   7.9026330220761934E-008
it converges at iter <          89
CPU time for calculation is:    0.0000000000000000
The solution is:   -1.0000000301649803        2.9999999794414380        1.9999999716844545
```

그림 1 Jacobi Method

```
[a2015-17231@workstation1 assignment]$ gfortran 3_GaussSeidel.f90
[a2015-17231@workstation1 assignment]$ ./a.out
 Max_iteration?
1000
 iter=           1 l2 norm=   2.9171971243815529
 iter=           2 l2 norm=   2.3906696868908615
 iter=           3 l2 norm=  0.66208726106359039
 iter=           4 l2 norm=  0.19111668859194067
 iter=           5 l2 norm=   5.6512940078960849E-002
 iter=           6 l2 norm=   1.6937669231435110E-002
 iter=           7 l2 norm=   5.1141715712565185E-003
 iter=           8 l2 norm=   1.5504105345290585E-003
 iter=           9 l2 norm=   4.7104870317478853E-004
 iter=          10 l2 norm=   1.4328376201546515E-004
 iter=          11 l2 norm=   4.3611846872474890E-005
 iter=          12 l2 norm=   1.3278866768957817E-005
 iter=          13 l2 norm=   4.0438766002870453E-006
 iter=          14 l2 norm=   1.2316238308794828E-006
 iter=          15 l2 norm=   3.7512984175212656E-007
 iter=          16 l2 norm=   1.1426092192085327E-007
 iter=          17 l2 norm=   3.4803315519426817E-008
 it converges at iter <          17
 CPU time for calculation is:    0.0000000000000000
 The solution is:  -0.99999999284273278        3.0000000038313290        2.0000000003525682
```

그림 2 Gauss_Seidel Method

한편, iterative method간 수렴 속도를 비교하면, 위의 문제에서는 Jacobi는 89번째부터, Gauss_Seidel은 17번째부터 주어진 error 이내로 들어오게 되는데, 그 속도가 후자의 방법이 월등히 빠르다는 것을 확인할 수 있다. 또한 Jacobi method는 각 solution을 동시에 update 하므로 기존의 solution과 iterative solution이 각각 비슷한 error을 갖고 있지만, Gauss-Seidel method의 경우, 지속적으로 update되므로 후에 등장할수록 기존의 solution과 매우 가 가까워지는 것을 확인할 수 있다. 즉, solution의 정확도가 후발로 갈수록 높아지는 장점이 있다.

4. f:R^n ---> R^n, f(x,y,z) = (x^2,y^2,z^2)인 함수를 메인 프로그램 외부에 짜시오. 코드는 한 눈에 보이게 짜야하며, 메인 프로그램에는

real,dimension(3) :: x

x=[2,3,5]

print *, f(x)

와 동등한 문장만 있어야 합니다. 그 외 필수적인 요소는 당연히 더 있어도 됩니다.

Module을 이용하여 3차원 벡터 x의 각 좌표를 받은 후, (x^2, y^2, z^2)을 output으로 출력하는 프로그래밍을 한다. 결과물은 다음과 같다.

```
[a2015-17231@workstation1 assignment]$ gfortran function.f90
[a2015-17231@workstation1 assignment]$ ./a.out
 x=
2
 y=
3
 z=
5
 f is:    4.00000000       9.00000000       25.0000000
```