역행렬과 반복법

김기택

국민대학교 소프트웨어학과

역행렬 (Inverse matrix)

역행렬

- $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ 방정식을 푸는 가장 기초적인 방법은 등호 양쪽에 $\mathbf{A}^{-1}$을 곱하는 것이다.

$$\mathbf{A}^{-1}\,\mathbf{A}\,\mathbf{x} = \mathbf{A}^{-1}\,\mathbf{b} \quad \rightarrow \quad \mathbf{I}\,\mathbf{x} = \mathbf{A}^{-1}\,\mathbf{b}$$

  - $\mathbf{A}^{-1}$ 는 행렬 $\mathbf{A}$ 의 역행렬이다.

- $n \times n$ 행렬 $\mathbf{A}$ 의 역행렬을 구하는 가장 경제적인 방법은 다음 방정식을 푸는 것이다.

$$\mathbf{A}\,\mathbf{X} = \mathbf{I}$$

  - $\mathbf{X}$ 는 $\mathbf{A}$ 의 역행렬이다. ($\mathbf{I}$ 는 단위 행렬)

$$\mathbf{A}^{-1}\,\mathbf{A}\,\mathbf{X} = \mathbf{A}^{-1}\,\mathbf{I} \quad \rightarrow \quad \mathbf{X} = \mathbf{A}^{-1}$$

- 역행렬은 **높은 계산 비용** 때문에 가능하면 피하는 것이 좋다.
  - 방정식을 푸는 과정이므로, 계산 비용은 분해 단계에서 n^3, 해 단계에서 각 외부조건 벡터에서 n^3에 비례하므로 단순한 $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ 를 푸는 비용보다 훨씬 비싸다.
  - 또 다른 심각한 단점은 밴드 행렬의 경우 역행렬 중에 **밴드 구조가 손실**된다. (밴드 행렬의 장점이 없어진다.)

예제 2.13

피벗팅과 함께 LU 분해를 사용하여 역행렬을 구하는 함수를 작성하라. 다음 행렬을 이용하여 테스트하라.

$$\mathbf{A} = \begin{bmatrix} 0.6 & -0.4 & 1.0 \\ -0.3 & 0.2 & 0.5 \\ 0.6 & -1.0 & 0.5 \end{bmatrix}$$

[풀이] 모듈 LUpivot 의 함수 LUdecomp를 사용한다. (모듈 LUdecomp 의 함수 LUdecomp 와 구별한다.) 즉, 피벗팅을 사용하는 프로그램을 사용한다.

예제 2.13 프로그램

```
import numpy as np
from LUpivot import *

def matInv(a):
    n = len(a[0])
    aInv = np.identity(n)    단위행렬 생성
    a, seq = LUdecomp(a)    피벗팅을 하는 LU분해법 사용
    for i in range(n):
        aInv[:,i] = LUsolve(a,aInv[:,i],seq)    LU분해법의 해를 구하기
    return aInv

a = np.array([[ 0.6, -0.4, 1.0], [-0.3, 0.2, 0.5], [ 0.6, -1.0, 0.5]])

aOrig = a.copy()       # Save original [a]
aInv = matInv(a)      # Invert [a] (original [a] is destroyed)

print("\naInv =\n",aInv)
print("\nCheck: a*aInv =\n", np.dot(aOrig,aInv))
```

예제 2.13 프로그램 출력

```
aInv =
[[  1.66666667      -2.22222222      -1.11111111]
 [  1.25            -0.83333333      -1.66666667]
 [  0.5              1.               0. ]]

Check: a*aInv =
[[  1.00000000e+00  -4.44089210e-16  -1.11022302e-16]
 [  0.00000000e+00   1.00000000e+00   5.55111512e-17]
 [  0.00000000e+00  -3.33066907e-16   1.00000000e+00]]
```

예제 2.14

앞서 작성한 프로그램을 이용하여 다음 행렬의 역행렬을 구하라.

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & 5 \end{bmatrix}$$

[풀이] 행렬이 삼중 대각행렬이므로 LUdecmp3 모듈을 사용하여 **A X** = **I** 식을 푼다.

예제 2.14 프로그램

```
## example2_14
import numpy as np
from LUdecomp3 import *

n = 6
d = np.ones(n)*2.0
e = np.ones(n-1)*(-1.0)
c = e.copy()
d[n-1] = 5.0        마지막 행의 마지막 열이 5 임
aInv = np.identity(n)

c,d,e = LUdecomp3(c,d,e)

for i in range(n):
    aInv[:,i] = LUsolve3(c,d,e,aInv[:,i])

print("\nThe inverse matrix is:\n",aInv)
```

예제 2.14 프로그램 출력

```
The inverse matrix is:
[[ 0.84   0.68   0.52   0.36   0.2    0.04]
 [ 0.68   1.36   1.04   0.72   0.4    0.08]
 [ 0.52   1.04   1.56   1.08   0.6    0.12]
 [ 0.36   0.72   1.08   1.44   0.8    0.16]
 [ 0.2    0.4    0.6    0.8    1.     0.2 ]
 [ 0.04   0.08   0.12   0.16   0.2    0.24]]]
```

반복적인 방법

개요

- 지금까지 직접적으로 해를 구하는 방법을 보았다.
  - 직접법의 특징은 유한한 수의 연산으로 해를 계산한다. 컴퓨터의 정밀도가 매우 높다면 해는 정확할 것이다.

- 반복법 (간접법)
  - 해의 초기 추측값으로 시작하여 다음 **x** 의 변화가 무시될 때까지 해를 반복적으로 계산함
  - 장점
    - 계수 행렬의 0 이 아닌 요소만 저장하는 것이 가능함 – 특히 크기가 매우 큰 희소 행렬의 경우 유리하다.
    - 반복적인 과정은 자체 교정(self-correcting) 이다. 반복적인 사이클을 통해 반올림 오류가 다음 번 주기에서 수정된다.
  - 단점
    - 일반적으로 반복 횟수가 클 수 있어 느리다.
    - 항상 해로 수렴하지는 않는다. 계수 행렬이 대각선으로 우세한 경우에만 수렴이 보장된다. (초기 추측은 수렴에 필요한 반복 횟수에만 영향을 준다.)

Gauss-Seidel 방법

- 방정식 $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ 는 스칼라 표기법으로 표현하면 다음과 같다.

$$\sum_{j=1}^{n} A_{ij} x_j = b_i \quad i = 1, 2, \ldots, n$$

- 합산 부호에서 x_i 를 포함하는 항을 추출하면

$$A_{ii} x_i + \sum_{j=1, j\neq i}^{n} A_{ij} x_j = b_i \quad i = 1, 2, \ldots, n$$

- x_i 에 대해 풀면 다음과 같다. (시작 벡터 $\mathbf{x}$ 를 선택하여 시작하고 이전 값과 차이가 충분히 작아 질 때까지 반복한다.)

초기 추측값, x_i

$$\begin{aligned} x_i &= \frac{1}{A_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} A_{ij} x_j\right), \qquad i = 1, 2, \ldots, n \\ x_i &\leftarrow \frac{1}{A_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} A_{ij} x_j\right), \qquad i = 1, 2, \ldots, n \end{aligned} \tag{2.34}$$

Gauss-Seidel 수렴 개선 방법

- 수렴을 개선하는 방법으로서 이완(relaxation)이라고 알려진 기술을 활용한다.
- 새로운 x_i 값을 이전 값의 가중 평균과 식 (2.34)에 의해 예측된 값으로 결정한다.

$$x_i \leftarrow \frac{\omega}{A_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} A_{ij}x_j\right) + (1-\omega)x_i\,, \qquad i = 1, 2, \dots, n \qquad (2.35)$$

- 여기서 가중값 ***ω* 를 이완 계수(relaxation factor)**라고 한다.
  - ω 가 1 이면 식 (2.34) 와 식 (2.35) 가 동일하여 이완이 일어나지 않는다.
  - $\omega < 1$ 이면 식 (2.35) 는 식 (2.34)에 의해 주어진 값 사이의 보간(interpolation)을 나타낸다. 이것을 언더이완(under relaxation)이라고 하고, $\omega > 1$ 인 경우 외삽(extrapolation) 또는 과다 이완(over relaxation) 이 발생한다.

- 이완 계수를 미리 결정하는 실제적인 방법은 없다.

이완 계수 추정값 계산

- 실행시간 동안 이완 계수의 추정값을 계산할 수 있다.
  - 다음 값을 k 번째 반복 동안 $\mathbf{x}$ 의 변화의 크기라고 하자. (이완 없이 수행)

$$\Delta x^{(k)} = \left|\mathbf{x}^{(k-1)} - \mathbf{x}^{(k)}\right|$$

  - k 가 충분히 큰 경우 (예: $k \geq 5$), 최적값 ω 의 근사값은 다음과 같다. (여기서 p 는 양의 정수이다.)

$$\omega_{opt} \approx \frac{2}{1 + \sqrt{1 - \left(\frac{\Delta x^{(k+p)}}{\Delta x^{(k)}}\right)^{1/p}}}$$

- 이완 기능이 있는 Gauss-Seidel 알고리즘의 필수 요소는 다음과 같다.
  - $\omega = 1$ 인 k 반복을 수행한 후 $\Delta x^{(k)}$를 기록한다. (k = 10 이 합리적 임)
  - 추가적으로 p 반복을 수행한 후, $\Delta x^{(k+p)}$를 기록한다.
  - 식 (2.36) 에서 ω_{opt} 를 계산한 후, $\omega = \omega_{opt}$를 사용하여 모든 후속 반복을 수행한다.

gaussSeidel 프로그램

- 이완 기능을 갖춘 Gauss-Seidel 방법을 구현함.
  - $k = 10$, $p = 1$ 을 사용하여 최적 이완계수를 구함
  - 식 (2.35)의 반복 공식에서 개선된 **x** 를 계산하는 iterEqs 함수를 제공하여야 함 (예제 2.17 참조)

```
## module gaussSeidel
''' x,numIter,omega = gaussSeidel(iterEqs,x,tol = 1.0e-9)
    Gauss-Seidel method for solving [A]{x} = {b}.
    The matrix [A] should be sparse. User must supply the
    function iterEqs(x,omega) that returns the improved {x},
    given the current {x} ('omega' is the relaxation factor).
'''
import numpy as np
import math                                    수렴 한계

def gaussSeidel(iterEqs,x,tol = 1.0e-9):
    omega = 1.0
    k = 10
    p = 1
```

```
  for i in range(1,501):
     xOld = x.copy()    원래 변수값 보호
     x = iterEqs(x,omega)              개선된 x 값 계산
     dx = math.sqrt(np.dot(x-xOld,x-xOld))
     if dx < tol: return x,i,omega          계산 결과 반환

   # Compute relaxation factor after k+p iterations
     if i == k: dx1 = dx
     if i == k + p:
        dx2 = dx
        omega = 2.0/(1.0+math.sqrt(1.0-(dx2/dx1)**(1.0/p)))

  print('Gauss-Seidel failed to converge')    수렴 실패 출력
```

예제 2.15

이완 없이 Gauss-Seidel 방법에 의해 다음 방정식을 풀어라.

$$\begin{bmatrix} 4 & -1 & 1 \\ -1 & 4 & -2 \\ 1 & -2 & 4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 12 \\ -1 \\ 5 \end{bmatrix}$$

[풀이]

주어진 방정식을 반복 공식에 맞추어 정리하면

$$x_1 = \frac{1}{4}(12 + x_2 - x_3)$$

$$x_2 = \frac{1}{4}(-1 + x_1 + 2x_3)$$

$$x_3 = \frac{1}{4}(5 - x_1 + 2x_2)$$

시작값을 $x_1 = x_2 = x_3 = 0$ 으로 선택하여 첫번째 반복을 실행하면,

예제 2.15 (continued)

$$x_1 = \frac{1}{4}(12 + 0 - 0) = 3$$

$$x_2 = \frac{1}{4}[-1 + 3 - 2(0)] = 0.5$$

$$x_3 = \frac{1}{4}[5 - 3 + 2(0.5)] = 0.75$$

두번째 반복을 진행하면 다음과 같다.

$$x_1 = \frac{1}{4}(12 + 0.5 - 0.75) = 2.9375$$

$$x_2 = \frac{1}{4}[-1 + 2.9375 - 2(0.75)] = 0.85938$$

$$x_3 = \frac{1}{4}[5 - 2.9375 + 2(0.85938)] = 0.94531$$

세번째 반복을 진행하면 다음과 같다.

예제 2.15 (continued)

$$x_1 = \frac{1}{4}(12 + 0.85938 - 0.94531) = 2.97852$$
$$x_2 = \frac{1}{4}[-1 + 2.97852 - 2(0.94531)] = 0.96729$$
$$x_3 = \frac{1}{4}[5 - 2.97852 + 2(0.96729)] = 0.98902$$

5번 더 반복하면 결과는 소수점 다섯 자리 내에서 정확한 해 $x_1 = 3$, $x_2 = x_3 = 1$ 과 일치한다.

예제 2.17

이완법을 사용하여 Gauss-Seidel 방법으로 다음 n 개의 동시 방정식을 풀기 위한 컴퓨터 프로그램을 작성하라. (프로그램은 어떤 n 값에도 동작해야 함) n = 20 으로 프로그램을 작성하라. 정확한 해는 $x_i = -n/4 + i/2,\ i=1, 2, \ldots, n$ 으로 표시된다.

대칭 행렬이나 밴드 행렬은 아님

$$\begin{bmatrix} 2 & -1 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ -1 & 2 & -1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & -1 & 2 & -1 \\ 1 & 0 & 0 & 0 & \cdots & 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

[풀이] 반복법을 적용하기 위해 행렬을 산술방정식으로 정리할 필요가 있다. 산술방정식을 다시 이완을 적용한 방정식으로 정리한다. 위 방정식은 첫행과 마지막 행을 제외하면 매우 규칙적이다. 이 행렬에 대한 산술 이완식(2.35)는 다음과 같다.

예제 2.17 (continued)

초기 추측값 또는 이전 주기에서 얻어진 값

$$x_1 = \frac{\omega(x_2 - x_n)}{2} + (1-\omega)x_1$$

$$x_i = \frac{\omega(x_{i-1} + x_{i+1})}{2} + (1-\omega)x_i\,, \qquad i = 2, 3, \ldots, n-1$$

$$x_n = \frac{\omega(1 - x_1 + x_{n-1})}{2} + (1-\omega)x_n$$

위 이완식은 프로그램 iterEqs 에서 평가된다.

예제 2.17 프로그램

```
## example2_17
import numpy as np
from gaussSeidel import *

def iterEqs(x,omega):
    n = len(x)
    x[0] = omega*(x[1] - x[n-1])/2.0 + (1.0 - omega)*x[0]
    for i in range(1,n-1):
        x[i] = omega*(x[i-1] + x[i+1])/2.0 + (1.0 - omega)*x[i]
        x[n-1] = omega*(1.0 - x[0] + x[n-2])/2.0 + (1.0 - omega)*x[n-1]
    return x

n = eval(input("Number of equations ==> "))
x = np.zeros(n)
x,numIter,omega = gaussSeidel(iterEqs,x)

print("\nNumber of iterations =",numIter)
print("\nRelaxation factor =",omega)
print("\nThe solution is:\n",x)
```

예제 2.17 프로그램 출력

```
Number of equations ==> 20

Number of iterations = 259

Relaxation factor = 1.7054523107131399

The solution is:
[  -4.50000000e+00  -4.00000000e+00  -3.50000000e+00  -3.00000000e+00
   -2.50000000e+00  -2.00000000e+00  -1.50000000e+00  -9.99999997e-01
   -4.99999998e-01   2.14047151e-09   5.00000002e-01   1.00000000e+00
    1.50000000e+00   2.00000000e+00   2.50000000e+00   3.00000000e+00
    3.50000000e+00   4.00000000e+00   4.50000000e+00   5.00000000e+00 ]
```