연세대학교 통계 데이터 사이언스 학회 ESC 23-2 SUMMER WEEK6

Iterative Methods for Solving Linear Systems

[ESC 방학세션 4조] 김민주 이상윤 조수연

Contents

Part I. Classical iterative method

Part II. Krylov Methods

Introduction

Solving sparse linear system

Iterative method VS Direct method

- $Ax = b \; where \; x^{(k)} \rightarrow x \; as \; k \rightarrow \infty$
- matrix-vector product만 필요

- Ex) LU factorization: A가 sparse여도 L,U는 아닐 수 있으므로 더 복잡할 수 있음.

When to use iterative methods

- Matrix A가 크고 direct method가 computationally expensive할 때
- Ax의 계산이 용이할 때

ESC

1. Classical Iterations or Splitting methods

Classical iterations or splitting methods

Splitting methods

A=M-N, M은 nonsingular

$Ax = b \quad \leftrightarrow \quad Mx - Nx = b \quad \leftrightarrow \quad x = M^{-1}Nx + M^{-1}b \quad \rightarrow$ x=f(x)

즉, linear system Ax=b의 x를 함수 $f(x) = M^{-1}Nx + M^{-1}b$의 fixed point로 본다.

Fixed pint problem의 해를 찾기 위해 x → $f(x)$ → $f(f(x))$ → ⋯ 의 sequence가 수렴할 때까지 이 과정을 반복한다.

Splitting method update rule

Given a matrix $A = M - N$, the update rule for a general splitting method is

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b, \qquad x^{(0)} = 0. \tag{7.1}$$

Classical iterations or splitting methods

Splitting method update rule

Given a matrix $A = M - N$, the update rule for a general splitting method is

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b, \qquad x^{(0)} = 0. \tag{7.1}$$

초기값 x_0에 대해 linear system $Ae = b - Ax_0$의 e를 구하면, $Ax = b$의 solution: $x = x_0 + e$

- When will this process converge?

$x = M^{-1}Nx + M^{-1}$를 만족시키는 exact solution x에 대해 step k의 error $e^{(k)} = x^{(k)} - x$라 하면,

$e^{(k+1)} = x^{(k+1)} - x = M = M^{-1}Nx^{(k)} - M^{-1}Nx = M^{-1}Ne^{(k)}$

$e^{(k+1)} = M^{-1}Ne^{(k)}$

$e^{(k)} = (M^{-1}N)^k e^{(0)}$

즉, $M^{-1}N$의 성질에 따라 convergence가 결정 (Denote **iteration matrix** $\boldsymbol{G = M^{-1}N}$)

Classical iterations or splitting methods

Theorem 7.1: Convergence of splitting methods

Given b, and $A = M - N$ with A and M non-singular, the iteration

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b$$

converges, that is, $x^{(k)} \to x$ for any starting vector $x^{(0)}$ if and only if

$$\rho(M^{-1}N) < 1,$$

where $\rho(X)$ is the largest modulus of the eigenvalues of the matrix X. ($\rho(X)$ is called the spectral radius).

(<=) $\rho(G) < 1$ 이면, $x^{(k)} \to x$ 임을 보이면 됨.
즉, $e^{(k)} = G^{(k)}e^{(0)} \to 0$이 되면 되므로 $G^k \to 0$을 보이면 된다.

$\|G\| *$

Pf) (=>) 먼저, $\rho(G) \geq 1$이면, 가 converge하지 않는 초기값 $x^{(0)}$가 적어도 하나 존재한다. ($\rho(A) = \max\{|\lambda_1|, \ldots, |\lambda_n|\}$)

즉, ∃v s.t $Gv = \lambda v$ with $|\lambda| \geq 1$

Then, 초기값 $x^{(0)} = x + v$ 에 대해 $e^{(k)} = G^{(k)}e^{(0)} = G^k v = \lambda^k v$ 이므로 $k \to \infty$ 일 때, error e가 0으로 수렴 X

Classical iterations or splitting methods

How should we pick M and N?

1. Pick M s.t linear system Mz=d are easy to solve (ex) D(diagonal), L(lower-triangular))
2. Pick M s.t $\rho(M^{-1}N) = \rho(G) < 1$ where A=M-N

↓

I. **Jacobi method**: A=M-N where M=D, N=L+U

II. **Gauss-Seidel method**: A=M-N where M=D-L, N=U

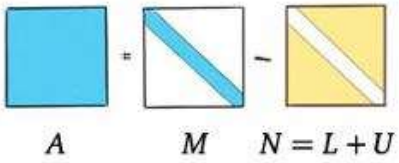
III. **SOR method**: A=M-N where $\mathrm{M} = \frac{1}{w}(D - L),\ N = \left(\frac{1}{w} - 1\right)D + U$

2.Jacobi method

Jacobi method

Jacobi Iteration

- A=M-N where M=D, N=L+U



Splitting method update rule

Given a matrix $A = M - N$, the update rule for a general splitting method is

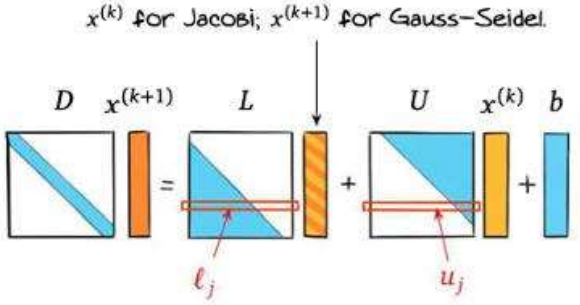
$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b, \qquad x^{(0)} = 0. \tag{7.1}$$

Jacobi iteration

Suppose $A = D - L - U$, as above. The update formula for Jacobi iteration is given by

$$Dx^{(k+1)} = (L + U)x^{(k)} + b.$$

- Matrix-based formula
  - $x^{(k+1)} = D^{-1}(b + (L + U)x^{(k)})$
- Element-based formula
  - $x_i{}^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j \neq i} a_{ij}x_j{}^{(k)})$



Jacobi method

Convergence of Jacobi Iteration

Theorem 7.2: Convergence of Jacobi iteration

If A is strictly diagonally dominant, i.e.,

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|$$

for $i = 1, 2, \ldots, n$, then Jacobi iteration converges for any initial guess $x^{(0)}$.

Gershgorin disc

Let $A \in \mathbb{C}^{n \times n}$. For $1 \leq i \leq n$, the **Gershgorin disc** $\mathscr{D}_i$ is the disc in the complex plane with the center at a_{ii} and radius $r = \sum_{j \neq i} |a_{ij}|$:

$$\mathscr{D}_i = \{z \in \mathbb{C} \mid |z - a_{ii}| \leq \sum_{j \neq i} |a_{ij}|\}.$$

Pf) Iteration matrix $G = D^{-1}(L + U)$에 대해 spectral radius $\rho(G) < 1$ 임을 보이면 된다.

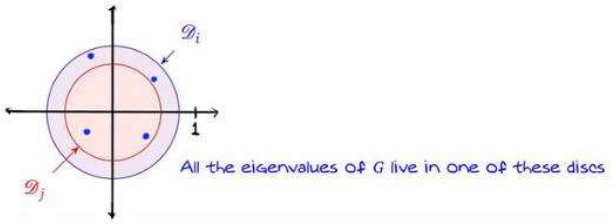
Gershgorin disc theorem(Darve Chap2 page41)에 따라,

Theorem 2.5: Gershgorin disk theorem

All the eigenvalues of $A \in \mathbb{C}^{n \times n}$ are located in one of its Gershgorin discs.

$D_i = \{z \in C \mid |z - g_{ii}| \leq \sum_{j \neq i} |g_{ij}|\}$에서 G의 모든 고유값은 D_i안에 존재한다. (b/c $g_{ii} = 0$ for all i)

따라서, $\sum_{j \neq i} |g_{ij}| = \sum_{j \neq i} \frac{|a_{ij}|}{|a_{ii}|} < 1 \Rightarrow |a_{ii}| > \sum_{j \neq i} |a_{ij}|$



Jacobi method

Example1

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum \right)$$

Ax=b with A = $\begin{bmatrix}2 & 1\\ 5 & 7\end{bmatrix}$, b = $\begin{bmatrix}11\\ 13\end{bmatrix}$ 에 대해 $x^{(0)} = \begin{bmatrix}1\\ 1\end{bmatrix}$ 일 때,

$x^{(k+1)} = D^{-1}(b + (L+U)x^{(k)})$에서, $D^{-1} = \begin{bmatrix}\frac{1}{2} & 0\\ 0 & \frac{1}{7}\end{bmatrix}, L = \begin{bmatrix}0 & 0\\ -5 & 0\end{bmatrix}, U = \begin{bmatrix}0 & -1\\ 0 & 0\end{bmatrix}$

$$x^{(1)} = \begin{bmatrix}0 & -1/2\\ -5/7 & 0\end{bmatrix}\begin{bmatrix}1\\ 1\end{bmatrix} + \begin{bmatrix}11/2\\ 13/7\end{bmatrix} = \begin{bmatrix}5.0\\ 8/7\end{bmatrix} \approx \begin{bmatrix}5\\ 1.143\end{bmatrix}$$

$$x^{(2)} = \begin{bmatrix}0 & -1/2\\ -5/7 & 0\end{bmatrix}\begin{bmatrix}5.0\\ 8/7\end{bmatrix} + \begin{bmatrix}11/2\\ 13/7\end{bmatrix} = \begin{bmatrix}69/14\\ -12/7\end{bmatrix} \approx \begin{bmatrix}4.929\\ -1.714\end{bmatrix}$$

$$\vdots$$

Jacobi method

Example1

Ax=b with A = $\begin{bmatrix} 2 & 1 \\ 5 & 7 \end{bmatrix}$, b = $\begin{bmatrix} 11 \\ 13 \end{bmatrix}$ 에 대해 $x^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ 일 때,

In [2]:
```python
import numpy as np

ITERATION_LIMIT = 1000

# initialize the matrix
A = np.array([[2.,1.],
              [5.,7.]])
# initialize the RHS vector
b = np.array([11.,13.])

# prints the system
print("System:")
for i in range(A.shape[0]):
    row = [f"{A[i, j]}*x{j + 1}" for j in range(A.shape[1])]
    print(f'{" + ".join(row)} = {b[i]}')
print()
```

```
System:
2.0*x1 + 1.0*x2 = 11.0
5.0*x1 + 7.0*x2 = 13.0
```

In [5]:
```python
x = np.zeros_like(b)
for it_count in range(ITERATION_LIMIT):
    if it_count != 0:
        print(f"Iteration {it_count}: {x}")
    x_new = np.zeros_like(x)

    for i in range(A.shape[0]):
        s1 = np.dot(A[i, :i], x[:i])
        s2 = np.dot(A[i, i + 1:], x[i + 1:])
        x_new[i] = (b[i] - s1 - s2) / A[i, i]
        if x_new[i] == x_new[i-1]:
          break

    if np.allclose(x, x_new, atol=1e-4, rtol=0.):
        break

    x = x_new

print("Solution: ")
print(x)
error = np.dot(A, x) - b
print("Error:")
print(error)
```

Jacobi method

Example1

Ax=b with A = $\begin{bmatrix} 2 & 1 \\ 5 & 7 \end{bmatrix}$, b = $\begin{bmatrix} 11 \\ 13 \end{bmatrix}$ 에 대해 $x^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ 일 때,

⋮

```
Iteration 1: [5.5        1.85714286]
Iteration 2: [ 4.57142857 -2.07142857]
Iteration 3: [ 6.53571429 -1.40816327]
Iteration 4: [ 6.20408163 -2.81122449]
Iteration 5: [ 6.90561224 -2.57434402]
Iteration 6: [ 6.78717201 -3.07543732]
Iteration 7: [ 7.03771866 -2.99083715]
Iteration 8: [ 6.99541858 -3.16979904]
Iteration 9: [ 7.08489952 -3.1395847 ]
Iteration 10: [ 7.06979235 -3.20349966]
Iteration 11: [ 7.10174983 -3.19270882]
Iteration 12: [ 7.09635441 -3.21553559]
Iteration 13: [ 7.1077678  -3.21168172]
Iteration 14: [ 7.10584086 -3.21983414]
Iteration 15: [ 7.10991707 -3.21845776]
Iteration 16: [ 7.10922888 -3.22136934]
Iteration 17: [ 7.11068467 -3.22087777]
Iteration 18: [ 7.11043889 -3.22191762]
Iteration 19: [ 7.11095881 -3.22174206]
Iteration 20: [ 7.11087103 -3.22211344]
Iteration 21: [ 7.11105672 -3.22205074]
Iteration 22: [ 7.11102537 -3.22218337]
Solution:
[ 7.11102537 -3.22218337]
Error:
[-0.00013263 -0.00015675]
```

Jacobi method

Example2

Linear system:

$$\begin{aligned} 10x_1 - x_2 + 2x_3 &= 6, \\ -x_1 + 11x_2 - x_3 + 3x_4 &= 25, \\ 2x_1 - x_2 + 10x_3 - x_4 &= -11, \\ 3x_2 - x_3 + 8x_4 &= 15. \end{aligned}$$

Ax=b with A = $\begin{bmatrix} 10 & -1 & 2 & 0 \\ -1 & 11 & -1 & 3 \\ 2 & -1 & 10 & -1 \\ 0 & 3 & -1 & 8 \end{bmatrix}$, b = $\begin{bmatrix} 6 \\ 25 \\ -11 \\ 15 \end{bmatrix}$ 에 대해 $x^{(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ 일

x_1	x_2	x_3	x_4
0.6	2.27272	-1.1	1.875
1.04727	1.7159	-0.80522	0.88522
0.93263	2.05330	-1.0493	1.13088
1.01519	1.95369	-0.9681	0.97384
0.98899	2.0114	-1.0102	1.02135

In [1]:
```python
import numpy as np

ITERATION_LIMIT = 1000

# initialize the matrix
A = np.array([[10., -1., 2., 0.],
              [-1., 11., -1., 3.],
              [2., -1., 10., -1.],
              [0.0, 3., -1., 8.]])
# initialize the RHS vector
b = np.array([6., 25., -11., 15.])

# prints the system
print("System:")
for i in range(A.shape[0]):
    row = [f"{A[i, j]}*x{j + 1}" for j in range(A.shape[1])]
    print(f'{" + ".join(row)} = {b[i]}')
print()
```

```
System:
10.0*x1 + -1.0*x2 + 2.0*x3 + 0.0*x4 = 6.0
-1.0*x1 + 11.0*x2 + -1.0*x3 + 3.0*x4 = 25.0
2.0*x1 + -1.0*x2 + 10.0*x3 + -1.0*x4 = -11.0
0.0*x1 + 3.0*x2 + -1.0*x3 + 8.0*x4 = 15.0
```

Jacobi method

Example2

Linear system:

$$\begin{aligned} 10x_1 - x_2 + 2x_3 &= 6, \\ -x_1 + 11x_2 - x_3 + 3x_4 &= 25, \\ 2x_1 - x_2 + 10x_3 - x_4 &= -11, \\ 3x_2 - x_3 + 8x_4 &= 15. \end{aligned}$$

Ax=b with A = $\begin{bmatrix} 10 & -1 & 2 & 0 \\ -1 & 11 & -1 & 3 \\ 2 & -1 & 10 & -1 \\ 0 & 3 & -1 & 8 \end{bmatrix}$, b = $\begin{bmatrix} 6 \\ 25 \\ -11 \\ 15 \end{bmatrix}$ 에 대해 $x^{(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ 일 때,

x_1	x_2	x_3	x_4
0.6	2.27272	-1.1	1.875
1.04727	1.7159	-0.80522	0.88522
0.93263	2.05330	-1.0493	1.13088
1.01519	1.95369	-0.9681	0.97384
0.98899	2.0114	-1.0102	1.02135

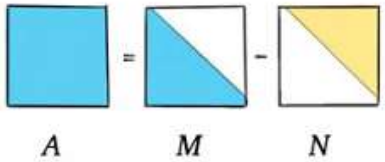
```
Iteration 1: [ 0.6         2.27272727 -1.1         1.875     ]
Iteration 2: [ 1.04727273  1.71590909 -0.80522727  0.88522727]
Iteration 3: [ 0.93263636  2.05330579 -1.04934091  1.13088068]
Iteration 4: [ 1.01519876  1.95369576 -0.96810863  0.97384272]
Iteration 5: [ 0.9889913   2.01141473 -1.0102859   1.02135051]
Iteration 6: [ 1.00319865  1.99224126 -0.99452174  0.99443374]
Iteration 7: [ 0.99812847  2.00230688 -1.00197223  1.00359431]
Iteration 8: [ 1.00062513  1.9986703  -0.99903558  0.99888839]
Iteration 9: [ 0.99967415  2.00044767 -1.00036916  1.00061919]
Iteration 10: [ 1.0001186   1.99976795 -0.99982814  0.99978598]
Iteration 11: [ 0.99994242  2.00008477 -1.00006833  1.0001085 ]
Iteration 12: [ 1.00002214  1.99995896 -0.99996916  0.99995967]
Iteration 13: [ 0.99998973  2.00001582 -1.00001257  1.00001924]
Iteration 14: [ 1.00000409  1.99999268 -0.99999444  0.9999925 ]
Iteration 15: [ 0.99999816  2.00000292 -1.0000023   1.00000344]
Iteration 16: [ 1.00000075  1.99999868 -0.99999899  0.99999862]
Iteration 17: [ 0.99999967  2.00000054 -1.00000042  1.00000062]
Iteration 18: [ 1.00000014  1.99999976 -0.99999982  0.99999975]
Iteration 19: [ 0.99999994  2.0000001  -1.00000008  1.00000011]
Iteration 20: [ 1.00000003  1.99999996 -0.99999997  0.99999995]
Iteration 21: [ 0.99999999  2.00000002 -1.00000001  1.00000002]
Iteration 22: [ 1.          1.99999999 -0.99999999  0.99999999]
Iteration 23: [ 1.  2. -1.  1.]
Iteration 24: [ 1.  2. -1.  1.]
Iteration 25: [ 1.  2. -1.  1.]
Iteration 26: [ 1.  2. -1.  1.]
Iteration 27: [ 1.  2. -1.  1.]
Iteration 28: [ 1.  2. -1.  1.]
Solution:
[ 1.  2. -1.  1.]
Error:
[ 3.95795396e-10 -7.29656335e-10  5.13315612e-10 -5.86034332e-10]
```

3. Gauss-seidel method

Gauss-seidel method

Gauss-seidel Iteration

- A=M-N where M=D-L, N=U



Splitting method update rule

Given a matrix $A = M - N$, the update rule for a general splitting method is

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b, \qquad x^{(0)} = 0. \tag{7.1}$$

Gauss–Seidel iteration

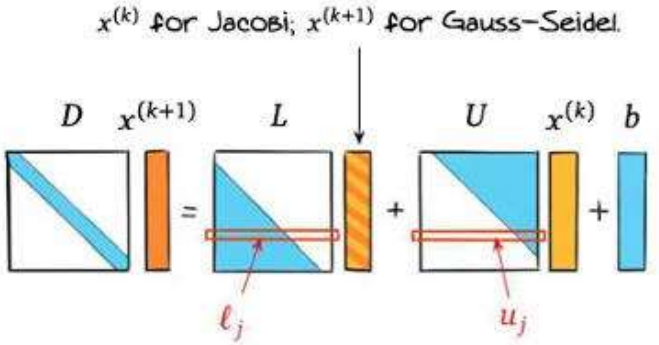
Suppose that $A = D - L - U$, as above. The update formula for Gauss–Seidel iteration is

$$(D - L)x^{(k+1)} = Ux^{(k)} + b.$$

- Matrix-based formula
  - $(D - L)x^{(k+1)} = Ux^{(k)} + b$

    $\Rightarrow Dx^{(k+1)} = Lx^{(k+1)} + Ux^{(k)} + b$

- Element-based formula
  - $x_i{}^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij}x_j{}^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j{}^{(k)})$



Gauss-seidel method

Convergence of Gauss-seidel iteration

Theorem 7.4: Convergence of Gauss–Seidel iteration

If A is symmetric positive definite, Gauss–Seidel iteration converges for any initial guess $x^{(0)}$.

Theorem 7.3: Householder–John theorem

If the real matrices A and $A-B-B^T$ *are both symmetric positive definite, and B is real, then the spectral radius of*

$$H = (A-B)^{-1}B$$

satisfies $\rho(H) < 1$.

Pf) Thm7.3에서 A:=D-L-U, B:= $-U = -L^T$라 하자.

이 때, A와 A-B-B'가 모두 symmetric PD이므로,

$H = (A-B)^{-1}B = (D-L-U+U)^{-1} * (-U) = -(D-L)^{-1}U = -G$이므로

$\rho(H)$<1

따라서, Thm7.1에 의해 A가 symmetric positive definite일 때, convergence가 보장된다.

-A와 A-B-B'가 모두 symmetric PD이고, B가 real이면,

H의 spectral radius가 1보다 작다.

Gauss-seidel method

Example1

Ax=b with A = $\begin{bmatrix}16 & 3\\ 7 & -11\end{bmatrix}$, b = $\begin{bmatrix}11\\ 13\end{bmatrix}$ 에 대해 $x^{(0)} = \begin{bmatrix}1\\ 1\end{bmatrix}$ 일 때,

$x^{(k+1)} = L_*^{-1}(b + Ux^{(k)})$에서, $L_*^{-1} = \begin{bmatrix}16 & 0\\ 7 & -11\end{bmatrix}^{-1}, U = \begin{bmatrix}0 & -3\\ 0 & 0\end{bmatrix}$

$$x^{(1)} = \begin{bmatrix}0.000 & -0.1875\\ 0.000 & -0.1193\end{bmatrix}\begin{bmatrix}1.0\\ 1.0\end{bmatrix} + \begin{bmatrix}0.6875\\ -0.7443\end{bmatrix} = \begin{bmatrix}0.5000\\ -0.8636\end{bmatrix}.$$

$$x^{(2)} = \begin{bmatrix}0.000 & -0.1875\\ 0.000 & -0.1193\end{bmatrix}\begin{bmatrix}0.5000\\ -0.8636\end{bmatrix} + \begin{bmatrix}0.6875\\ -0.7443\end{bmatrix} = \begin{bmatrix}0.8494\\ -0.6413\end{bmatrix}$$

$$x^{(3)} = \begin{bmatrix}0.000 & -0.1875\\ 0.000 & -0.1193\end{bmatrix}\begin{bmatrix}0.8494\\ -0.6413\end{bmatrix} + \begin{bmatrix}0.6875\\ -0.7443\end{bmatrix} = \begin{bmatrix}0.8077\\ -0.6678\end{bmatrix}$$

$$x^{(4)} = \begin{bmatrix}0.000 & -0.1875\\ 0.000 & -0.1193\end{bmatrix}\begin{bmatrix}0.8077\\ -0.6678\end{bmatrix} + \begin{bmatrix}0.6875\\ -0.7443\end{bmatrix} = \begin{bmatrix}0.8127\\ -0.6646\end{bmatrix}$$

$$x^{(5)} = \begin{bmatrix}0.000 & -0.1875\\ 0.000 & -0.1193\end{bmatrix}\begin{bmatrix}0.8127\\ -0.6646\end{bmatrix} + \begin{bmatrix}0.6875\\ -0.7443\end{bmatrix} = \begin{bmatrix}0.8121\\ -0.6650\end{bmatrix}$$

$$x^{(6)} = \begin{bmatrix}0.000 & -0.1875\\ 0.000 & -0.1193\end{bmatrix}\begin{bmatrix}0.8121\\ -0.6650\end{bmatrix} + \begin{bmatrix}0.6875\\ -0.7443\end{bmatrix} = \begin{bmatrix}0.8122\\ -0.6650\end{bmatrix}$$

$$x^{(7)} = \begin{bmatrix}0.000 & -0.1875\\ 0.000 & -0.1193\end{bmatrix}\begin{bmatrix}0.8122\\ -0.6650\end{bmatrix} + \begin{bmatrix}0.6875\\ -0.7443\end{bmatrix} = \begin{bmatrix}0.8122\\ -0.6650\end{bmatrix}$$

Gauss-seidel method

Example1

Ax=b with A = $\begin{bmatrix} 16 & 3 \\ 7 & -11 \end{bmatrix}$, b = $\begin{bmatrix} 11 \\ 13 \end{bmatrix}$ 에 대해 $x^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ 일 때,

```python
import numpy as np

ITERATION_LIMIT = 1000

# initialize the matrix
A = np.array(
    [[16.0, 3.0],
     [7.0, -11.0]]
)
# initialize the RHS vector
b = np.array([11.0, 13.0])

print("System of equations:")
for i in range(A.shape[0]):
    row = [f"{A[i,j]:3g}*x{j+1}" for j in range(A.shape[1])]
    print("[{0}] = [{1:3g}]".format(" + ".join(row), b[i]))

x = np.zeros_like(b, np.float_)
for it_count in range(1, ITERATION_LIMIT):
    x_new = np.zeros_like(x, dtype=np.float_)
    print(f"Iteration {it_count}: {x}")
    for i in range(A.shape[0]):
        s1 = np.dot(A[i, :i], x_new[:i])
        s2 = np.dot(A[i, i + 1 :], x[i + 1 :])
        x_new[i] = (b[i] - s1 - s2) / A[i, i]
    if np.allclose(x, x_new, rtol=1e-8):
        break
    x = x_new

print(f"Solution: {x}")
error = np.dot(A, x) - b
print(f"Error: {error}")
```

```
System of equations:
[ 16*x1 +   3*x2] = [ 11]
[  7*x1 + -11*x2] = [ 13]
Iteration 1: [0. 0.]
Iteration 2: [ 0.6875     -0.74431818]
Iteration 3: [ 0.82705966 -0.65550749]
Iteration 4: [ 0.81040765 -0.66610422]
Iteration 5: [ 0.81239454 -0.66483984]
Iteration 6: [ 0.81215747 -0.6649907 ]
Iteration 7: [ 0.81218576 -0.6649727 ]
Iteration 8: [ 0.81218238 -0.66497485]
Iteration 9: [ 0.81218278 -0.66497459]
Iteration 10: [ 0.81218274 -0.66497462]
Solution: [ 0.81218274 -0.66497462]
Error: [-9.17345417e-08  0.00000000e+00]
```

Gauss-seidel method

Example2

Ax=b with A = $\begin{bmatrix} 2 & 3 \\ 5 & 7 \end{bmatrix}$, b = $\begin{bmatrix} 11 \\ 13 \end{bmatrix}$ 에 대해 $x^{(0)} = \begin{bmatrix} 1.1 \\ 2.3 \end{bmatrix}$ 일 때,

$$x^{(1)} = \begin{bmatrix} 0 & -1.500 \\ 0 & 1.071 \end{bmatrix} \begin{bmatrix} 1.1 \\ 2.3 \end{bmatrix} + \begin{bmatrix} 5.500 \\ -2.071 \end{bmatrix} = \begin{bmatrix} 2.050 \\ 0.393 \end{bmatrix}.$$

$$x^{(2)} = \begin{bmatrix} 0 & -1.500 \\ 0 & 1.071 \end{bmatrix} \begin{bmatrix} 2.050 \\ 0.393 \end{bmatrix} + \begin{bmatrix} 5.500 \\ -2.071 \end{bmatrix} = \begin{bmatrix} 4.911 \\ -1.651 \end{bmatrix}$$

$$x^{(3)} = \cdots.$$

In [6]:

```python
#Gauss-Seidel
import numpy as np

ITERATION_LIMIT = 100

# initialize the matrix
A = np.array([[2.0, 3.0],
              [5.0, 7.0]])
# initialize the RHS vector
b = np.array([11.0, 13.0])

print("System of equations:")
for i in range(A.shape[0]):
    row = [f"{A[i,j]:3g}*x{j+1}" for j in range(A.shape[1])]
    print("[{0}] = [{1:3g}]".format(" + ".join(row), b[i]))

x = np.zeros_like(b, np.float_)
for it_count in range(1, ITERATION_LIMIT):
    x_new = np.zeros_like(x, dtype=np.float_)
    print(f"Iteration {it_count}: {x}")
    for i in range(A.shape[0]):
        s1 = np.dot(A[i, :i], x_new[:i])
        s2 = np.dot(A[i, i + 1 :], x[i + 1 :])
        x_new[i] = (b[i] - s1 - s2) / A[i, i]
    if np.allclose(x, x_new, rtol=1e-8):
        break
    x = x_new

print(f"Solution: {x}")
error = np.dot(A, x) - b
print(f"Error: {error}")
```

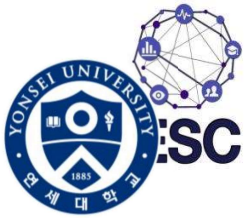
Gauss-seidel method

Example2

Ax=b with A = $\begin{bmatrix} 2 & 3 \\ 5 & 7 \end{bmatrix}$, b = $\begin{bmatrix} 11 \\ 13 \end{bmatrix}$ 에 대해 $x^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ 일 때,

```
Iteration 1: [0. 0.]
Iteration 2: [ 5.5        -2.07142857]
Iteration 3: [ 8.60714286 -4.29081633]
Iteration 4: [11.93622449 -6.66873178]
Iteration 5: [15.50309767 -9.21649833]
Iteration 6: [ 19.3247475  -11.94624822]
Iteration 7: [ 23.41937232 -14.87098023]
Iteration 8: [ 27.80647035 -18.00462168]
Iteration 9: [ 32.50693251 -21.36209465]
Iteration 10: [ 37.54314198 -24.95938713]
Iteration 11: [ 42.93908069 -28.81362907]
Iteration 12: [ 48.7204436 -32.943174 ]
Iteration 13: [ 54.914761   -37.36768643]
Iteration 14: [ 61.55152964 -42.10823546]
Iteration 15: [ 68.66235319 -47.18739513]
Iteration 16: [ 76.2810927  -52.62935193]
Iteration 17: [ 84.44402789 -58.46001992]
Iteration 18: [ 93.19002988 -64.7071642 ]
Iteration 19: [102.5607463  -71.40053307]
Iteration 20: [112.60079961 -78.57199972]
Iteration 21: [123.35799958 -86.25571399]
Iteration 22: [134.88357098 -94.48826499]
Iteration 23: [ 147.23239748 -103.30885534]
Iteration 24: [ 160.46328302 -112.75948787]
Iteration 25: [ 174.6392318  -122.88516557]
Iteration 26: [ 189.82774836 -133.73410597]
Iteration 27: [ 206.10115896 -145.35797068]
Iteration 28: [ 223.53695603 -157.81211145]
Iteration 29: [ 242.21816717 -171.15583369]
Iteration 30: [ 262.23375054 -185.45267896]
Iteration 31: [ 283.67901644 -200.77072745]
Iteration 32: [ 306.65609116 -217.18292227]
Iteration 33: [ 331.27436841 -234.76741672]
Iteration 34: [ 357.65112506 -253.60794649]
Iteration 35: [ 385.91191973 -273.79422636]
Iteration 36: [ 416.19184257 -295.42236755]
Iteration 37: [ 448.63856132 -318.59541523]
Iteration 38: [ 483.39312264 -343.42365917]
Iteration 39: [ 520.63546676 -370.02584912]
Iteration 40: [ 560.53602367 -398.52715977]
Iteration 41: [ 603.29078965 -429.06451404]
Iteration 42: [ 649.09722105 -461.78372932]
Iteration 43: [ 698.17559399 -496.83970999]
Iteration 44: [ 750.75956499 -534.39965926]
Iteration 45: [ 807.09953391 -574.64252422]
Iteration 46: [ 867.46376634 -617.75964786]
Iteration 47: [ 932.13977107 -663.95697934]
Iteration 48: [1001.43546901 -713.45890648]
Iteration 49: [1075.68065965 -766.48632532]
Iteration 50: [1155.22949245 -823.30676035]
Iteration 51: [1240.46017052 -884.18563606]
Iteration 52: [1331.77675413 -949.4133956 ]
Iteration 53: [ 1429.62009371 -1019.30006693]
Iteration 54: [ 1534.4501004  -1094.17864314]
Iteration 55: [ 1646.76796471 -1174.40566906]
Iteration 56: [ 1767.10553362 -1260.3632363 ]
Iteration 57: [ 1896.04485745 -1352.46061247]
Iteration 58: [ 2034.1909187 -1451.1363705]
Iteration 59: [ 2182.20455575 -1556.66039696]
Iteration 60: [ 2340.79059545 -1670.1361396 ]
Iteration 61: [ 2510.70420941 -1791.50300672]
Iteration 62: [ 2692.75451006 -1921.53693577]
Iteration 63: [ 2887.80640365 -2060.66814547]
Iteration 64: [ 3096.7947162  -2210.19906443]
Iteration 65: [ 3320.70662664 -2370.07759046]
Iteration 66: [ 3560.61636569 -2541.44027549]
Iteration 67: [ 3817.66041824 -2725.04315231]
Iteration 68: [ 4093.06472647 -2921.76052034]
Iteration 69: [ 4388.14078051 -3132.52912693]
Iteration 70: [ 4704.2936934  -3356.35268614]
Iteration 71: [ 5043.02895721 -3600.30639801]
Iteration 72: [ 5405.95959701 -3859.5425693 ]
Iteration 73: [ 5794.81385394 -4137.29560996]
Iteration 74: [ 6211.44341494 -4434.88815353]
Iteration 75: [ 6657.83223029 -4753.73730735]
Iteration 76: [ 7136.10596103 -5095.36140073]
Iteration 77: [ 7648.5421011  -5461.38721507]
Iteration 78: [ 8197.58082261 -5853.55773043]
Iteration 79: [ 8785.83659565 -6273.74042546]
Iteration 80: [ 9416.1106382  -6723.93617014]
Iteration 81: [10091.40425521 -7206.28875372]
Iteration 82: [10814.93313058 -7723.09509327]
Iteration 83: [11590.14263991 -8276.81617136]
Iteration 84: [12420.72425705 -8870.08875503]
Iteration 85: [13310.63313255 -9505.73795182]
Iteration 86: [ 14264.10692773 -10186.79066267]
Iteration 87: [ 15285.685994   -10916.48999571]
Iteration 88: [ 16380.23499357 -11698.31070969]
Iteration 89: [ 17552.96606454 -12535.97576039]
Iteration 90: [ 18809.46364058 -13433.47402898]
Iteration 91: [ 20155.71104348 -14395.07931677]
Iteration 92: [ 21598.11897515 -15425.37069654]
Iteration 93: [ 23143.55604481 -16529.25431772]
Iteration 94: [ 24799.38147658 -17711.98676899]
Iteration 95: [ 26573.48015348 -18979.20010963]
Iteration 96: [ 28474.30016444 -20336.92868889]
Iteration 97: [ 30510.89303333 -21791.63788095]
Iteration 98: [ 32692.95682142 -23350.25487245]
Iteration 99: [ 35030.88230867 -25020.20164905]
Solution: [ 37535.80247357 -26809.43033827]
Error: [-5367.68606765     0.        ]
```

$$\mathbf{x} = A^{-1}\mathbf{b} = \begin{bmatrix} -38 \\ 29 \end{bmatrix}$$



4.SOR method

SOR method

SOR method

-Iteration method에서 원하는 것은 convergence뿐만 아니라 fast convergence!

-fast convergence를 위해 parameter w를 이용해 Gauss-Seidel method를 변형한 것

Successive over-relaxation Iteration

- Recall Gauss-Seidel iteration update: $Dx^{(k+1)} = Lx^{(k+1)} + Ux^{(k)} + b$
- 즉, $x_{GS}{}^{(k+1)} = x^{(k)} + \left[D^{-1}\left(b + Lx_{GS}{}^{(k+1)} + Ux^{(k)}\right) - x^{(k)}\right]$
- $x_{GS}{}^{(k+1)} = x^{(k)} + \Delta x_{GS}{}^{(k)}$
- SOS은 update term을 scaling한 것! : $x_{SOR}{}^{(k+1)} = x^{(k)} + w\Delta x_{GS}{}^{(k)}, 0 < w < 2$
- W가 0에 가까우면 Gauss-Seidel metho에 의한 small correction, 2에 가까우면 large correction
- 즉, 적절한 w를 고르면 fast convergence
- $x_{SOR}{}^{(k+1)} = x_{SOR}{}^{(k)} + w\left[D^{-1}\left(b + Lx_{SOR}{}^{(k+1)} + Ux_{SOR}{}^{(k)}\right) - x_{SOR}{}^{(k)}\right]$

SOR method

SOR Iteration

- A=M-N where $M = \frac{1}{w}(D - L),\ N = \left(\frac{1}{w} - 1\right) D + U$

Gauss–Seidel iteration

Suppose that $A = D - L - U$, as above. The update formula for Gauss–Seidel iteration is

$$(D - L)x^{(k+1)} = Ux^{(k)} + b.$$

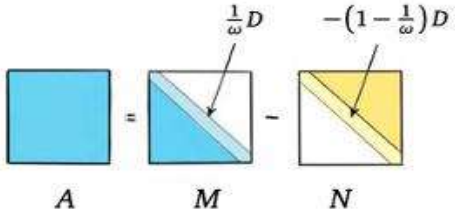
Successive over-relaxation iteration

Let $\omega \in (0, 2)$, and suppose $A = D - L - U$ as above. The update formula for successive over-relaxation iteration is

$$(D - \omega L)\, x_{\text{SOR}}^{(k+1)} = \omega b + \left((1 - \omega)D + \omega U\right) x_{\text{SOR}}^{(k)}.$$

- Iteration matrix ($M^{-1}N$): $G = (D - wL)^{-1}((1 - w)D + wU)$

$$A = M - N,\ M = \frac{1}{\omega}D - L,\ N = \left(\frac{1}{\omega} - 1\right)D + U,$$



SOR method

Convergence of SOR method

Theorem 7.1: Convergence of splitting methods

Given b, and $A = M - N$ with A and M non-singular, the iteration

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b$$

converges, that is, $x^{(k)} \to x$ for any starting vector $x^{(0)}$ if and only if

$$\rho(M^{-1}N) < 1,$$

where $\rho(X)$ is the largest modulus of the eigenvalues of the matrix X. ($\rho(X)$ is called the spectral radius).

-$\rho(G) < 1$일 때, convergence가 보장되었음.

즉, $\rho(G)$가 작을수록 더 빨리 converge

Cf)

Theorem 7.5: Convergence of SOR

For any matrix A, if $\omega \notin (0,2)$, then there is an initial guess $x^{(0)}$ so that SOR will not converge.
Conversely, if A is symmetric positive definite, then the SOR method converges for any $\omega \in (0,2)$.

SOR method

Convergence of SOR method

Theorem 7.6: Eigenvalues of SOR and Jacobi

Suppose that

$$A = D(I - L - U),$$

where L is lower-triangular and U is upper-triangular, so that both have diagonal entries equal to zero.
Recall the iteration matrices we have seen so far:

Jacobi	$G_J = L + U$
Gauss–Seidel	$G_{GS} = (I - L)^{-1}U$
SOR	$G_\omega = (I - \omega L)^{-1}((1-\omega)I + \omega U)$

(so $G_{GS} = G_\omega$ when $\omega = 1$). Assume that the eigenvalues of $\alpha L + \alpha^{-1}U$ are the same as the eigenvalues of G_J for all $\alpha \in \mathbb{C}$. Then, for $\omega \neq 0$ and $\lambda \neq 0$, λ is an eigenvalue of G_ω if and only if

$$\pm \frac{\lambda + \omega - 1}{\omega \lambda^{1/2}}$$

is an eigenvalue of G_J.

Jacobi vs. Gauss–Seidel

Suppose that A satisfies the assumptions of Theorem 7.6. Then

$$\rho(G_{GS}) = \rho(G_J)^2,$$

which suggests that Gauss–Seidel iteration converges in about half the number of iterations required by Jacobi iteration. In fact, this is what we observe in many practical situations.

Optimal choice of w

$$\omega_{\text{opt}} = \frac{2}{1 + \sqrt{1 - \rho(G_J)^2}}$$

$$\rho(G_\omega) = \frac{1 - \sqrt{1 - \rho(G_J)^2}}{1 + \sqrt{1 - \rho(G_J)^2}}.$$

SOR method

Example1

Ax=b with A = $\begin{bmatrix} 3 & -1 & 1 \\ -1 & 3 & -1 \\ 1 & -1 & 3 \end{bmatrix}$, b = $\begin{bmatrix} -1 \\ 7 \\ -7 \end{bmatrix}$ 에 대해 $x^{(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ 일 때,

w=1.25로 SOR method를 적용하면,

Successive over-relaxation iteration

Let $\omega \in (0,2)$, and suppose $A = D - L - U$ as above. The update formula for successive over-relaxation iteration is

$$(D - \omega L)\, x_{\text{SOR}}^{(k+1)} = \omega b + \big((1-\omega)D + \omega U\big)\, x_{\text{SOR}}^{(k)}.$$

$$\begin{aligned} 3x_1 - x_2 + x_3 &= -1 \\ -x_1 + 3x_2 - x_3 &= 7 \\ x_1 - x_2 + 3x_3 &= -7 \end{aligned}$$

$$\begin{aligned} x_1^{(k+1)} &= (-1 + x_2^{(k)} - x_3^{(k)})/3 \\ x_2^{(k+1)} &= (7 + x_1^{(k+1)} + x_3^{(k)})/3 \\ x_3^{(k+1)} &= (-7 - x_1^{(k+1)} + x_2^{(k+1)})/3 \end{aligned}$$

$$\begin{aligned} x_1^{(k+1)} &= (1-\omega)x_1^{(k)} + \omega(-1 + x_2^{(k)} - x_3^{(k)})/3 \\ x_2^{(k+1)} &= (1-\omega)x_2^{(k)} + \omega(7 + x_1^{(k+1)} + x_3^{(k)})/3 \\ x_3^{(k+1)} &= (1-\omega)x_3^{(k)} + \omega(-7 - x_1^{(k+1)} + x_2^{(k+1)})/3 \end{aligned}$$

$$x_1^{(1)} = (1-\omega)x_1^{(0)} + \omega(-1 + x_2^{(0)} - x_3^{(0)})/3 = (1-1.25)\cdot 0 + 1.25\cdot(-1+0-0)/3 = -0.41667$$

$$x_2^{(1)} = (1-\omega)x_2^{(0)} + \omega(7 + x_1^{(1)} + x_3^{(0)})/3 = -0.25\cdot 0 + 1.25\cdot(7-0.41667+0)/3 = 2.7431$$

$$x_3^{(1)} = (1-\omega)x_3^{(0)} + \omega(-7 - x_1^{(1)} + x_2^{(1)})/3 = -0.25\cdot 0 + 1.25\cdot(-7+0.41667+2.7431)/3 = -1.6001$$

$$\begin{aligned} x^{(2)} &= (1.4972, 2.1880, -2.2288)^T, \\ x^{(3)} &= (1.0494, 1.8782, -2.0141)^T, \\ x^{(4)} &= (0.9428, 2.0007, -1.9723)^T, \end{aligned}$$

$= (1, 2, -2)^T$

SOR method

Example2

Ax=b with A = $\begin{bmatrix} 4 & -1 & -6 & 0 \\ -5 & -4 & 10 & 8 \\ 0 & 9 & 4 & -2 \\ 1 & 0 & -7 & 5 \end{bmatrix}$, b = $\begin{bmatrix} 2 \\ 21 \\ -12 \\ -6 \end{bmatrix}$ 에 대해 $x^{(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ 일 때,

w=0.5로 SOR method를 적용하면,

Iteration	x_1	x_2	x_3	x_4
1	0.25	−2.78125	1.6289062	0.5152344
2	1.2490234	−2.2448974	1.9687712	0.9108547
3	2.070478	−1.6696789	1.5904881	0.76172125
...	...	...	...	...
37	2.9999998	−2.0	2.0	1.0
38	3.0	−2.0	2.0	1.0

```python
import numpy as np
from scipy import linalg

def sor_solver(A, b, omega, initial_guess, convergence_criteria):
    step = 0
    phi = initial_guess[:]
    residual = linalg.norm(A @ phi - b)  # Initial residual
    while residual > convergence_criteria:
        for i in range(A.shape[0]):
            sigma = 0
            for j in range(A.shape[1]):
                if j != i:
                    sigma += A[i, j] * phi[j]
            phi[i] = (1 - omega) * phi[i] + (omega / A[i, i]) * (b[i] - sigma)
        residual = linalg.norm(A @ phi - b)
        step += 1
        print("Step {} Residual: {:10.6g}".format(step, residual))
    return phi

# An example case that mirrors the one in the Wikipedia article
residual_convergence = 1e-6
omega = 0.5  # Relaxation factor

A = np.array([[4, -1, -6, 0],
              [-5, -4, 10, 8],
              [0, 9, 4, -2],
              [1, 0, -7, 5]])

b = np.array([2, 21, -12, -6])

initial_guess = np.zeros(4)

phi = sor_solver(A, b, omega, initial_guess, residual_convergence)
print(phi)
```

SOR method

Example2

Ax=b with A = $\begin{bmatrix} 4 & -1 & -6 & 0 \\ -5 & -4 & 10 & 8 \\ 0 & 9 & 4 & -2 \\ 1 & 0 & -7 & 5 \end{bmatrix}$, b = $\begin{bmatrix} 2 \\ 21 \\ -12 \\ -6 \end{bmatrix}$ 에 대해 $x^{(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ 일 때,

w=0.5로 SOR method를 적용하면,

Iteration	x_1	x_2	x_3	x_4
1	0.25	−2.78125	1.6289062	0.5152344
2	1.2490234	−2.2448974	1.9687712	0.9108547
3	2.070478	−1.6696789	1.5904881	0.76172125
...	...	...	...	...
37	2.9999998	−2.0	2.0	1.0
38	3.0	−2.0	2.0	1.0

```
Step 1 Residual:    14.6179
Step 2 Residual:    11.2948
Step 3 Residual:    3.67768
Step 4 Residual:    3.19424
Step 5 Residual:    2.53585
Step 6 Residual:    0.68396
Step 7 Residual:    0.75634
Step 8 Residual:   0.593797
Step 9 Residual:   0.189146
Step 10 Residual:   0.190827
Step 11 Residual:   0.144984
Step 12 Residual:  0.0584829
Step 13 Residual:  0.0500644
Step 14 Residual:  0.0366401
Step 15 Residual:  0.0175857
Step 16 Residual:  0.0134192
Step 17 Residual: 0.00950158
Step 18 Residual:  0.0051081
Step 19 Residual: 0.00363697
Step 20 Residual: 0.00250919
Step 21 Residual: 0.00144923
Step 22 Residual: 0.000991173
Step 23 Residual: 0.000670862
Step 24 Residual: 0.000405216
Step 25 Residual: 0.000270828
Step 26 Residual: 0.000180835
Step 27 Residual: 0.000112307
Step 28 Residual: 7.40858e-05
Step 29 Residual: 4.90057e-05
Step 30 Residual: 3.09616e-05
Step 31 Residual: 2.0275e-05
Step 32 Residual: 1.3326e-05
Step 33 Residual: 8.50875e-06
Step 34 Residual: 5.54917e-06
Step 35 Residual: 3.63169e-06
Step 36 Residual: 2.33393e-06
Step 37 Residual: 1.51871e-06
Step 38 Residual: 9.91113e-07
[ 2.99999971 -2.          1.99999994  0.99999995]
```

Part II. Krylov Methods

Krylov Methods

Krylov Methods

Krolov Methods란 Ax=b 의 해 x 에 대해, n번째 x를 n차 Krylov subspace에서 찾는 방법이다.

즉, iteration의 횟수가 많아질수록 더 많아진 Krylov subspace의 벡터들을 사용하여 x를 나타낼 수 있고, 그만큼 exact solution에 근접하게 된다. 대표적인 방법으로는 GMRES, CG 등이 있다.

$$\mathcal{K}_n = \langle b, Ab, \ldots, A^{n-1}b \rangle \qquad x_n \in \mathcal{K}_n$$

Krylov Methods

GMRES

$$x^{(n)} = argmin \|r^{(n)}\| (= \|Ax^{(n)} - b\|) \text{ where } x^{(n)} \in K_n$$

GMRES란, Generalized Minimal RESidual의 약자로, residual을 최소화하는 x를 찾는 Krylov method이다.

- non-singular한 행렬이라면 어떤 것이든 적용할 수 있다.
- 계산비용이 많이 든다.

Krylov Methods

GMRES

$$
\begin{aligned}
\|r^{(n)}\| &= \|Ax^{(n)} - b\| \\
&= \|AK_n c - b\| \\
&= \|AQ_n y - b\| \\
&= \|Q_{n+1}\tilde{H}_n y - b\| \\
&= \|\tilde{H}_n y - Q_{n+1}^{*} b\| \\
&= \|\tilde{H}_n y - \|b\| e_1\|
\end{aligned}
$$

$AQ_n = Q_{n+1}H_n$

Q_{n+1}^{*}

$\therefore$ *minimize* $\|\tilde{H}_n y - \|b\| e_1\|$

<Procedure>

1. $q_1 = \frac{b}{\|b\|}$
2. generate $\tilde{H}_n$ by Arnoldi iteration
3. find $y = \text{argmin} \|\tilde{H}_n y - \|b\| e_1\|$
4. $x^{(n)} = Q_n y$

2,3,4 반복

Krylov Methods

Properties of GMRES

1. Invariance Property

Scale-invariance. *If A is changed to σA for some $\sigma \in \mathbb{C}$, and b is changed to σb, the residuals $\{r_n\}$ change to $\{\sigma r_n\}$.*

Invariance under unitary similarity transformations. *If A is changed to UAU^* for some unitary matrix U, and b is changed to Ub, the residuals $\{r_n\}$ change to $\{U^* r_n\}$.*

2. Convergence

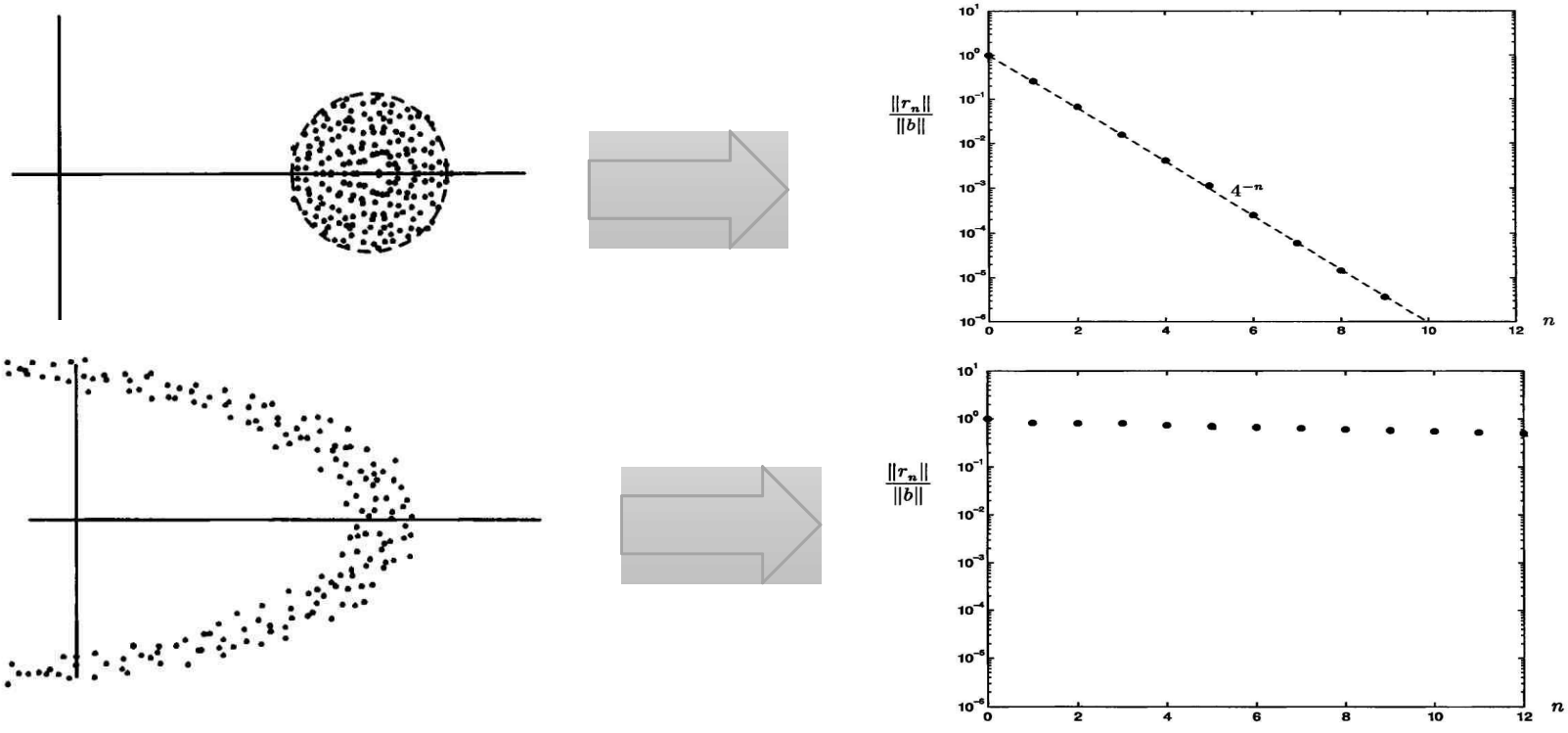
2-1. monotical convergence

$$\|r_{n+1}\| \leq \|r_n\|.$$

2-2 After at most m steps, GMRES must converges. (m 은 A가 m*m 인 행렬 에서 나온 수)

Krylov Methods

Relationship between Location of Eigenvalues and Convergence rate



Krylov Methods

Conjugate Gradient

사전지식

1. CG는 Symmetric & Positive Definite matrix에 대해서만 적용 가능하다.
2. Q_k : krylov space K(A,b,k) 의 orthonomal basis
3. $x^{(k)} = Q_k \mathrm{y} \in K(A,b,k)$: k th iterate을 통해 구한 x.
4. $r^{(k)} = \mathrm{b} - \mathrm{A}x^{(k)} = \mathrm{b} - \mathrm{A}Q_k \mathrm{y}$: k th iterate을 통해 구해진 residual.
5. A-norm : $\|x\|_A = \sqrt{x^T A x}, \; or \; \|x\|_A^2 = x^T A x$

Darve 책 기준, x를 찾는 것을 residual을 최소화하는 것으로 설명하므로, residual의 inverse A norm을 사용할 것임.

$$\|r^{(k)}\|_{A^{-1}} = \sqrt{(r^{(k)})^T A^{-1} r^{(k)}} = \sqrt{(x - x^{(k)})^T A (x - x^{(k)})} = \|x - x^{(k)}\|_A$$

Krylov Methods

Conjugate Gradient

minimize $\|r^{(k)}\|_{A^{-1}}^2 = \|b - Ax^{(k)}\|_{A^{-1}}^2 \leftrightarrow$ **enforce** $r^{(k)} \perp \mathrm{K(A,b,k)}$

residual의 inverse A norm을 최소화하는 것은, iterate을 통해 만들어진 residual 을 krylov space와 직교하도록 만드는 것과 같은 과정이다.

pf)

$$\|r^{(k)}\|_{A^{-1}}^2 = (b - AQ_k y)^T A^{-1}(b - AQ_k y) = b^T x - 2y^T Q_k^T b + y^T Q_k^T A Q_k y.$$

변수는 y 뿐이므로, y에 대해 미분 ->

$$\therefore\ Q_k^T(\mathrm{b} - \mathrm{A}Q_k\mathrm{y}) = 0$$

따라서, 앞으로 생성되는 x들은 residual을 krylov space와 orthogonal하도록 하는 x들이라고 가정한다.

Krylov Methods

Conjugate Gradient

A-Conjugate (A-orthogonal)

: $w^T \mathrm{A} z = 0$

A-Conjugacy of search directions in CG : $(\Delta x^{(k)})^T \mathrm{A}\, \Delta x^{(l)} = 0,\ \mathrm{k} \neq \mathrm{l}$

*search direction: 이전 x 들과 직교하는 방향을 찾아주는 벡터. $\Delta \mathrm{x}$

pf)

$$\mathrm{Def})\ \Delta x^{(k)} = x^{(k+1)} - x^{(k)}$$

$$A\Delta x^{(k)} = Ax^{(k+1)} - \mathrm{A}x^{(k)}$$

$$= (Ax^{(k+1)} - \mathrm{b}) - (\mathrm{A}x^{(k)} - \mathrm{b})$$

$$= -r^{(k+1)} - (-r^{(k)}) = r^{(k)} - r^{(k+1)}$$

이때 residual은 모두 K(A,b,k)와 직교하므로 $\mathrm{A}\Delta x^{(k)} \perp Q_k$

이 때 $\Delta x^{(l)} = x^{(l+1)} - x^{(l)} \in Q_l$ 이 므 로

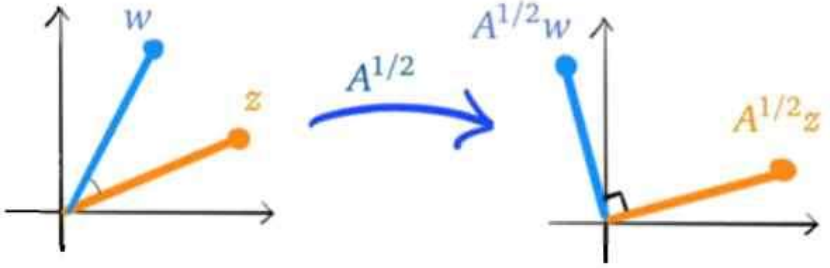
$$\mathrm{A}\Delta x^{(k)} \perp \Delta x^{(l)} \leftrightarrow (\Delta x^{(k)})^T \mathrm{A}\, \Delta x^{(l)} = 0,\ \mathrm{k} \neq \mathrm{l}$$

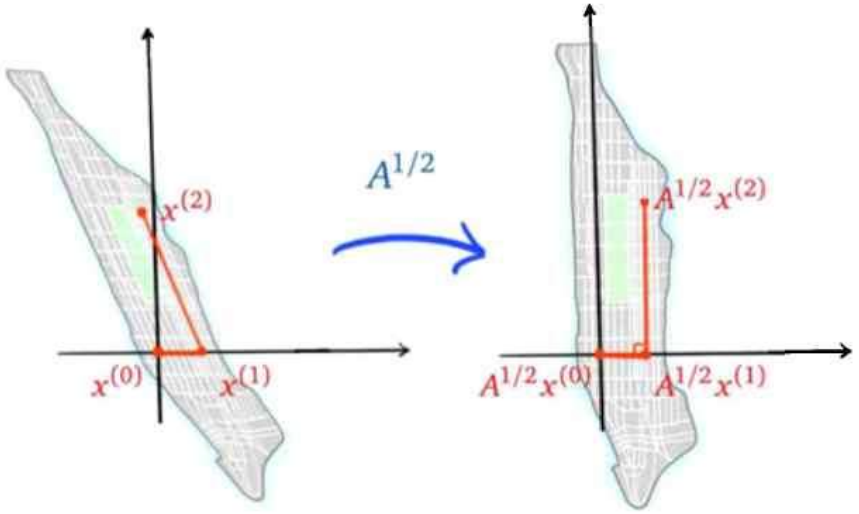
Krylov Methods

Conjugate Gradient

A-conjugacy의 기하적 해석

$$w^T A z = (A^{1/2} w)^T (A^{1/2} z) = 0$$





A-conjugate한 두 벡터 w,z는, $A^{\frac{1}{2}}$ 의 선형변환을 거치면 서로 수직인 벡터이다.

-> CG에서는 search direction이 A-conjugate하므로, $A^{\frac{1}{2}}$ 의 선형변환을 거치면 orthogonal하게 exact solution을 찾아갈 수 있다.

Krylov Methods

Interpretation of CG

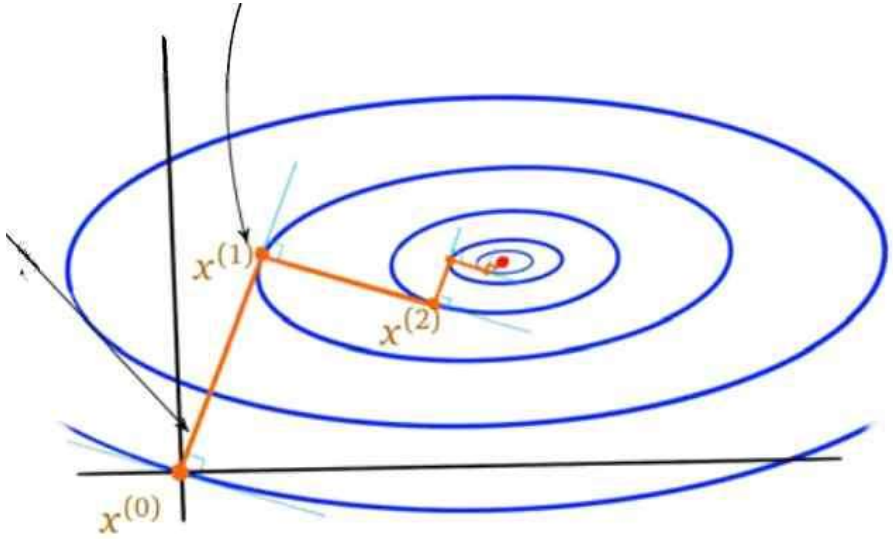


Figure 8.1. *Gradient descent.*

optimal x 를 모르면 사용할 수 없음!

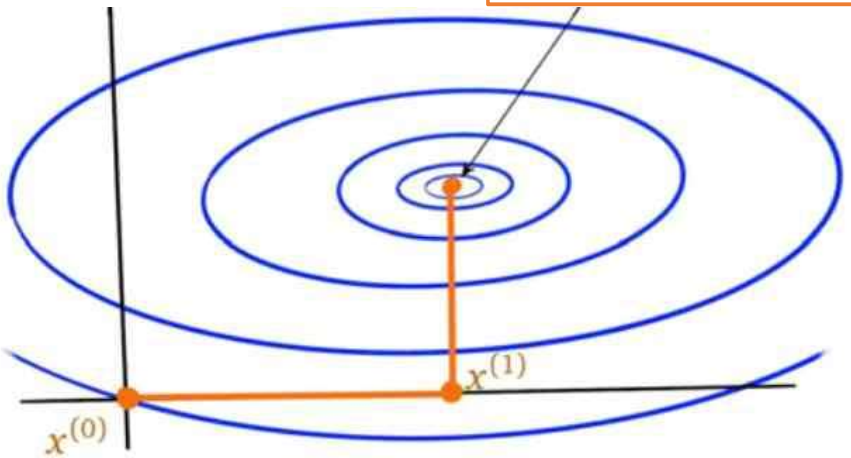
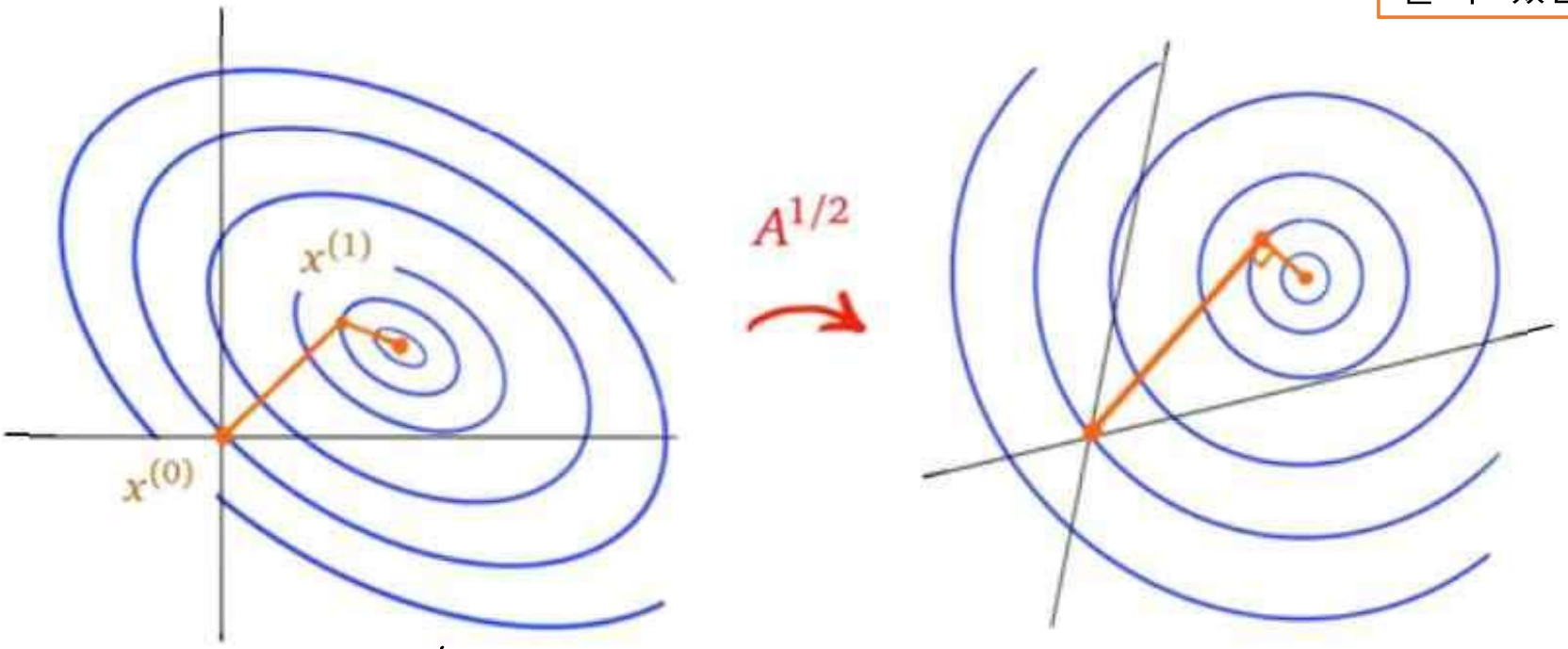


Figure 8.2. *A more efficient search method*

Krylov Methods

Interpretation of CG

optimal x 를 몰라도 사용할 수 있음!



$$(x^{(k)} - x)^T (x^{(k+1)} - x) = 0$$

$$Ax^{(k)} - Ax = 0$$
$$\sim = b$$

Krylov Methods

Conjugate Gradient

Search direction & Step size

$$\Delta x^{(k)} = \mu_{k+1} p^{(l+1)}$$

$p^{(k)}$: k th search direction. 이전 search direction들과 orthogonal하게 생성되어 exact solution을 향해 움직이는 방향을 나타내는 벡터. 크기는 1.

μ_k : k th step size. search direction으로 결정된 방향으로 어느정도 크기만큼 움직이는지를 나타냄.

Krylov Methods

CG algorithm

CG algorithm

Given a symmetric positive definite $n \times n$ matrix A and a vector b:

1. Choose some $x^{(0)}$ (for example, $x^{(0)} = 0$).
2. Let $r^{(0)} = b - Ax^{(0)}$, $p^{(0)} = 0$, and $k = 1$.
3. While $r^{(k-1)} \neq 0$,

$$\tau_{k-1} = \frac{(r^{(k-1)})^T r^{(k-1)}}{(r^{(k-2)})^T r^{(k-2)}},$$
$$p^{(k)} = r^{(k-1)} + \tau_{k-1} p^{(k-1)},$$
$$\mu_k = \frac{(r^{(k-1)})^T r^{(k-1)}}{(p^{(k)})^T A p^{(k)}},$$
$$x^{(k)} = x^{(k-1)} + \mu_k p^{(k)},$$
$$r^{(k)} = r^{(k-1)} - \mu_k A p^{(k)},$$
$$k \leftarrow k + 1.$$

4. Return $x^{(k-1)}$.

* Trefethen 책에선 μ 가 α, τ가 β 로 표기됨

① p를 사용해 x를 update하는 식 찾음 (이미 앞에서 함 ; search direction으로 μ 만큼 이동)

①-1 ①를 응용해 r을 update하는 식을 찾음 (A를 양변에 곱해서)

② p 와 r의 관계식을 이용해 p를 update 하는 식을 찾음
-> 모든 update 식을 벡터p와 스칼라 μ, τ 로 표현 가능

③ p, μ, τ 를 찾음

Krylov Methods

CG algoritm - ①, ①-1

$$\Delta x^{(k)} = M_{k+1} p^{(k+1)} \quad \Rightarrow x^{(k+1)} = x^{(k)} + M_{k+1} p^{(k+1)}$$

$$\| \quad$$

$$x^{(k+1)} - x^{(k)}$$

$$A\Delta x^{(k)} = r^{(k)} - r^{(k+1)}$$

Krylov Methods

CG algorithm - ②

Properties of Search direction

1. Search directions span Krylov space (+ residuals also span Krylov space, too)

2. Search directions are ‘A-conjugate’ with each other. (앞에서 언급한 delta x 의 A-conjugacy 와 같은내용)

$$\left(p^{(\ell)}\right)^T A p^{(k)} = 0 \qquad \text{for all } \ell \neq k.$$

3. Search directions are ‘A-conjugate’ with residuals. $\left(p^{(\ell)}\right)^T A r^{(k)} = 0 \qquad \text{for all } \ell \leq k-1.$

Krylov Methods

CG algorithm - ②

pf of property 1&3)

$\Delta x^{(l+1)} \in K(A, M, l)$

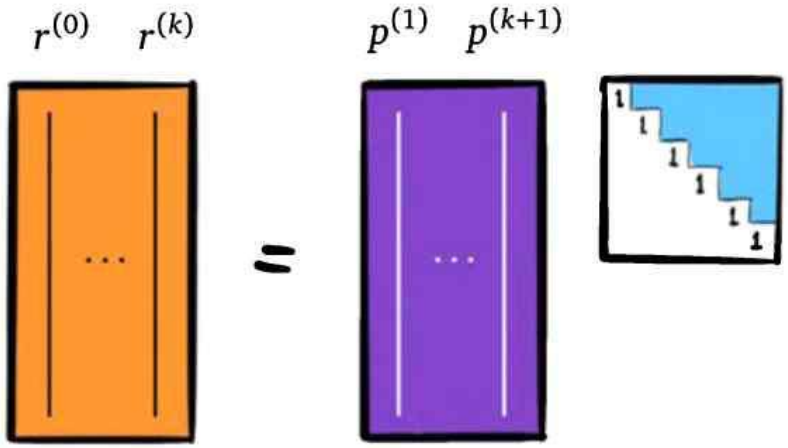
$p^{(l)} \in K(A, M, l) \quad \langle b, Ab, \cdots, A^{(l-1)}b \rangle$

Krylov Methods

CG algorithm - ②

property 1 (p와 r 이 Krylov space를 span) 에 의해, 둘은 같은 span을 갖는다는 것을 알 수 있다.

따라서 r들은 p의 선형결합으로 나타낼 수 있고, 이를 행렬 형태로 나타내면 upper-triangle 형태가 된다. (kth residual을 나타내는데 k th 이하의 p 만 쓰이므로.)



Krylov Methods

CG algorithm - ②

property 2, 3 에 의해, 위의 형태를 bidiagonal 형태로 고칠 수 있다.

$$0 = (p^{(k)})^T A r^{(k)}$$

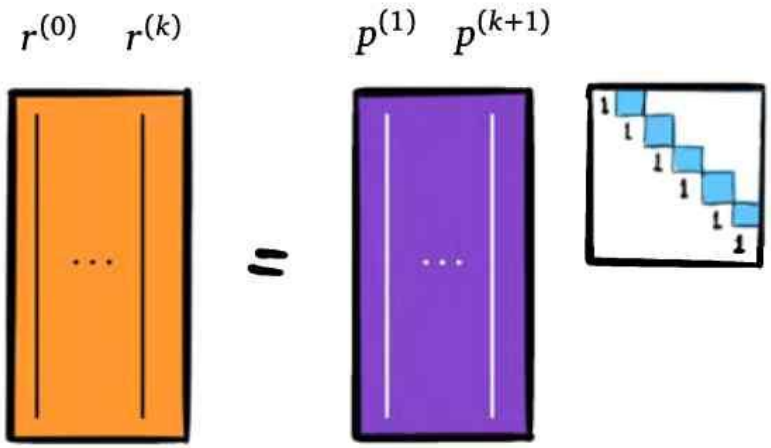
$$= (p^{(k)})^T A (\alpha_1 p^{(1)} + \cdots + \alpha_k p^{(k)} + p^{(k+1)})$$

$$= \alpha_k (p^{(k)})^T A p^{(k)}$$

Krylov Methods

CG algorithm - ②

이 bidiagonal form에서 r을 p로 나타내면 다음과 같다.



$$r^{(k)} = p^{(k+1)} - \tau_k p^{(k)}$$

Krylov Methods

CG algorithm - ③ ; τ, p 구하기

방금 구한 r 과 p 의 관계식에서 τ만 구하면, p를 update하는 식을 찾을 수 있음.

$$r^{(k)} = p^{(k+1)} - \tau_k p^{(k)}$$

$$(p^{(k)})^T A r^{(k)} = (p^{(k)})^T A p^{(k+1)} - \tau_k (p^{(k)})^T A p^{(k)}$$

$$\tau_k = -\frac{(p^{(k)})^T A r^{(k)}}{(p^{(k)})^T A p^{(k)}}$$

$$p^{(k+1)} = r^{(k)} - \tau_k p^{(k)}$$

Krylov Methods

CG algorithm - ③ ; μ 구하기

$$x^{(k+1)} - x^{(k)} = \mu_{k+1} p^{(k+1)}$$
$$r^{(k)} - r^{(k+1)} = \mu_{k+1} A p^{(k+1)}$$
$$(p^{(k+1)})^T r^{(k)} = \mu_{k+1} (p^{(k+1)})^T A p^{(k+1)}$$
$$LHS = (r^{(k)} + \tau_k p^{(k)})^T r^{(k)} = (r^{(k)})^T r^{(k)}$$
$$\therefore \mu_{k+1} = \frac{(r^{(k)})^T r^{(k)}}{(p^{(k+1)})^T A p^{(k+1)}}$$

$$\therefore x^{(k+1)} = x^{(k)} + \mu_{k+1} p^{(k+1)}$$
$$r^{(k+1)} = r^{(k)} + \mu_{k+1} A p^{(k+1)}$$

또 한 $Ap^{(k)} = -\frac{1}{\mu_k}(r^{(k)} - r^{(k-1)})$ 이 므 로

$$\tau_k = -\frac{(p^{(k)})^T A r^{(k)}}{(p^{(k)})^T A p^{(k)}} = \frac{1}{\mu_k} \frac{(r^{(k)} - r^{(k-1)})^T r^{(k)}}{(p^{(k)})^T A p^{(k)}}$$
$$= \frac{(r^{(k)})^T r^{(k)}}{((p^{(k)})^T A p^{(k)}) \mu_k} = \frac{(r^{(k)})^T r^{(k)}}{(r^{(k-1)})^T r^{(k-1)}}$$

으로 τ를 간단하게 정리할 수 있다.
(처음 알고리즘 식에 쓰여진 대로)

Krylov Methods

Interpretation of CG

즉, CG는 최적화를 위해 gradient의 개념을 사용하지만, 그냥 gradient가 아닌 ‘A-conjugate’한 gradient를 사용함 (exact x를 모르므로). 그래서 Conjugate Gradient라 부름

아는 정보만을 활용하기 위해 A를 곱해 A-conjugate의 개념을 추가했는데, 이것을 기하적으로 해석하면 A^(1/2) 만큼 변환한 평면에서 orthogonal한 search direction들을 활용해, 같은 방향으로는 한 번만 움직여 최적값을 찾아가는 것이라 할 수 있음.

Krylov Methods

CG의 장점

장점

1. A가 n*n matrix일 때, n번 이전의 iteration에서 끝낼 수 있다 (원하는 오차범위에 따라)
2. A가 sparse 한 경우, 계산속도가 매우 빠르다.
3. exact solution 이 K(A,b,k)에 있으면, 그 해를 unique하게 찾을 수 있다.

또한, dim(K(A,b,k)) <n 일때 (어느 k 부터, k가 커져도 dim이 더이상 증가하지 않을 때))도 마찬가지로 exact solution을 찾을 수 있다.

Krylov Methods

Convergence of CG

$$\frac{\|e_n\|_A}{\|e_0\|_A} \leq 2 \Big/ \left[\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1} \right)^n + \left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1} \right)^{-n} \right] \leq 2 \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \right)^n \approx 2(1-\frac{1}{\sqrt{\kappa}})^n$$

1. k=condition number of A

convergence rate은 A의 condition number에 비례한다. 즉, condition number가 커질수록 느리게 수렴한다.

2. 원하는 오차의 범위를 설정하면, 언제까지 iteration을 반복하면 되는지 정할 수 있다.

References

[1] Darve, E.,&Wootters, M.(2021). Numerical Linear Algebra with Julia. SIAM

[2] Trefethen, L. N., & Bau, D.(1997). Numerical linear algebra (1st ed.). SIAM

Figure

[1] Darve, E.,&Wootters, M.(2021). Numerical Linear Algebra with Julia. (p237) SIAM

[2] Darve, E.,&Wootters, M.(2021). Numerical Linear Algebra with Julia. (p238) SIAM

[3] Darve, E.,&Wootters, M.(2021). Numerical Linear Algebra with Julia. (p239) SIAM

[4] Darve, E.,&Wootters, M.(2021). Numerical Linear Algebra with Julia. (p241) SIAM

[5] Darve, E.,&Wootters, M.(2021). Numerical Linear Algebra with Julia. (p242) SIAM

[6] Darve, E.,&Wootters, M.(2021). Numerical Linear Algebra with Julia. (p247) SIAM

[7] Trefethen, L. N., & Bau, D.(1997). Numerical linear algebra (1st ed.). (p272,273) SIAM

[8] Darve, E.,&Wootters, M.(2021). Numerical Linear Algebra with Julia. (p270) SIAM

[9] Darve, E.,&Wootters, M.(2021). Numerical Linear Algebra with Julia. (p279) SIAM

[10] Darve, E.,&Wootters, M.(2021). Numerical Linear Algebra with Julia. (p281) SIAM

[11] Darve, E.,&Wootters, M.(2021). Numerical Linear Algebra with Julia. (p271) SIAM

[12] Darve, E.,&Wootters, M.(2021). Numerical Linear Algebra with Julia. (p272) SIAM

END