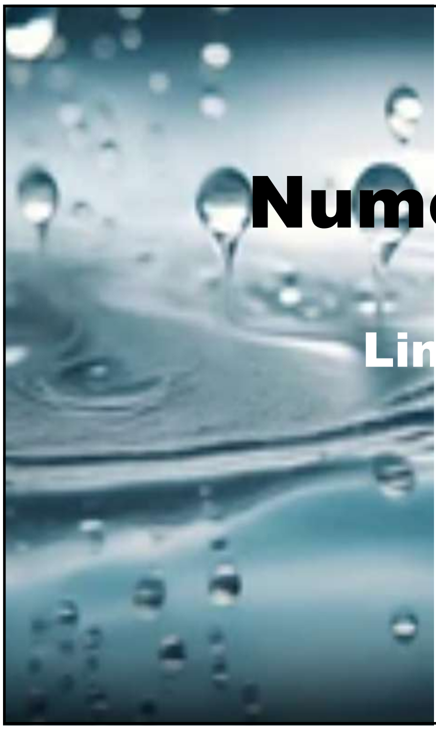


Numerical Analysis

Linear Algebraic Equations

Linear Algebraic Equations

- **선형 방정식**

-. $\mathrm{ax} + \mathrm{by} + \mathrm{c} = 0$ 또는 $\mathrm{ax} + \mathrm{by} = -\mathrm{c}$ 와 같은 형태의 방정식을 x 와 y 에 대한 선형 방정식이라고 부른다.

-. $\mathrm{ax} + \mathrm{by} + \mathrm{cz} = \mathrm{d}$ 는 x, y, z 에 대한 선형 방정식이다.

-. n개의 변수에 대한 선형 방정식은 다음과 같다.

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = \mathrm{b}$$

위의 같은 방정식의 해는 $x_1, x_2, \ldots, x_n$ 의 실수로 구성된다.

만약, 방정식의 개수가 2개 이상이라면, 선형 방정식의 해는 연립 선형 방정식을 통해 구할 수 있다.

Linear Algebraic Equations

Solving Small numbers of Equations

-. 선형 시스템을 구하는 방법은 여러가지 방법이 있다.

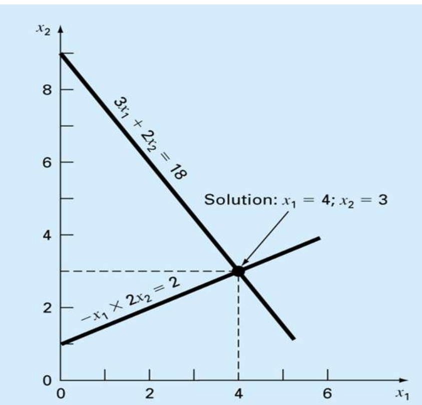
- Graphical method.
- Cramer's rule.
- Method of elimination.
- Computer methods.

For $n \leq 3$

Linear Algebraic Equations

도식적 방법(Graphical Methods)

Example: 두 개의 방정식의 해를 교점을 도식화 해 찾아낼 수 있다.



$$a_{11}x_1 + a_{12}x_2 = b_1$$

$$a_{21}x_1 + a_{22}x_2 = b_2$$

위 두 식을 x_2의 항으로 표현하면 다음과 같다.

$$x_2 = -\left(\frac{a_{11}}{a_{12}}\right)x_1 + \frac{b_1}{a_{12}}$$

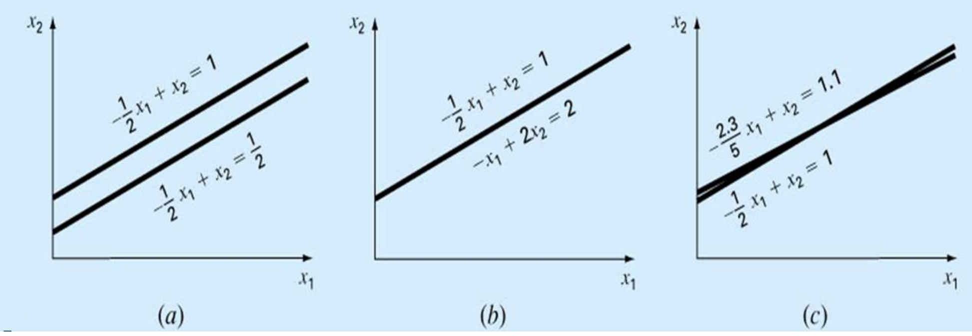
$$x_2 = -\left(\frac{a_{21}}{a_{22}}\right)x_1 + \frac{b_2}{a_{22}}$$

해는 두 선이 만나는 교점이다.

Linear Algebraic Equations

- **도식적 방법(Graphical Methods)**

(a) 해가 없는 경우 (b) 무수한 해가 존재하는 경우, (c)특이한 경우에 가까운 불량 조건



Linear Algebraic Equations

- **행렬식 (Determinants)**

행렬식은 3개의 방정식을 통해 다음과 같이 도식화 해 나타낼 수 있다.

$$[A]\{x\} = \{B\}$$

위 식에서 $[A]$ 는 계수 행렬이다.

$$[A] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Linear Algebraic Equations

- **행렬식 (Determinants)**

모든 행렬이 정방 행렬인 경우, 각 정방 행렬 $[A]$ 에 대해 연관을 짓는 하나의 숫자를 행렬식 D 라고 부른다. 만약 $[A]$ 의 차수가 1이면, $[A]$는 아래와 같이 1개의 원소로 구성되며, 이 원소가 행렬식이 된다.

$$[A] = [a_{11}], D = a_{11}$$

정방 행렬의 차수가 3이라면, 원소 a_{ij}의 소행렬식(minor) 는 $[A]$에서 i행과 j 열을 제외한 차수가 2인 행렬의 행렬식이 된다.

Linear Algebraic Equations

- **행렬식 (Determinants)**

$$D = \begin{vmatrix} \cancel{a_{11}} & \cancel{a_{12}} & \cancel{a_{13}} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

$$D_{11} = \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} = a_{22}a_{33} - a_{32}a_{23}$$

$$D_{12} = \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} = a_{21}a_{33} - a_{31}a_{23}$$

$$D_{13} = \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} = a_{21}a_{32} - a_{31}a_{22}$$

Linear Algebraic Equations

- **행렬식 (Determinants)**

$$D = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

크레이머의 법칙(Cramer's rule): 선형 방정식의 해를 선형 방정식에서 계수 행렬의 행렬식의 비를 통해 나타내는 방법이다. 예를 들어, 미지수 x_1 에 대해서,

$$x_1 = \frac{\begin{vmatrix} b_1 & a_{12} & a_{13} \\ b_2 & a_{22} & a_{23} \\ b_3 & a_{32} & a_{33} \end{vmatrix}}{D}$$

Linear Algebraic Equations

- **소거법 (Method of Elimination)**

-. 선형 방정식의 해를 구하는 기본적인 방법은, 1개의 미지수에 대해서, 1개의 방정식을 연속적으로 구하는 방법이다. 특히, 1개의 미지수에 대해 구한 결과를 통해, 남아있는 선형 방정식으로 부터 해당 되는 미지수를 대체하여, 미지수 자체를 소거해 낼 수 있다.

-. 이러한 소거법은 방정식의 개수가 2개 이상인 경우의 선형 시스템의 문제로 확대할 수 있지만, 이 방법은 수기로 풀기에는 상당히 느리다는 단점이 있다.

Linear Algebraic Equations

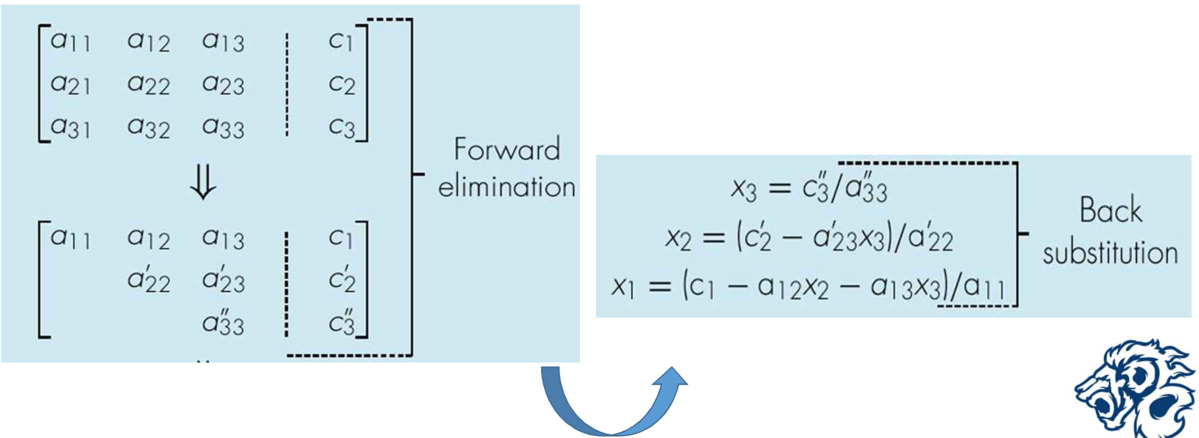
- 가우스 소거법 (Naïve Gauss Elimination)

  -. 일반적인 n 개의 방정식은 다음과 같이 나타낼 수 있다.

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &= b_2 \\
\vdots \qquad\qquad\qquad\qquad & \quad \vdots \\
a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n &= b_n
\end{aligned}
$$

Linear Algebraic Equations

- 가우스 소거법 (Naïve Gauss Elimination)



Linear Algebraic Equations

- **전진 소거 (Forward Elimination)**

-. n 개의 방정식을 위와 같이 상삼각 행렬로 만들기 위해서는 전진소거법을 이용한다.

Pivot 계수

$$\boxed{a_{11}x_1 + a_{12}x_2 + \cdots a_{1n}x_n = b_1}$$ ← Pivot 방정식

$$a_{21}x_1 + a_{22}x_2 + \cdots a_{2n}x_n = b_2$$

$$a_{11}x_1 + a_{12}x_2 + \cdots a_{1n}x_n = b_1$$

$$a_{22}'x_2 + \cdots a_{2n}'x_n = b_2'$$

Linear Algebraic Equations

- **전진 소거 (Forward Elimination)**

-. 먼저, a_{21}을 제거하기 위해서, pivot 방정식에 $\frac{a_{21}}{a_{11}}$을 곱해서, 2번째 식에서 빼준다.

$$a_{11}x_1 + a_{12}x_2 + \cdots a_{1n}x_n = b_1$$

$$a_{11}\frac{a_{21}}{a_{11}}x_1 + a_{12}\frac{a_{21}}{a_{11}}x_2 + \cdots a_{1n}\frac{a_{21}}{a_{11}}x_n = b_1\frac{a_{21}}{a_{11}}$$

$$\boxed{(a_{21} - a_{11}\frac{a_{21}}{a_{11}})x_1 + (a_{22} - a_{12}\frac{a_{21}}{a_{11}})x_2 + \cdots (a_{2n} - a_{1n}\frac{a_{21}}{a_{11}})x_n = (b_2 - b_1\frac{a_{21}}{a_{11}})}$$

0

$$a_{22}'x_2 + \cdots a_{2n}'x_n = b_2'$$

Linear Algebraic Equations

- **전진 소거 (Forward Elimination)**

-. 같은 방법으로 반복하면, 다음과 같이 정리할 수 있다.

$$a_{11}x_1 + a_{12}x_2 + \cdots a_{1n}x_n = b_1$$
$$a_{22}'x_2 + \cdots a_{2n}'x_n = b_2'$$
$$a_{32}'x_2 + \cdots a_{3n}'x_n = b_3'$$
$$\cdots$$
$$a_{n2}'x_2 + \cdots a_{nn}'x_n = b_n'$$

Linear Algebraic Equations

- **전진 소거 (Forward Elimination)**

-. 이제는 a_{22}'가 pivot 계수가 되어, pivot 방정식에 $\frac{a_{32}'}{a_{22}'}$을 곱해서, 세번째 식에서, 빼주면, 다음과 같이 정리된다.

$$a_{22}'x_2 + \cdots a_{2n}'x_n = b_2'$$

$$\boxed{(a_{32}' - a_{22}'\frac{a_{32}'}{a_{22}'})x_2 + (a_{33}' - a_{22}'\frac{a_{32}'}{a_{22}'})x_3 \ldots (a_{3n}' - a_{2n}'\frac{a_{32}'}{a_{22}'})x_n = (b_3' - b_2\frac{a_{32}'}{a_{22}'})}$$

0

$$a_{33}''x_3 + \cdots a_{3n}''x_n = b_3''$$

Linear Algebraic Equations

- **전진 소거 (Forward Elimination)**

-. 같은 방법으로 반복하면, 다음과 같이 정리할 수 있다.

$$a_{11}x_1 + a_{12}x_2 + \cdots a_{1n}x_n = b_1$$
$$a_{22}'x_2 + \cdots a_{2n}'x_n = b_2'$$
$$a_{32}'x_2 + \cdots a_{3n}'x_n = b_3'$$
$$\cdots$$
$$a_{n2}'x_2 + \cdots a_{nn}'x_n = b_n'$$

Linear Algebraic Equations

- **전진 소거 (Forward Elimination)**

-. 같은 방법으로 반복하면, 다음과 같이 정리할 수 있다.

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1$$
$$a_{22}'x_2 + a_{23}'x_3 + \cdots + a_{2n}'x_n = b_2'$$
$$a_{23}''x_3 + \cdots + a_{3n}''x_n = b_3''$$
$$\cdots$$
$$a_{nn}^{(n-1)}x_n = b_n^{(n-1)}$$

Linear Algebraic Equations

- **후진 대입 (Back substitution)**

  -. 가장 아래 쪽의 방정식인 $a_{nn}^{(n-1)} x_n = b_n^{(n-1)}$ 으로 부터,

$$x_n = \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}}$$

  -. (n-1)번째 방정식인 $a_{n(n-1)}^{(n-2)} x_{n-1} + a_{nn}^{(n-2)} x_n = b_n^{(n-2)}$ 으로 부터,

$$a_{n(n-1)}^{(n-2)} x_{n-1} = b_n^{(n-2)} - a_{nn}^{(n-2)} \cdot \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}}$$

  위 식을 통해 x_{n-1}을 구할 수 있다.

Linear Algebraic Equations

- **후진 대입 (Back substitution)**

  -. 앞의 방법을 반복하면, x_i는 다음과 같이 구할 수 있게 된다.

$$x_i = \frac{b_i^{(i-1)} - \sum_{j=i+1}^{n} a_{ij}^{(i-1)} x_j}{a_{ii}^{(i-1)}} \qquad \text{for } i = n-1, n-2, \ldots, 1$$

Linear Algebraic Equations

- **Operation counting**

-. 가우스 소거법의 실행시간은 부동소수점의 연산량에 의해 결정된다.
-. 덧셈과 뺄셈, 곱셈과 나눗셈의 연산시간은 거의 같다.
-. 전체 연산량을 통해, 알고리즘의 소요시간을 추정할 수 있다.

$$\sum_{i=1}^{m} cf(i) = c\sum_{i=1}^{m} f(i) \qquad \sum_{i=1}^{m} f(i) + g(i) = \sum_{i=1}^{m} f(i) + \sum_{i=1}^{m} g(i)$$

$$\sum_{i=1}^{m} 1 = 1 + 1 + 1 + \cdots + 1 = m \qquad \sum_{i=k}^{m} 1 = m - k + 1$$

$$\sum_{i=1}^{m} i = 1 + 2 + 3 + \cdots + m = \frac{m(m+1)}{2} = \frac{m^2}{2} + O(m)$$

$$\sum_{i=1}^{m} i^2 = 1^2 + 2^2 + 3^2 + \cdots + m^2 = \frac{m(m+1)(2m+1)}{6} = \frac{m^3}{3} + O(m^2)$$

Linear Algebraic Equations

- **가우스 소거법에서의 연산량**

: 가우스 소거법은 다음과 같이 의사 코드를 통해 나타낼 수 있다.

```
def gaussian_elimination(A, b):
    n = len(A)

    for k in range(n - 1):
        for i in range(k + 1, n):
            factor = A[i, k] / A[k, k]
            A[i, k:] = A[i, k:] - factor * A[k, k:]
            b[i] = b[i] - factor * b[k]

    x = np.zeros(n)
    for i in range(n - 1, -1, -1):
        x[i] = (b[i] - np.dot(A[i, i + 1:], x[i + 1:])) / A[i, i]

    return x
```

전진 소거

후진 대입

Linear Algebraic Equations

가우스 소거법에서의 연산량

: 먼저, `for i in range(k + 1, n):` 에서, k 는 0부터 (n-2)까지 이므로, i 는 1부터 (n-1)까지 반복하게 된다. 따라서,

$$\sum_{i=2}^{n} 1 = n - 2 + 1 = n - 1$$

: 이고, 1번의 iteration 에서, factor 를 구하는 데, 1번의 나누기가 사용되고, A의 한 항에 대해 소거를 위해서, 1번의 곱셈과 뺄셈이 사용되고, b에 대한 소거를 위해서 1번의 곱셈과 뺄셈이 사용됨을 알 수 있다.

: 따라서, 한 번의 iteration 에서 곱셈의 양은 $1 + (n - 1) + 1 = n + 1$ 이고, 전체에서 곱셈의 양은 $(n - 1)(1 + n)$ 이 된다. 전체 뺄셈의 양은 $(n - 1)(n)$ 이 된다.

Linear Algebraic Equations

가우스 소거법에서의 연산량

: 이제, `for k in range(n - 1):` 에서, k 는 0부터 (n-2)까지 이므로, 앞에서 구한 연산의 양을 확장할 수 있다.

Outer Loop k	Middle Loop i	Addition/Subtraction flops	Multiplication/Division flops
1	2, n	$(n-1)(n)$	$(n-1)(n+1)$
2	3, n	$(n-2)(n-1)$	$(n-2)(n)$
⋮	⋮		
k	$k+1$, n	$(n-k)(n+1-k)$	$(n-k)(n+2-k)$
⋮	⋮		
$n-1$	n, n	(1)(2)	(1) (3)

Linear Algebraic Equations

- **가우스 소거법에서의 연산량**

: 따라서, 전체에서 사용된 덧셈과 뺄셈은

$$\sum_{k=1}^{n-1}(n-k)(n+1-k) = \sum_{k=1}^{n-1}[n(n+1) - k(2n+1) + k^2]$$

: 이고, 이것은 다음과 같이 정리할 수 있다.

$$n(n+1)\sum_{k=1}^{n-1}1 - (2n+1)\sum_{k=1}^{n-1}k + \sum_{k=1}^{n-1}k^2$$

$$= n(n+1)(n-1) - (2n+1)\frac{(n-1)n}{2} + \frac{n(n-1)(2n-1)}{6}$$

$$= \left(n^3 + O(n)\right) - \left(n^3 + O(n^2)\right) + \left(\frac{1}{3}\cdot n^3 + O(n^2)\right) = \frac{1}{3}\cdot n^3 + O(n)$$

Linear Algebraic Equations

- **가우스 소거법에서의 연산량**

: 전체에서 사용된 곱셈과 나눗셈의 양은

$$\sum_{k=1}^{n-1}(n-k)(n+2-k) = n(n+2)\sum_{k=1}^{n-1}1 - n(n+2)\sum_{k=1}^{n-1}k + \sum_{k=1}^{n-1}k^2$$

$$= n(n+2)(n-1) - \left(n^2 + 2n\right)\frac{(n-1)n}{2} + \frac{n(n-1)(2n-1)}{6}$$

$$= \left(n^3 + O(n^2)\right) - \left(n^3 + O(n)\right) + \left(\frac{1}{3}\cdot n^3 + O(n^2)\right) = \frac{1}{3}\cdot n^3 + O(n^2)$$

Linear Algebraic Equations

- **가우스 소거법에서의 연산량**

: 따라서, 전체의 연산량은 다음과 같이 나타낼 수 있다.

$$\frac{1}{3}\cdot n^3 + O(n^2) + \frac{1}{3}\cdot n^3 + O(n) = \frac{2}{3}\cdot n^3 + O(n^2)$$

: 만약, n 이 커지면, $O(n^2)$ 항은 매우 작아지게 되므로, 가우스 소거법에서, 전진대입에 의한 연산량 (FLOP)은 $\frac{2}{3}\cdot n^3$ 으로 수렴한다고 할 수 있다.

Linear Algebraic Equations

- **가우스 소거법에서의 연산량**

: 후진 대입의 경우는, `for i in range(n - 1, -1, -1):` 으로 부터
덧셈과 뺄셈의 경우는 i 가 (n-1) 부터 0까지 진행되는 동안,
뺄셈은 (n-1)일 때를 제외하고는, 한 번의 iteration에 1번씩 수행되고,
덧셈은 (n-1)과 (n-2)를 제외하고는 i-1 만큼씩 수행되므로,
덧셈과 뺄셈의 연산량은

$$\sum_{i=1}^{n-1}(1 + i - 1) = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} = \frac{1}{2}n^2 + O(n)$$

Linear Algebraic Equations

- **가우스 소거법에서의 연산량**

: 후진 대입의 경우는, `for i in range(n - 1, -1, -1):` 으로 부터
나눗셈의 경우는 1번의 iteration에 1번씩 이루어지고, 곱셈의 경우는
A항의 수만큼 수행되므로, 곱셈과 나눗셈에 대한 연산량은,

$$\sum_{i=1}^{n}(1+i-1)=\sum_{i=1}^{n} i=\frac{(n+1)n}{2}=\frac{1}{2}n^2+O(n)$$

따라서, 후진대입에 필요한 총 FLOP은,

$$\frac{1}{2}n^2+O(n)+\frac{1}{2}n^2+O(n)=n^2+O(n)$$

Linear Algebraic Equations

- **가우스 소거법에서의 연산량**

: 따라서, 가우스 소거법에서 사용되는 총 연산량은,

$$\left(\frac{2}{3}n^3+O(n^2)\right)+\left(n^2+O(n)\right)\rightarrow\frac{2}{3}n^3+O(n^2)$$

Linear Algebraic Equations

TABLE 9.1 Number of Flops for Gauss Elimination.

n	Elimination	Back Substitution	Total Flops	$2n^3/3$	Percent Due to Elimination
10	705	100	805	667	87.58%
100	671550	10000	681550	666667	98.53%
1000	6.67×10^8	1×10^6	6.68×10^8	6.67×10^8	99.85%

(1) 선형 시스템의 차원이 늘어나면, 연산시간은 증가하게 되고, FLOP 의 양은 차원 증가분의 세제곱에 비례하여 증가하게 된다.(즉 10 이 증가한다면, FLOP 의 수는 $1000=10^3$ 이 된다.)
(2) 가우스 소거법의 FLOP 의 대부분은 전진 소거에 의해 결정된다.

Linear Algebraic Equations

- **소거법 (Method of Elimination)의 단점**

-. **Division by zero**: 전진 소거 단계나 후진 대입법 단계 모두에서 0으로 나누어지는 경우가 발생할 수 있다.

-. **Round off Errors**: 연속적인 계산 과정에서 반올림 오차가 발생할 수 있다.

-. **Ill-conditioned systems**: 계수 행렬의 계수값이 약간만 바뀌어도 근의 값이 크게 바뀔 수 있는 불량 조건 시스템이 발생할 수 있다. 또는, 2개 이상의 방정식이 매우 유사하다면, 조건식을 만족하는 해들이 넓은 범위를 갖게 될 수 있다. 또한, 반올림에 의해 계수 값의 변화가 일어난다면, 이러한 차이가, 해의 오차를 증가시킬 수 있는 원인이 될 수 있다.

Linear Algebraic Equations

- **소거법 (Method of Elimination)의 단점**

-. **Singular systems**: 두 개의 방정식이 동일하다면, 1개의 자유도를 잃게 되고, n개의 미지수에 대해 n-1개의 방정식을 사용하게 되어, 특이 시스템 형태가 발생할 수 있다. 특히, 선형 시스템의 규모가 큰 경우에는, 특이 행렬 시스템의 행렬식이 0이 된다는 사실을 근거로 하여, 소거 단계 후에, 컴퓨터를 이용한 계산에서 행렬식이 0인지를 검사할 수 있다. 만약, 대각 원속가 0으로 생성된다면, 계산을 종료한다.

Linear Algebraic Equations

- **Techniques for Improving solutions**

-. **유효 숫자**의 개수를 늘려서, 정밀도를 확장시킬 수 있다.
-. **축의 선택법(Pivoting)** : 만약 피벗 원소(pivot element)가 0이면, 정규화 단계에서 0으로 나누어지게 된다. 피벗 원소가 0에 매우 가까워도 동일한 문제가 발생한다. 이러한 경우, 다음 방법을 사용할 수 있다.

(1) Partial pivoting: 가장 큰 원소가 피벗 원소가 되도록 행렬의 행을 교환할 수 있다.
(2) Complete pivoting: 전체 행과 열에서 가장 큰 수를 찾은 후, 교환한다.

Linear Algebraic Equations

- **Techniques for Improving solutions**

-. **유효 숫자**의 개수를 늘려서, 정밀도를 확장시킬 수 있다.
-. **축의 선택법(Pivoting)** : 만약 피벗 원소(pivot element)가 0이면, 정규화 단계에서 0으로 나누어지게 된다. 피벗 원소가 0에 매우 가까워도 동일한 문제가 발생한다. 이러한 경우, 다음 방법을 사용할 수 있다.

(1) 부분 피벗화(Partial pivoting): 가장 큰 원소가 피벗 원소가 되도록 행렬의 행을 교환할 수 있다.
(2) 완전 피벗화(Complete pivoting): 전체 행과 열에서 가장 큰 수를 찾은 후, 행과 열을 교환한다.

Linear Algebraic Equations

- **가우스 소거법 (Naïve Gauss Elimination)**

```
SUB Gauss(a, b, n, x, tol, er)
  DIMENSION s(n)
  er = 0
  DOFOR i = 1, n
    s_i = ABS(a_i,1)
    DOFOR j = 2, n
      IF ABS(a_i,j)>s_i THEN s_i = ABS(a_i,j)
    END DO
  END DO
  CALL Eliminate(a, s, n, b, tol, er)
  IF er ≠ −1 THEN
    CALL Substitute(a, n, b, x)
  END IF
END Gauss
```

```
SUB Eliminate(a, s, n, b, tol, er)
  DOFOR k = 1, n − 1
    CALL Pivot (a, b, s, n, k)
    IF ABS (a_k,k/s_k) < tol THEN
      er = −1
      EXIT DO
    END IF
    DOFOR i = k + 1, n
      factor = a_i,k/a_k,k
      DOFOR j = k + 1, n
        a_i,j = a_i,j − factor*a_k,j
      END DO
      b_i = b_i − factor * b_k
    END DO
  END DO
  IF ABS(a_n,n/s_n) < tol THEN er = −1
END Eliminate
```

Linear Algebraic Equations

- 가우스 소거법 (Naïve Gauss Elimination)

```
SUB Gauss(a, b, n, x, tol, er)
  DIMENSION s(n)
  er = 0
  DOFOR i = 1, n
    s_i = ABS(a_i,1)
    DOFOR j = 2, n
      IF ABS(a_i,j)>s_i THEN s_i = ABS(a_i,j)
    END DO
  END DO
  CALL Eliminate(a, s, n, b, tol, er)
  IF er ≠ −1 THEN
    CALL Substitute(a, n, b, x)
  END IF
END Gauss
```

```
SUB Substitute(a, n, b, x)
  x_n = b_n/a_n,n
  DOFOR i = n − 1, 1, −1
    sum = 0
    DOFOR j = i + 1, n
      sum = sum + a_i,j * x_j
    END DO
    x_n = (b_n − sum) / a_n,n
  END DO
END Substitute
```


Linear Algebraic Equations

- 부분피벗화(Partial pivoting)

  를 통해 대각원소가 0에 가까운 수가 발생되는 지 점검할 수 있다.

```
SUB Pivot(a, b, s, n, k)
  p = k
  big = ABS(a_k,k/s_k)
  DOFOR ii = k + 1, n
    dummy = ABS(a_ii,k/s_ii)
    IF dummy > big THEN
      big = dummy
      p = ii
    END IF
  END DO
  IF p ≠ k THEN
    DOFOR jj = k, n
      dummy = a_p,jj
      a_p,jj = a_k,jj
      a_k,jj = dummy
    END DO
    dummy = b_p
    b_p = b_k
    b_k = dummy
    dummy = s_p
    s_p = s_k
    s_k = dummy
  END IF
END pivot
```

Linear Algebraic Equations

가우스 조던 소거법 (Gauss-Jordan Elimination)

-. 가우스 소거법에서 발전한 형태로, 가장 큰 차이점은 다음과 같다.

(1) 미지수가 제거된 경우, 연관된 방정식에서 뿐만 아니라, 다른 모든 방정식에서 해당 미지수는 제거된다.
(2) 모든 행이 피벗 원소에 의해 나누어져서 정규화 된다.
(3) 소거 단계는 결과적으로 항등 행렬을 만들게 된다.
(4) 해를 구하는 데에 있어, 후진 대입을 할 필요가 없어진다.

Linear Algebraic Equations

가우스 조던 소거법 (Gauss-Jordan Elimination)

-. 가우스 소거법에서 발전한 형태로, 가장 큰 차이점은 다음과 같다.

$$\left[\begin{array}{ccc:c} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array}\right] \quad \left[\begin{array}{ccc:c} 1 & 0 & 0 & b_1^{(n)} \\ 0 & 1 & 0 & b_2^{(n)} \\ 0 & 0 & 1 & b_3^{(n)} \end{array}\right] \quad \begin{array}{llll} x_1 & & & = b_1^{(n)} \\ & x_2 & & = b_2^{(n)} \\ & & x_3 & = b_3^{(n)} \end{array}$$

Linear Algebraic Equations

- **가우스 조던 소거법에서의 Flop Count**

$$FLOPS \cong n^3 + O(n^2)$$

-. 곱셈과 나눗셈에서 flops : $n^3 + 3n^2 - 2n \Rightarrow n^3 + O(n^2)$ $as\ n\ increases$!
-. 50% 이상 연산 횟수 증가!

Recall: 가우스 소거법의 FLOPS:

$$FLOPS \cong \frac{2n^3}{3} + O(n^2)$$

따라서, 수행 시간 면에서는 Gauss 소거법이 더 효율적이다.

Linear Algebraic Equations

```
SUB GaussJordan(aug, m, n, x)
DOFOR k = 1, m
  d = aug(k, k)
  DOFOR j = 1, n
    aug(k, j) = aug(k, j)/d
  END DO
  DOFOR i = 1, m
    IF i ≠ k THEN
      d = aug(i, k)
      DOFOR j = k, n
        aug(i, j) = aug(i, j) – d*aug(k, j)
      END DO
    END IF
  END DO
END DO
DOFOR k = 1, m
  x(k) = aug(k, n)
END DO
END GaussJordan
```