연세대학교 통계 데이터 사이언스 학회 ESC 23-2 SUMMER WEEK1

Conditioning and Stability

[ESC 30기 임원진] 박정현 왕재혁 김채현 김시은 김두은

1.Introduction

Introduction

Intro

이 책의 주제 = **수치해석**

1. 수치해석이 뭐지?
   - 과학, 공학 등 여러 분야에서 발생하는 수학적 문제를 해결하기 위한 알고리즘의 설계와 분석에 관한 것 → scientific computing

2. 수치해석을 왜 배울까?
   - 대부분의 문제를 정확히 풀기 어려움 → 충분히 근접한 정답이 나올 때까지 반복하는 과정을 거치며 해를 구해야 한다.
   - **빠르게 수렴하는 반복 알고리즘을 찾고 결과 근사치의 정확도를 평가**하는 것 = 수치해석에서 중요한 주제 !
   - 근사해의 정확도를 추정하고 극한값이 실제 해에 수렴하도록 하는 것 또한 주요 관심사

Introduction

Source of Approximation

Computation이 시작하기 '전' 발생하는 approx

Modeling	일부 물리적 특징의 단순화 or 생략
Empirical measurements	실험 기기의 유한한 accuracy EX) 표본 크기, 랜덤 노이즈, 편향
Previous computations	결과가 근사적인 이전 계산 단계에 의해 생성된 입력 데이터

Computation이 일어나는 '동안' 발생하는 approx

Truncation or discretization	수학적 모델의 일부 특징은 생략 or 단순화 가능 (예: 무한 급수에서 유한한 수의 항만 사용 등)
Rounding	반올림 -> 부정확

예시) 지구

지구의 겉넓이: $4\pi r^2$

여기서 approx는?

[computation 시작 '전']

- 실제 모양을 이상화하여 지구는 구체로 model되었다.
- r은 '약' 6370km -> 경험적 측정과 이전 계산의 조합을 기반

[during computation]

- pi -> 무한한 수이지만, 보통 특정 시점에서 자른다.
- 입력 데이터에 대한 수치, 산술 연산 결과 -> 반올림

따라서 이러한 approx에 따라 계산된 결과의 accuracy는 달라진다.

- accuracy를 결정하는데 있어 중요한 역할을 한다.
- Error analysis: 이러한 approx들이 알고리즘의 accuracy나 stability에 미치는 영향에 대한 연구

Introduction

Absolute Error and Relative Error *Relative error: % 가능

Absolute error = approximate value - true value

Relative error = absolute error / true value

(1) 참 값이 50, approx value가 60이면 absolute error는 10이 된다. 이 경우 상대 오차는 10/50 = 0.2가 된다

(2) 참 값이 9000, approx value가 9010이면 absolute error는 10이 된다. 하지만 이 경우 상대 오차는 10/9000 = 0.00111111111이 된다.

-> 즉 앞의 경우는 20%, 뒤의 경우는 약 0.1%의 상대 오차로 절대 오차와 달리 큰 차이를 보인다.

- precision: 숫자가 표현되는 자릿수
- accuracy: 원하는 것에 근사할 때 정확한 유의 자릿수

ex) 3.2526037646908O4은 매우 precise한 수라고 할 수 있다. 하지만 pi에 대한 근사치로 accurate하냐? → no!

Introduction

Data Error and Computational Error

x : true value of the input, $f(x)$: desired true result

$\hat{x}$: inexact input, $\hat{f}(x)$: approximation to the function

$$\begin{aligned} \text{Total error} &= \hat{f}(\hat{x}) - f(x) \\ &= (\hat{f}(\hat{x}) - f(\hat{x})) + (f(\hat{x}) - f(x)) \\ &= \text{computational error} + \text{propagated data error.} \end{aligned}$$

- 두 번째 줄 식
  - 같은 input에 대한 exact & approximate function의 차 = computational error
  - exact function 값에 대한 차이 ($\hat{x}$ & x) = propagated data error

Introduction

Truncation Error and Rounding Error

Computational Error = Truncation Error + Rounding Error

(1) truncation error: 근사치 계산에서 발생하는 오차

- approximations such as truncating an infinite series
- ex) 테일러 급수: 계산하는 항이 길어질수록 더 정확해질 것이다. (오차는 줄어들 것이다)

(2) rounding error: 1/3 = 0.333333333… → 컴퓨터 계산 시 어느 정도까지만 보는데, 그 때 생기는 오차

→ 이 둘은 tradeoff 관계이다!

: 급수에서 항을 추가하면 truncation error는 줄어들지만, 더 늘어난 급수를 계산하는데 있어 rounding error는 늘어난다.

Introduction

Finite Difference Approximation Example

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \dots$$

→ 테일러 급수 전개

→ h^2이후의 항을 제거하면 다음과 같은 결과가 나온다.

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

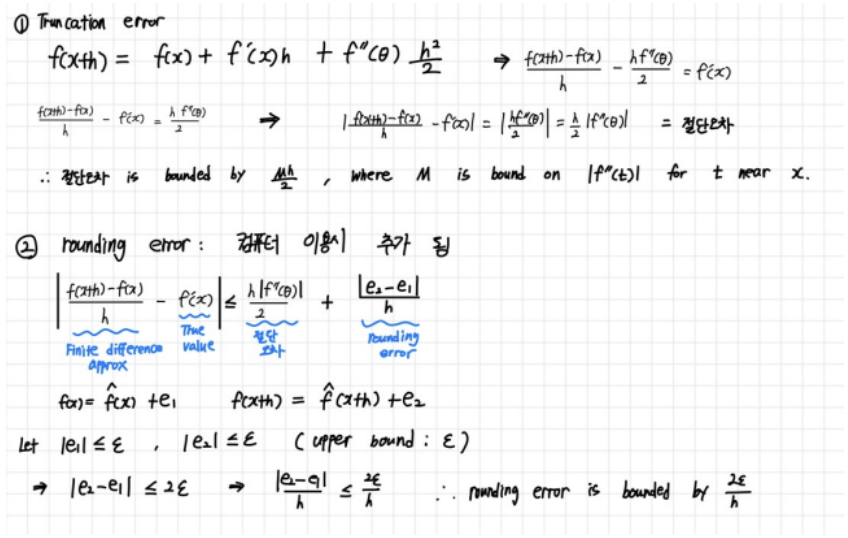
즉, $f'(x)$는 $\frac{f(x+h)-f(x)}{h}$로 근사하여 풀 수 있다.

Introduction

Finite Difference Approximation Example

테일러 급수: $\theta \in [x, x+h]$ (x와 x+h 사이에 세타가 존재)

$$f(x+h) = f(x) + f'(x)h + f''(\theta)h^2/2$$



- truncation error of the finite difference approximation is bounded by $Mh/2$, where $M$is a bound on $|f''(t)|$ for t near x.
  - 절단한 부분이 존재하기 때문에, 절단 오차를 구할 수 있다.
- rounding error is bounded by $2\epsilon/h$
- total computational error: $\frac{Mh}{2} + \frac{2\epsilon}{h}$
  - h가 감소하면 첫 번째 항은 감소하고, 두 번째 항은 증가한다.

    → tradeoff between truncation error and rounding error

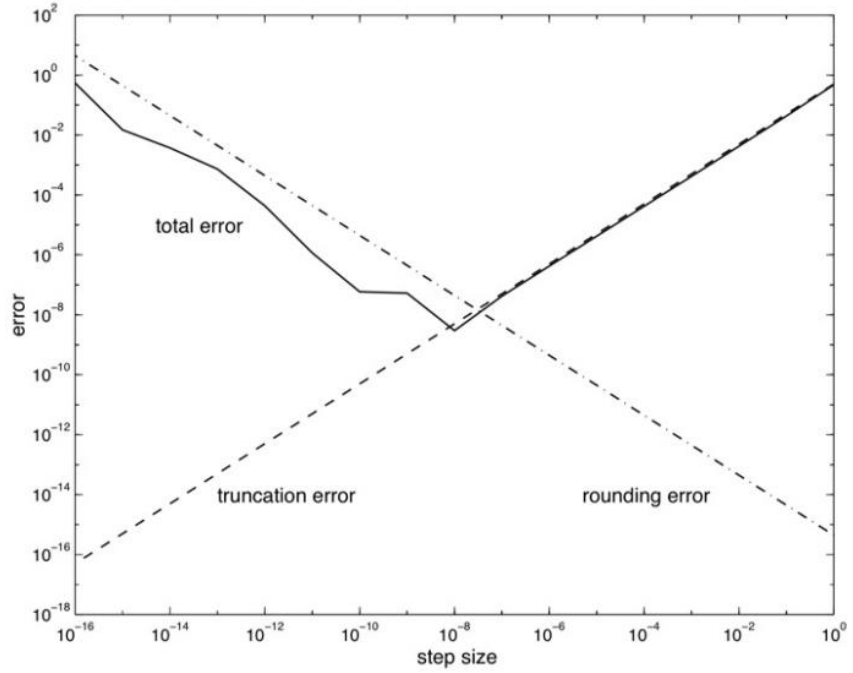
h에 대해서 위에 식을 미분한 다음 0으로 두면,

$h = 2\sqrt{\epsilon/M}$이 된다.

→ total computational error는 h가 다음과 같을 때 minimized된다.

Introduction

Finite Difference Approximation Example



- $f(x) = sin(x)$ at $x = 1$에 해당하는 것
- step size = h
- M = 1
- $\epsilon \approx 10^{-16}$

- total error는 $h \approx 10^{-8} \approx \sqrt{\epsilon}$일 때 최소에 도달한다.
- h가 커지면, truncation error가 커지기 때문에 total error는 커진다.
- h가 작아지면, rounding error가 커지기 때문에 total error는 커진다.

Introduction

Forward and Backward Error

어려운 문제를 같거나 밀접하게 연관된 해를 가지는 더 쉬운 문제로 대체해 계산 문제를 품

(ex) 무한 차원 공간 → 유한 차원 공간, 비선형 → 선형, 미분방정식 → 대수방정식 (실제 얻는 값 : approximate value)

$$y = f(x) \rightarrow \hat{y} = f(\hat{x})$$

$$y = f(x) = cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots \rightarrow \hat{y} = \hat{f}(x) = 1 - \frac{x^2}{2}$$

발생하는 error를 forward , backward error로 나눠서 분석하고자 함!

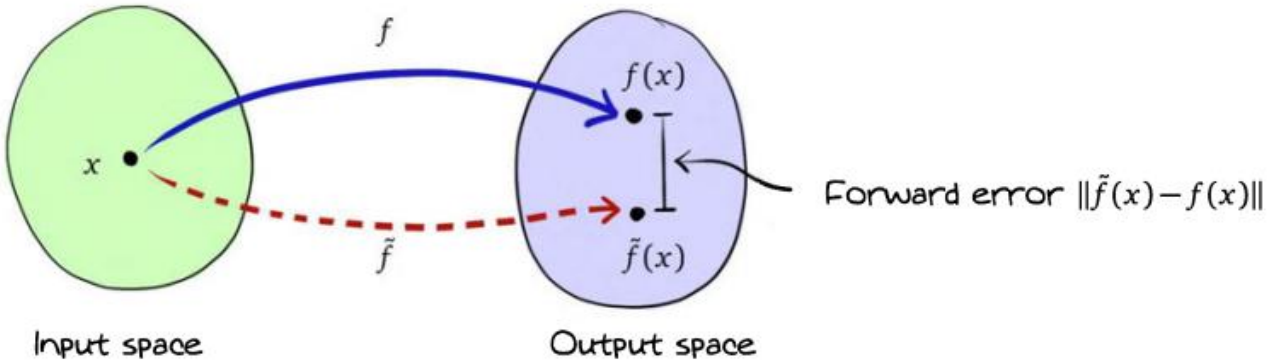
Introduction

Forward Error

$y = f(x)$, where $f : \mathbb{R} \to \mathbb{R}$, but obtain $\hat{y}$

The difference between the computed and true values

$$\Delta y = \hat{y} - y$$

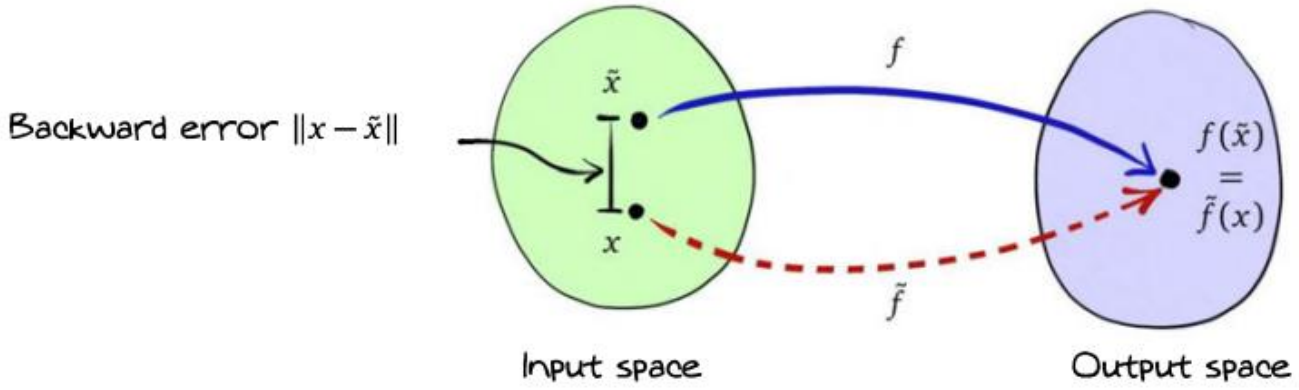


Introduction

Backward Error

The difference between actual input and input for which computed result is exactly correct

$$\Delta x = \hat{x} - x$$



여기서 $\hat{x}$는

$\hat{y} = f(\hat{x})$ 를 만족하는 $\hat{x}$

그래서 **backward** error임.

Introduction

Forward and Backward Error

(ex) $f(x) = \sqrt{x}, x = 2$일때 forward error, backward error 구하기

$y = \sqrt{2} = 1.41421 \rightarrow \hat{y} = 1.4$

$\hat{y} = f(\hat{x})$ 이므로, $1.4 = \sqrt{1.96} \rightarrow \hat{x} = 1.96$

Forward error : $| \Delta y | = | \hat{y} - y | = | 1.4 - 1.41421.. | \approx 0.0142$

Backward error : $| \Delta x | = | \hat{x} - x | = | 1.96 - 2 | = 0.04$

Relative Forward error : $\frac{|\hat{y}-y|}{|y|} \approx \frac{0.0142}{1.41421} \approx 0.01$ (1%)

Relative Backward error : $\frac{|\hat{x}-x|}{|x|} \approx \frac{0.04}{2} \approx 0.02$ (2%)

Introduction

Vector Norm

- 숫자(Scalar) 의 크기 개념을 벡터로 확장한 것
- 실수공간에서 다음의 성질들을 만족함

1. Homogeneity property: 스칼라 a와 벡터 v에 대하여 $\|av\| = \|a\|\|v\|$
2. Triangle inequality: 벡터 u, v에 대하여 $\|u + v\| \leq \|u\| + \|v|$
3. Positive definiteness: 실수 벡터 v에 대하여 $\|v\| \geq 0$

가장 많이 쓰는 것은 p-norm으로, n차원 벡터 x에 대하여 $\|\mathbf{x}\|_p = (\sum_{i=1}^{n} |x_i|^p)^{\frac{1}{p}}$ 로 정의됨

Introduction

Matrix Norm

- Vector norm으로부터 유도된 matrix norm은 I.과 같이 정의되며, 행렬이 벡터의 norm을 최대한으로 늘릴 수 있는 비율을 의미함
- Matrix norm도 vector norm의 성질을 모두 만족하며, 추가적으로 II.를 만족함
- Matrix norm은 III.의 공식을 통해 비교적 쉽게 계산할 수 있음

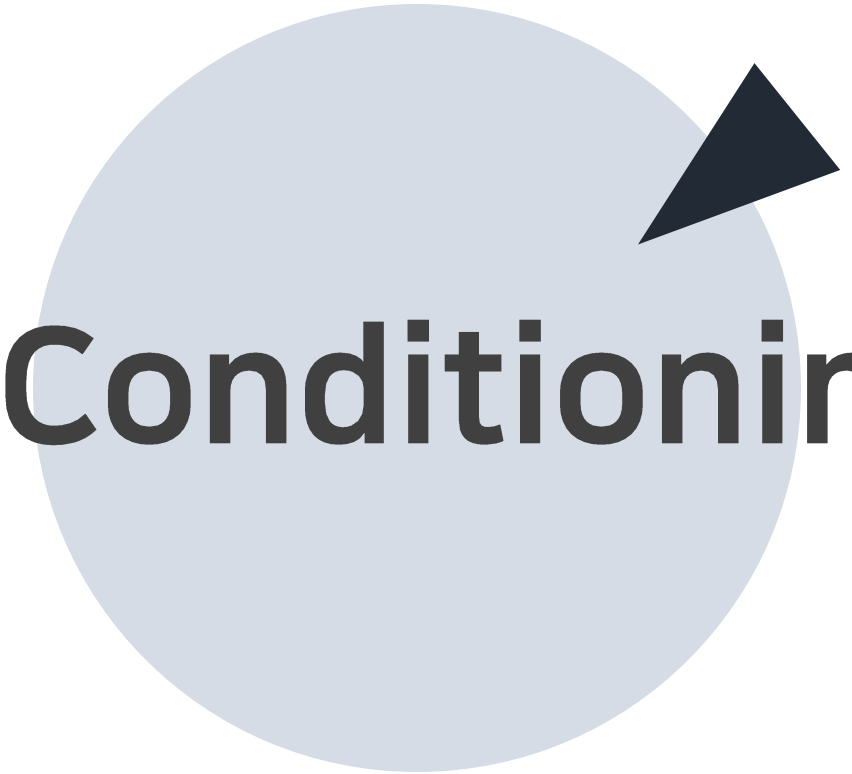
I. Def $\|A\| = \max_{\mathbf{x}\neq 0} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}$

II. Sub-multiplicative $\|A\mathbf{x}\| \leq \|A\| \cdot \|\mathbf{x}\|, \ \|AB\| \leq \|A\| \cdot \|B\|$

III. $\|A\|_1 = \max_j \sum_{i=1}^{n} |a_{ij}|$

$\|A\|_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|$

$\|A\|_2 = \sigma_{\max}$


2.Conditioning

Conditioning

Well Posed Problem

Well Posed Problem은 다음 속성을 만족한 경우!

1. Solution exists [해가 있다]
2. Solution is unique [해가 유일하다]
3. Solution depends continuously on problem data [해가 연속적으로 변화한다]

⇒ 만족 X 경우가 ill posed problem

이때 Well posed problem이어도 data perturbation에 대해 sensitive할 수 있음

- Data perturbation? 관측치의 측정 오차, 데이터 입력 오류 등
- Sensitivity? x가 perturbed 됨에 따라 f(x)가 얼마나 변화하는가?

Conditioning

Sensitivity & Conditioning

Sensitivity = Forward Error / Backward Error = $\|f(x) - \hat{f}(x)\| / \|x - \hat{x}\| = \|f(x) - f(\hat{x})\| / \|x - \hat{x}\|$

Normalize 해주는 의미로 Relative Sensitivity를 계산

Relative Sensitivity = Relative Forward Error / Relative Backward Error = **Conditioning** $\gg$ 1

$$(\|f(x) - \hat{f}(x)\| / \|f(x)\|) / (\|x - \hat{x}\| / \|x\|) = \|f(x) - f(\hat{x})\| \|x\| / \|x - \hat{x}\| \|f(x)\|$$

$\Leftrightarrow$ Relative Forward Error = Conditioning x Relative Backward Error *일종의 amplification factor (진폭 계수) 역할

Conditioning은 정확하게 값이 알려져 있지 않거나 input에 따라 변화하는 경우가 대부분이므로,

Condition number에 대한 rough estimate 혹은 upper bound가 사용됨

Conditioning

Condition number Example *Condition Number가 함수 지정에 따라 변화

- Evaluating function f for approximate input $\hat{x} = x + \Delta x$ instead of true input x gives

  Absolute forward error: $f(x + \Delta x) - f(x) \approx f'(x)\Delta x$

  Relative forward error: $\dfrac{f(x + \Delta x) - f(x)}{f(x)} \approx \dfrac{f'(x)\Delta x}{f(x)}$

  Condition number: $\text{cond} \approx \left| \dfrac{f'(x)\Delta x / f(x)}{\Delta x / x} \right| = \left| \dfrac{x\, f'(x)}{f(x)} \right|$

- Consider $f(x) = \sqrt{x}$
- Since $f'(x) = 1/(2\sqrt{x})$,

$$\text{cond} \approx \left| \frac{x\, f'(x)}{f(x)} \right| = \left| \frac{x/(2\sqrt{x})}{\sqrt{x}} \right| = \frac{1}{2}$$

- So forward error is about half backward error, consistent with our previous example with $\sqrt{2}$
- Similarly, for $f(x) = x^2$,

$$\text{cond} \approx \left| \frac{x\, f'(x)}{f(x)} \right| = \left| \frac{x\,(2x)}{x^2} \right| = 2$$

which is reciprocal of that for square root, as expected

Conditioning

Condition number of matrix

- Square nonsingular matrix의 condition number는 I.과 같이 정의됨
- 이는 주어진 벡터에 대한 maximum relative stretching과 maximum relative shrinking의 비율을 의미함
- Matrix norm은 쉽게 계산할 수 있지만 inverse의 경우 계산 비용이 많이 들기 때문에, 보통 선택한 벡터 $\boldsymbol{y}$에 대하여 $A\mathbf{z} = \mathbf{y} \rightarrow \frac{\|\mathbf{z}\|}{\|\mathbf{y}\|} \leq \|A^{-1}\|$ 임을 이용해 추정함
- Non-square matrix의 경우 full rank인 경우 II., 아닌 경우 III.과 같이 pseudoinverse를 이용해 condition number를 계산할 수 있음

I. $\text{cond}(A) = \|A\| \cdot \|A^{-1}\| = \left(\max_{\mathbf{x}\neq 0} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}\right) \cdot \left(\min_{\mathbf{x}\neq 0} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}\right)^{-1}$

II. $\text{cond}(A) = \|A\|_2 \cdot \|A^+\|_2, \text{ where } A^+ = (A^\top A)^{-1} A^\top$

III. $\text{cond}_2(A) = \frac{\sigma_{\max}}{\sigma_{\min}}, \text{ where } A^+ = V\Sigma^+ U^\top$

Conditioning

Conditioning of solving linear system

- 주어진 데이터에 perturbation이 있을 때, 즉 관측치의 측정 오차, 데이터 입력 오류, 숫자 반올림 등으로 실제 값과 차이가 생겼을 때 condition number를 통해 solution이 실제 값과 얼마나 차이가 생길지 upper bound를 추정할 수 있음

Ex1)

Ax = b를 풀 때, 데이터 중 벡터 b에 변화가 생겼다고 하면

$A\hat{\mathbf{x}} = A(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{b} + \Delta\mathbf{b}$ 이고,

$\|\Delta\mathbf{x}\| = \|A^{-1}\Delta\mathbf{b}\| \leq \|A^{-1}\| \cdot \|\Delta\mathbf{b}\|$, 그리고

$\|\mathbf{b}\| = \|A\mathbf{x}\| \leq \|A\| \cdot \|\mathbf{x}\|$ 이므로

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \mathrm{cond}(A) \cdot \frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|}$$

Ex2)

Ax = b를 풀 때, 데이터 중 A에 변화가 생겼다고 하면

$(A + E)\hat{\mathbf{x}} = (A + E)(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{b}$

$\Delta\mathbf{x} = \hat{\mathbf{x}} - \mathbf{x} = A^{-1}(A\hat{\mathbf{x}} - \mathbf{b}) = -A^{-1}E\hat{\mathbf{x}}$

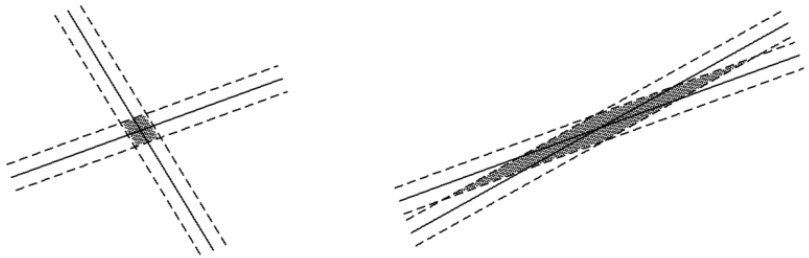
즉 $\|\Delta\mathbf{x}\| \leq \|A^{-1}\| \cdot \|E\| \cdot \|\hat{\mathbf{x}}\|$ 이므로

$$\frac{\|\Delta\mathbf{x}\|}{\|\hat{\mathbf{x}}\|} \leq \mathrm{cond}(A) \cdot \frac{\|E\|}{\|A\|}$$

Conditioning

Conditioning of solving linear system

- 즉 linear system에서 데이터 오차에 따른 솔루션의 오차는 행렬 A의 condition number로 upper bounded 되어 있음



well-conditioned　　ill-conditioned

- 평면에서 생각했을 때, 두 직선이 거의 수직일 경우 직선을 점선만큼 이동해도 솔루션이 크게 변하지 않음. 일반적으로, orthogonal matrix의 2-norm으로부터 유도된 condition number는 1
- 반대로 두 직선이 거의 평행할 경우, 일반적으로 singular에 가까울 경우 똑같은 범위만큼 직선이 이동했을 때 솔루션이 매우 크게 변할 수 있음

Conditioning

Conditioning of linear least squares problem (Numeric Example)

- 아래와 같이 데이터가 주어져 있고, 이 문제에 대한 least squares solution을 구한다고 하면 조금의 perturbation에도 솔루션이 매우 크게 변함

$$A = \begin{bmatrix} 0.913 & 0.659 \\ 0.780 & 0.563 \\ 0.457 & 0.330 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1.572 \\ 1.343 \\ 0.787 \end{bmatrix}, \quad \tilde{\mathbf{b}} = \begin{bmatrix} 1.57 \\ 1.34 \\ 0.78 \end{bmatrix}$$

```
> A = matrix(c(0.913, 0.780, 0.457, 0.659, 0.563, 0.330), nrow=3)
> b = c(1.572, 1.343, 0.787)
> btilde = c(1.57, 1.34, 0.78)
> lm(b~A-1)

Call:
lm(formula = b ~ A - 1)

Coefficients:
A1  A2
 1   1
```

```
> lm(btilde~A-1)

Call:
lm(formula = btilde ~ A - 1)

Coefficients:
    A1      A2
 30.38  -39.70
```

Conditioning

Conditioning of linear least squares problem (Numeric Example)

- 이는 이 행렬이 nearly rank deficient matrix이기 때문으로, 따라서 condition number가 매우 큰 것을 확인할 수 있음

```
> det(t(A) %*% A)
[1] 2.8011e-08
```

```
> kappa(A)
[1] 16982.46
```

- 따라서 특잇값 분해를 했을 때 두 번째 특잇값이 거의 0에 가까움. 따라서 그것을 0으로 취급하여 (= 일부 정보를 버려서) minimum norm solution을 구하면 훨씬 stable한 결과를 얻을 수 있음

```
> svd(A, nu =3, nv = 2)
$d
[1] 1.5846034169 0.0001056194

$u
           [,1]       [,2]       [,3]
[1,] -0.7105806 -0.2663089 -0.65127168
[2,] -0.6070674 -0.2359231  0.75882113
[3,] -0.3557308  0.9345694  0.00597497

$v
           [,1]       [,2]
[1,] -0.8108285 -0.5852838
[2,] -0.5852838  0.8108285
```

```
> pp %*% b
           [,1]
[1,] 1.1320077
[2,] 0.8171219
> pp %*% btilde
           [,1]
[1,] 1.1290744
[2,] 0.8150046
```

3.Stability & Complexity

Stability and Complexity

Machine Precision

> 💡 컴퓨터는 유한개의 bit로 수를 인식하기 때문에 오차가 발생한다!

- ϵ_{mach} : 1과 1과 구별될 수 있는 1보다 큰 다음 수 사이의 간격 (컴퓨터가 인식하는 최솟값)
- 두 값의 차이가 ϵ_{mach} 보다 작거나 같다면 두 값은 같은 값으로 정의된다.

ex. 다음 행렬이 주어져 있고, 이 행렬의 LU 분해를 계산하려 한다.

$$A = \begin{bmatrix} 1 & 1+\epsilon \\ 1-\epsilon & 1 \end{bmatrix}$$

이 행렬의 행렬식은 ϵ^2이다. 따라서 $\epsilon < \sqrt{\epsilon_{mach}}$이라면 이 행렬은 컴퓨터에서 singular로 인식된다.

A의 LU 분해는 다음과 같다.

$$A = \begin{bmatrix} 1 & 0 \\ 1-\epsilon & 1 \end{bmatrix} \begin{bmatrix} 1 & 1+\epsilon \\ 0 & \epsilon^2 \end{bmatrix}$$

이때 $\epsilon < \sqrt{\epsilon_{mach}}$라면 이 행렬은 컴퓨터에서 singular로 인식된다.

ex. 다음 행렬에 대한 least squares 문제를 풀려고 한다.

$$A = \begin{bmatrix} 1 & 1 \\ \epsilon & 0 \\ 0 & \epsilon \end{bmatrix}$$

이때 $A^\top A = \begin{bmatrix} 1+\epsilon^2 & 1 \\ 1 & 1+\epsilon^2 \end{bmatrix}$이므로, $\epsilon < \sqrt{\epsilon_{mach}}$라면 이 행렬은 컴퓨터에서 rank deficient로 인식된다.

Stability and Complexity

Meaning of $O(\epsilon_{mach})$

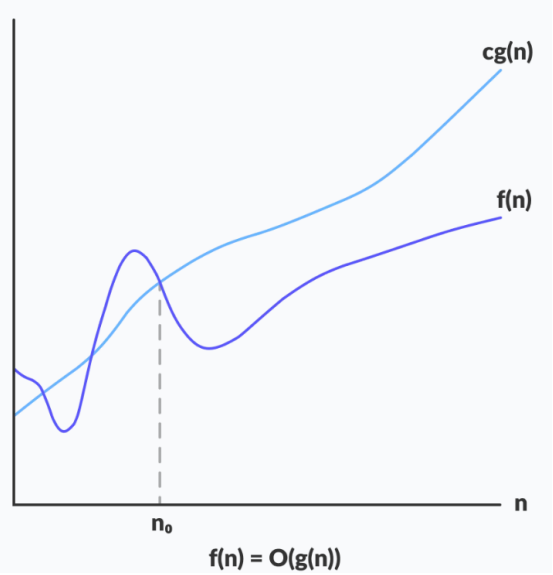
Big-O notation

> 모든 $t > t_0$ 에 대하여 $|f(t)| \leq Cg(t)$ 를 만족하는 상수 $C > 0$ 와 t_0 가 존재할 때 $f(t) = O(g(t))$이다.

※ O 안에 들아가는 항에 곱해진 상수는 제외하고 영향이 가장 큰 항을 주로 표기한다.

ex. $n^2 + 2n = O(n^2)$ as $n \to \infty$

$n > 2$ 인 n 에 대하여 $|n^2 + 2n| \leq 2n^2$ 이므로 다음과 같이 표현할 수 있다.



Stability and Complexity

$O(\epsilon_{mach})$

$$||\text{computed quantity}|| = O(\epsilon_{mach})$$

1. computed quantity 는 알고리즘 $\tilde{f}$ 를 통해 얻은 값으로, f 라는 문제의 data $x \in X$ 와 ϵ_{mach} 에 대한 값이다.
2. $\epsilon_{mach} \to 0$
3. 모든 data $x \in X$ 에 uniform 하게 적용된다.

ϵ_{mach} 은 고정된 값이지만 ideal computers 를 생각했을 때 $\epsilon_{mach} \to 0$ 로 표현할 수 있고, 감소하는 ϵ_{mach} 의 값에 computed quantity 가 uniformly bounded 된다고 생각하면 된다.

Stability and Complexity

Stability

- Problem $\rightarrow$ ***Conditioning***
- Algorithm $\rightarrow$ ***Stability***

> $\dfrac{||\tilde{x} - x||}{||x||} = O(\epsilon_{mach})$ 을 만족하는 모든 $x \in X$ 에 대하여
> $\dfrac{||\tilde{f}(x) - f(\tilde{x})||}{||f(\tilde{x})||} = O(\epsilon_{mach})$ 이면 $\tilde{f}$ 는 f 에 대한 *stable* algorithm이다.

Stability and Complexity

Backward Stability

앞으로 우리가 배울 대부분의 알고리즘은 stability 보다 더 엄밀하고 간단한 backward stability 라는 조건을 만족한다.

> $\frac{||\tilde{x} - x||}{||x||} = O(\epsilon_{mach})$ 을 만족하는 모든 $x \in X$ 에 대하여 $\tilde{f}(x) = f(\tilde{x})$ 이면 $\tilde{f}$ 는 f 에 대한 *backward stable* algorithm이다.

※ LU factorization without pivoting 알고리즘은 not backward stable 하다!

Stability and Complexity

Accuracy

Well-conditioned 한 문제 + Stable 한 algorithm = Accurate!

> 모든 $x \in X$ 에 대하여 $\frac{||\tilde{f}(x) - f(x)||}{||f(x)||} = O(\epsilon_{mach})$ 을 만족하면 $\tilde{f}$ 는 f 에 대한 accurate algorithm이다. 물론 f 는 well-conditioned 이어야 한다.

Stability and Complexity

Complexity of solving linear systems

> 💡 Linear systems 을 푸는 다양한 알고리즘에 대해 배우는 이유!

$Ax = b$ 라는 linear equation을 풀 때 A^{-1}를 직접 구하는 방법은 매우 비효율적이다.

1. A가 $n \times n$ 행렬일 때 Gaussian elimination으로 LU factorization을 한 후 linear system 을 푸는 방법은 LU factorization을 하는데 약 $n^3/3$ 이 곱셈 연산이, forward, backward-substitution 에 각각 $n^2/2$ 의 곱셈 연산이 필요하다.
   반면에 역행렬을 구해 푸는 방법은 약 n^3 의 연산을 필요로 한다.
2. Matrix inversion 으로 linear systems 을 푸는 것은 훨씬 많은 계산을 요하기도 하지만 부정확한 결과를 얻기도 한다.

ex. $3x = 18$ 이라는 식을 풀면 $18/3 = 6$이라는 결과가 나와야 하지만 역수를 곱해 구한다면 $x = 3^{-1} \times 18 = 0.333\ldots \times 18 = 5.999\ldots$ 이라는 부정확한 결과를 얻는다.

END