[2026-1] Computational Statistics

Jiho Kwak

June 2026

1. Computer Arithmetic

1.1. Units of Computer Storage

- bit = binary digit
- byte(B) = 8 bits

```
library(lobstr)
x <- 100
y <- c(20, 30)
sapply(ls(), function(z, env=parent.env(environment())) obj_size(get(z,
envir=env)))
```

```
 x  y
56 64
```

- 고정 헤더 크기(48) + 8 * 원소 개수

1.2. Storage of Characters

- ASCII: 7 bits -> $2^7 = 128$ characters
- Extended ASCII: 8 bits -> $2^8 = 256$ characters
- Unicode: UTF-8, UTF-16, UTF-32 -> last 7 digits conform to ASCII
  - UTF-8: current dominant character encoding

1.3. Integers: Fixed-Point Number System

- fixed-point number system: computer model for integers
- any representation of numbers in computer has to be an approximation
  - computer memory (finite) vs. cardinality of $\mathbb{Z}$ (infinite)
- number M of bits and method of representing negative numbers 다양
  - R: `integer` type $M = 32$
  - C: `char`, `int`, `short`, `long`, ⋯
  - Matlab: `(u)int8`, ⋯, `(u)int64`

Unsigned Integers

- 음이 아닌 정수
- range: $[0, 2^M - 1]$

Signed Integers

- model of $\mathbb{Z}$, 뺄셈에 대해 닫혀 있음
- 첫 번째 비트(=MSB)는 sign bit
  - 0: 음이 아닌 정수
  - 1: 음수
- two's complement representation for negative numbers
  - sign bit = 1
  - negate the remaining bits (0 → 1, 1 → 0)
  - add 1 to the result
- unsigned integer $2^M - x$와 음수 x의 표현 동일 (mod 2^M)

```
pryr:: bits(5L)
```

```
[1] "00000000 00000000 00000000 00000101"
```

```
pryr:: bits(-5L)
```

```
[1] "11111111 11111111 11111111 11111011"
```

```
pryr::bits(2L * 5L) # shift bits of 5 to the left (why?)
```

```
[1] "00000000 00000000 00000000 00001010"
```

```
pryr::bits(2L * -5L); # shift bits of -5 to left
```

```
[1] "11111111 11111111 11111111 11110110"
```

- modular arithmetic에 잘 대응됨
  - 덧셈/뺄셈: 그냥 이진 덧셈 후 M비트로 잘라냄 (+ overflow 참조)
- range (M-bit signed integer): $[-2^{M-1}, 2^{M-1} - 1$

```
pryr::bits(2147483647L)
```

```
[1] "01111111 11111111 11111111 11111111"
```

Overflow and Underflow

- R은 integer overflow/udnerflow에 대해 NA를 출력

```
.Machine$integer.max + 1L
```

```
Warning in .Machine$integer.max + 1L: NAs produced by integer overflow
```

```
[1] NA
```

1.4. Real Numbers: Floating-Point Number System

- computer model for the real line $\mathbb{R}$
- 대부분의 컴퓨터 시스템은 IEEE 754 standard 채택
- scientific notation: (b = base, e.g. 10진법, 2진법, ⋯)

$$\pm d_1.d_2d_3...d_p \times b^e, \quad 0 \leq d_i < b$$

- normalized vs. denormalized
  - $18 = +1.0010 \times 2^4$ (normalized) → 첫 자리는 무조건 1
  - $18 = +0.1001 \times 2^5$ (denormalized)
- 부동소수점 체계에서 컴퓨터는 다음을 저장
  - sign bit
  - fraction (mantissa, significand) of the normalized representation (유효숫자)
  - actual exponent plus a bias
- 체계별 비교
  - R: `double` (double precision)
  - C: `float`는 single precision, `double`은 double precision
  - Julia: `Float16` (half), `Float32` (single), `Float64` (double)
  - Python: `double`

Half Precision

- sign(1) + exponent (5) + fraction (10)
- 10 significand bits → $p = 11$
- 5 exponent bits: $e_{\max} = 15, e_{\min} = -14$, bias = 15 = 01111_2
  - $e_{\min} = 00001_2 = 01111_2 = -14$
  - $e_{\max} = 11110_2 = 01111_2 = +15$
- range of magnitude: $10^{\pm 4}$ ($\log_{10}(2^{15}) \simeq 4$)
- precision: $\log_{10} 2^{11} \simeq 3.311$
- value: (첫째 자리가 b_{15})

$$(value) = (-1)^{b_{15}} \times 2^{\left(\sum_{j=1}^{5} b_{15-j} 2^{5-j}\right)-15} \times \left(1 + \sum_{i=1}^{10} \frac{b_{10-i}}{2^i}\right)$$

```
# bitstring(Float16(5))  = "0100010100000000"
# bitstring(Float16(-5)) = "1100010100000000"
```

Single Precision

- sign(1) + exponent (8) + fraction (23)
- 23 significand bits → $p = 24$
- 8 exponent bits: $e_{\max} = 127, e_{\min} = -126$, bias = 127
- range of magnitude: $10^{\pm 38}$ ($\log_{10}(2^{127}) \simeq 38$)
- precision: $\log_{10} 2^{24} \simeq 7.225$

Double Precision

- sign(1) + exponent (11) + fraction (52)
- 52 significand bits → $p = 53$
- 11 exponent bits: $e_{\max} = 1023, e_{\min} = -1022$, bias = 1023
- range of magnitude: $10^{\pm 308}$ ($\log_{10}(2^{1023}) \simeq 308$)
- precision: $\log_{10} 2^{53} \simeq 15.95$
- value: (첫째 자리가 b_{15})

Special Floating-Point Numbers

- infinity($\pm\infty$): exponent $e_{\max} + 1$ + mantissa 0

```
pryr::bits(Inf); pryr::bits(-Inf)
```

```
[1] "01111111 11110000 00000000 00000000 00000000 00000000 00000000 00000000"
```

```
[1] "11111111 11110000 00000000 00000000 00000000 00000000 00000000 00000000"
```

- not a number (NaN): exponent $e_{\max} + 1$ + mantissa nonzero
  - in general `NaN != NaN` bitwise → test by `is.nan()`

```
pryr::bits(0 / 0)
```

```
[1] "01111111 11111000 00000000 00000000 00000000 00000000 00000000 00000000"
```

- exact zero: exponent $e_{\min} - 1$ + mantissa 0

```
pryr::bits(0.0)
```

```
[1] "00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000"
```

- denormalized numbers: exponent $e_{\min} - 1$ + mantissa nonzero
  - numbers less than $b^{e_{\min}}$
  - numbers are denormalized in the range $(0, b^{e_{\min}})$ → gradual underflow
  - e.g. in half-precision, $e_{\min} = -14$, but $2^{-24} = 0.0000000001_2 \times 2^{-14}$
  - without gradual underflow: `x != y`인데도 `x - y == 0`일 수 있음

Rounding

- p 이상의 significand bits 가진 경우 필요
- most computer system: IEEE 754 round to nearest, ties to even
- round to nearest: $0.1 = 1.10011001\ldots_2 \times 2^{-4}$
  - 1.100110011001⋯ → 100 과 101 사이는 1001 0000⋯ 이므로 101에 더 가까움
- ties to even: midway이면 even least significant digit로
  - precision이 4인 경우: 1.011100×2^1 → `1.011 100`은 `1.011`과 `1.100` 사이, round up

Errors

- real number x가 floating point number $[x]$로 표현되는 경우
  - absolute error

    $$|[x] - x|$$

  - relative error

    $$\frac{|[x] - x|}{|x|}$$

Machine Epsilons

- floating-point numbers는 uniform하게 분포하지 않음 (단 $(2^i, 2^{i+1}]$과 같은 구간 내에서는 uniform하게 분포)
- machine epsilon: 1 주위 숫자의 spacings
  - $\epsilon_{\max}$ = (1에 더했을 때 1이 아니게 하는 가장 작은 수) = $\frac{1}{2^p} + \frac{1}{2^{2p-1}}$
  - $\epsilon_{\min}$ = (1에서 뺐을 때 1이 아니게 하는 가장 작은 수) = $\frac{1}{2^{p+1}} + \frac{1}{2^{2p}}$
- $\left[1 - \frac{1}{2^{p+1}}, 1 + \frac{1}{2^p}\right]$ 사이의 수는 1로 표현

Machine Precision

- relative error는 $\epsilon_{\max}$에 의해 bounded 됨
- `double.neg.eps`: $\epsilon_{\max}$

1.5. Floating Point Arithmetic

- addition of two numbers x and y
  - $z = [x] + [y]$
  - $z \leftarrow [z]$

Overflow and Underflow of Floating-Point Number

- overflow: produces $\pm\infty$
- underflow: zero or denormalized numbers
- underflow 가 더 선호됨
  - logit link: $p = \frac{\exp(x^T\beta)}{1+\exp(x^T\beta)} = \frac{1}{1+\exp(-x^T\beta)}$
  - 전자는 `Inf / Inf = NaN`, 후자는 gradual underflow
- `.Machine$double.xmax`, `.Machine$double.xmin` → largest/smallest non-subnormal numbers

Lost Significant Digits

- magnitude 차이가 큰 두 숫자의 덧셈/뺄셈 → 작은 숫자의 precision 잃어버림
  - $a >> b, \quad a + b = a$

Catastrophic Cancellation

- 거의 같은 두 숫자의 뺄셈 → 모든 유효숫자 잃어버리는 현상
- operand의 rounding error에 의해 발생

Implications

- 큰 수와의 덧셈 전에 작은 수들끼리 더하기
- 비슷한 크기의 숫자들 끼리 pair로 더하기
- 거의 동일한 수끼리의 뺄셈 피하기

2. Solving Linear Systems of Equations

- $\mathbf{Ax} = \mathbf{b}$ 풀기 위한 컴퓨터 알고리즘

2.1. Algorithms

Measure of Algorithm Efficiency

- flop (floating point operation): 알고리즘 효율성의 단위, floating point 연산 횟수
- Big-O notation
  - `A * b` ($m \times n$ 행렬과 $n \times 1$ 벡터) → $2mn = O(mn)$ flops
  - `A * b` ($m \times n$ 행렬과 $n \times p$ 행렬) → $2mnp = O(mnp)$ flops
- $O(n \log n)$ → fast < $O(n)$ < $O(\log n)$

2.2. Triangular Systems

Lower/Upper Triangular System

- $\mathbf{Ax} = \mathbf{b}$를 풀 때 $\mathbf{A} \in \mathbb{R}^{n\times n}$ 이
  - lower traingular 이면: forward substitution → $x_1 = b_1/a_{11}$ 부터 차례로
  - upper triangular 이면: back substitution (backsolve) → $x_n = b_n/a_{nn}$ 부터 차례로
- 모두 n^2 flops

Implementation

- BLAS triangular system → 'done in place' i.e., $\mathbf{b}$ is overwritten by $\mathbf{x}$

```
    # forward substitution
    for (i in seq_len(n)) {
        for (j in seq_len(i-1)) {
            b[i] <- b[i] - A[i, j] * b[j]
        }
        b[i] <- b[i] / A[i, i]
    }
    # backsolve
    for (i in rev(seq_len(n))) {
        for (j in i + seq_len(max(n - i, 0))) {
            b[i] <- b[i] - A[i, j] * b[j]
        }
        b[i] <- b[i] / A[i, i]
    }
```

Algebraic Facts

- 삼각행렬 $\mathbf{A}$의 고유치 λ_i는 대각 원소 a_{ii}
- $\det(\mathbf{A}) = \prod_i a_{ii}$
- (단위) 상삼각행렬의 곱은 (단위) 상삼각행렬
- (단위) 상삼각행렬의 역행렬은 (단위) 상삼각행렬

2.3. Gaussian Elimination and LU Decomposition

- Gaussian Elimination의 결과 → $\mathbf{A} = \mathbf{L_1} \cdots \mathbf{L_m} \mathbf{U} = \mathbf{L}\mathbf{U}$

Elementary Operator Matrix

- $\mathbf{E}_{jk}(c) = \mathbf{I} + c\mathbf{e}_j\mathbf{e}_k^\top$: identity matrix의 (j, k) 원소를 c로 대체한 것
  - unit triangular, full rank
  - $\mathbf{E}_{jk}^{-1}(c) = \mathbf{E}_{jk}(-c)$
  - $n \times m$ 행렬 $\mathbf{X}$의 j번째 열 $\mathbf{x}_j$를 $c\mathbf{x}_k + \mathbf{x}_j$로 대체 ($2m$ flops)
  - e.g. $\mathbf{L_1}^{-1} = \mathbf{E}_{21}(-2)$
- Gaussian elimination은 $\mathbf{A}\mathbf{x} = \mathbf{b}$에 elementary operator matrix 반복 적용하는 것

Gauss Transformations

- column 1에 대해

$$\mathbf{M}_1 = \mathbf{E}_{n1}(c_{n1}) \cdots \mathbf{E}_{31}(c_{31})\mathbf{E}_{21}(c_{21}) = \begin{bmatrix} 1 & & \\ c_{21} & \ddots & \\ c_{n1} & & 1 \end{bmatrix}$$

- column k에 대해

$$\mathbf{M}_k = \mathbf{E}_{nk}(c_{nk})\cdots\mathbf{E}_{k+1,k}(c_{k+1,k}) = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & c_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & c_{n,k} & & 1 \end{bmatrix}.$$

- $\mathbf{M}_1, ..., \mathbf{M}_{n-1}$: Gauss transformations

LU Decomposition

- Gaussian elimination의 결과:

$$\mathbf{M}_{n-1}\cdots\mathbf{M}_1\mathbf{A} = \mathbf{U}.$$

- let
- thus we have the LU decompositon

$$\mathbf{A} = \mathbf{L}\mathbf{U},$$

  - $\mathbf{L}$: 단위 하삼각
  - $\mathbf{U}$: 상삼각

- LU 분해는 sub-matrix $\mathbf{A}[1:k, 1:k]$가 non-singular for $k = 1, ..., n-1$일 때 존재
- LU 분해가 존재하고 $\mathbf{A}$가 non-singular이면, LU 분해는 유일하고

$$\det(\mathbf{A}) = \det(\mathbf{L})\det(\mathbf{U}) = \prod_{k=1}^{n} u_{kk}.$$

  - R에서 determinant 계산하는 원리

- LU 알고리즘은 done in place: $\mathbf{A}$가 overwrite 됨

```
for (k in seq_len(n-1)) {
    for (i in k +seq_len(max(n - k, 0))) {
        A[i, k] <- A[i, k] / A[k, k]   # computes L
        for (j in k +seq_len(max(n - k, 0))) {
            A[i, j] <- A[i, j] - A[i, k] * A[k, j]  # computes U
        }
    }
}
```

- 계산 복잡도:

$$2(n-1)^2 + 2(n-2)^2 + \cdots + 2\cdot 1^2 \approx \frac{2}{3}n^3 \quad \text{flops}\ .$$

- LU 분해가 되면 $(\mathbf{L}\mathbf{U})\mathbf{x} = \mathbf{b}$ 계산에는 $2n^2$ flops

- One forward substitution (n^2 flops) to solve

$$\mathbf{L}\mathbf{y} = \mathbf{b}$$

- One back substitution (n^2 flops) to solve

$$\mathbf{U}\mathbf{x} = \mathbf{y}$$

Matrix Inversion

- multiple RHS → LU 분해는 한 번만
- 역행렬 $\Leftrightarrow \mathbf{A}\mathbf{X} = \mathbf{I}$인 $\mathbf{X}$ 찾기
  - n개의 RHS: $\mathbf{e}_1, ..., \mathbf{e}_n$ → 0 있으므로 $\frac{4}{3}n^3$ flops
  - 총:

$$\frac{2}{3}n^3 + \frac{4}{3}n^3 = 2n^3 \quad \text{flops} .$$

- 다음의 경우 제외하고 역행렬을 직접 계산하지 않음 (연립일차방정식의 풀이로 이해)
  - standard error 계산 필요
  - RHS의 개수가 n보다 매우 클 때
  - n이 작을 때

Pivoting

- 다음과 같은 경우 LU 분해 사용할 수 없음

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

  - pivot a_{kk}가 0이거나 0에 매우 가까운 경우
- e.g.

$$\begin{aligned} 0.001x_1 + x_2 &= 1 \\ x_1 + x_2 &= 2 \end{aligned}$$

  with solution

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1.001001 \\ 0.998999 \end{bmatrix} \approx \begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}.$$

- with two significant digits, GE via LU yields

$$\begin{aligned} 0.001x_1 + x_2 &= 1 \\ (1 - 1000)x_2 &= 2 - 1000 \end{aligned}$$

  or

$$\begin{aligned} 0.001x_1 + x_2 &= 1 \\ -1000x_2 &= -1000, \end{aligned}$$

yielding

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix}!$$

Condition Number and Stability

- $\mathbf{Ax} = \mathbf{b}$를 푸는 것을 다음의 사상으로 이해 $f : \mathbb{R}^{n \times n} \ni \mathbf{A} \mapsto \mathbf{A}^{-1}\mathbf{b} \in \mathbb{R}^n$
- ill-conditioned problem: $\mathbf{A}$의 작은 변화가 $f(\mathbf{A})$ 크게 변화시키는 경우
- condition number로 측정

$$\kappa(\mathbf{A}) = \sigma_{\max}(\mathbf{A})/\sigma_{\min}(\mathbf{A})$$

  - $\sigma_{\max}(\mathbf{A}) = \max_{\mathbf{x} \neq 0} \parallel \mathbf{Ax} \parallel / \parallel \mathbf{x} \parallel$ is the maximum singular value of $\mathbf{A}$;
  - $\sigma_{\min}(\mathbf{A}) = \min_{\mathbf{x} \neq 0} \parallel \mathbf{Ax} \parallel / \parallel \mathbf{x} \parallel$ is the minimum singular value of $\mathbf{A}$.
- $\mathbf{A}$가 singuar → $\sigma_{\min} = 0$, $\kappa(\mathbf{A}) = \infty$
- 알고리즘의 stability: 알고리즘 $\tilde{f}$에 의해 계산한 $\hat{x} = \tilde{f}(\mathbf{A})$라고 하면, 알고리즘은 stable if the relative error between $\hat{\mathbf{x}}$ and $\tilde{\mathbf{A}}^{-1}\mathbf{b}$ is bounded by $O(\epsilon)$.

Analysis of the Example

- 위의 행렬 $\mathbf{A}$의 condition number는 2.262로 풀기 어렵지 않으나, GE/LU 알고리즘의 instability에 의해 true solution과 computed solution 간의 차이가 크게 발생
- 발생 지점: $a_{21}/a_{11} = 1000$

Solution: Fix the Algorithm by Pivoting

- pivoting: elimination의 순서 변경

$$\begin{aligned} x_1 + x_2 &= 2 \\ 0.001x_1 + x_2 &= 1 \end{aligned}$$

- With two significant digits, GE yields

$$\begin{aligned} x_1 + x_2 &= 2 \\ (1 - 0.001)x_2 &= 1 - 0.002 \end{aligned}$$

  or

$$\begin{aligned} x_1 + x_2 &= 2 \\ 1.0x_2 &= 1.0, \end{aligned}$$

  yielding

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}.$$

- partial pivoting (GEPP): k번째 열 0으로 만들기 전 $\max_{i=k}^{n} |a_{ik}|$인 행을 k번째 행으로 교체

- ▸ 행별 비교에 드는 cost: $O(n^2)$

- LU with partial pivoting yields

$$\mathbf{PA} = \mathbf{LU},$$

  where $\mathbf{P}$ is a permutation matrix, $\mathbf{L}$ is unit lower triangular with $|\ell_{ij}| \leq 1$, and $\mathbf{U}$ is upper triangular.

- Complete pivoting (GECP): do both row and column interchanges so that the largest entry in the sub matrix `A[k:n, k:n]` is permuted to the (k, k)-th entry. This yields the decomposition

$$\mathbf{PAQ} = \mathbf{LU},$$

  where $|\ell_{ij}| \leq 1$.

- GECP가 더 stable 하지만 costs more computation
- partial pivoting은 대부분 stable

Implementation

- BLAS and LAPACK 라이브러리: GETRF (partial pivoting)
- R: `solve()`

```
library(Matrix)
A <- t(Matrix(c(2.0, -4.0, 2.0, 4.0, -9.0, 7.0, 2.0, 1.0, 3.0), ncol=3))
alu <- Matrix::lu(A)
ealu <- expand(alu)
```

3. Cholesky Decomposition

- a matrix M is positive (semi)definite if

$$\mathbf{x}^T\mathbf{M}\mathbf{x} > 0 \ (\geq 0), \quad \forall \mathbf{x} \neq \mathbf{0}.$$

- We can always assume that a postive (semi)definite matrix is symmetric

3.1. Theorem

- **Theorem**: Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be symmetric and positive definite. Then $\mathbf{A} = \mathbf{L}\mathbf{L}^T$, where $\mathbf{L}$ is lower triangular with positive diagonal entries and is unique.
- proof: by induction (HW)
- algorithm:

```
for (k in seq_len(n)) {
    for (j in k + seq_len(n - k)) {
        a_jk_div_a_kk <- A[j, k] / A[k, k]
        for (i in j - 1 + seq_len(n - j + 1)) {
```

```
            A[i, j] <- A[i, j] - A[i, k] * a_jk_div_a_kk    # L_22
        }
    }
    sqrt_akk <- sqrt(A[k, k])
    for (i in k - 1 + seq_len(n - k + 1)) {
        A[i, k] = A[i, k] / sqrt_akk    # l_11 and b
    }
}
```

- cost:

$$\frac{1}{2}[2(n-1)^2 + 2(n-2)^2 + \cdots + 2 \cdot 1^2] \approx \frac{1}{3}n^3 \quad \text{flops}$$

- stable한 알고리즘 → 분해 실패 = $\mathbf{A}$가 양정치가 아닌 것으로 판단

3.2. Pivoting

- pivoting은 준양정치 행렬의 Cholesky factor 얻고자 할 때만 필요
- symmetric pivoting: row와 column을 동시에 permute 하여 $\max_{k \le i \le n} a_{ii}$가 pivot이 되도록 함
  - If we encounter $\max_{k \le i \le n} a_{ii} = 0$, then $\mathbf{A}[k:n, k:n] = \mathbf{0}$ and the algorithm terminates
- With symmetric pivoting:

$$\mathbf{PAP}^T = \mathbf{LL}^T,$$

  where $\mathbf{P}$ is a permutation matrix and $\mathbf{L} \in \mathbb{R}^{n \times r}$, $r = \text{rank } (\mathbf{A})$.

Implementation

- LAPACK: dpotrf, dpstrf
- R: `base::chol()` or `Matrix::chol()`

3.3. Example: Positive Definite Matrix

```
library(Matrix)
A <- t(Matrix(c(4, 12, -16, 12, 37, -43, -16, -43, 98), nrow=3))
b <- c(1.0, 2.0, 3.0)

Achol <- Matrix::chol(Matrix::symmpart(A)) # upper triangle matrix; A = R^T R
y <- solve(t(Achol), b) # two triangular solves; only 2n^2 flops; Ax = R^T Rx
= b
solve(Achol, y) # Rx = y
```

```
[1] 28.583333 -7.666667  1.333333
```

- 행렬식

```
det(Achol)^2
```

```
[1] 36
```

- 역행렬

```
solve(Achol, solve(t(Achol))) # A^-1 = R^-1 R^-T; R^T A^-1 = R^-1
```

```
3 x 3 Matrix of class "dgeMatrix"
           [,1]        [,2]       [,3]
[1,]  49.361111 -13.5555556  2.1111111
[2,] -13.555556   3.7777778 -0.5555556
[3,]   2.111111  -0.5555556  0.1111111
```

3.4. Example: Positive Semidefinite Matrix

```
A <- Matrix(c(1,1,.5,1,1,.5,.5,.5,1), nrow=3)
(Achol <- base::chol(A, pivot=TRUE)) # turn off checking pd
```

```
Warning in chol.default(A, pivot = TRUE): the matrix is either rank-deficient
or not positive definite
```

```
     [,1]      [,2] [,3]
[1,]    1 0.5000000    1
[2,]    0 0.8660254    0
[3,]    0 0.0000000    0
attr(,"pivot")
[1] 1 3 2
attr(,"rank")
[1] 2
```

3.5. Applications

Multivariate Normal Density

- density:

$$\frac{1}{(2\pi)^{n/2}\det(\Sigma)^{1/2}}\exp\left(-\frac{1}{2}\mathbf{y}^T\Sigma^{-1}\mathbf{y}\right).$$

- Hence the log likelihood is

$$\frac{n}{2}\log(2\pi)-\frac{1}{2}\log\det\Sigma-\frac{1}{2}\mathbf{y}^T\Sigma^{-1}\mathbf{y}.$$

- Method 1:

  1. compute explicit inverse Σ^{-1} ($2n^3$ flops),

2. compute quadratic form ($2n^2 + 2n$ flops),
3. compute determinant ($2n^3/3$ flops).

- Method 2:

1. Cholesky decomposition $\Sigma = \mathbf{L}\mathbf{L}^T$ ($n^3/3$ flops),
2. Solve $\mathbf{L}\mathbf{x} = \mathbf{y}$ by forward substitutions (n^2 flops),
3. compute quadratic form $\mathbf{x}^T\mathbf{x}$ ($2n$ flops), and
4. compute determinant from Cholesky factor (n flops).

- To evaluate same multivariate normal density at many observations $y_1, y_2, ...$, we pre-compute the Cholesky decomposition ($n^3/3$ flops), then each evaluation costs n^2 flops.

Linear Regression by Cholesky

- Assume $\mathbf{X} \in \mathbb{R}^{n\times p}$ and $\mathbf{y} \in \mathbb{R}^n$.
  - Compute $\mathbf{X}^T\mathbf{X}$: np^2 flops
  - Compute $\mathbf{X}^T\mathbf{y}$: $2np$ flops
  - Cholesky decomposition of $\mathbf{X}^T\mathbf{X}$: $\frac{1}{3}p^3$ flops
  - Solve normal equation $\mathbf{X}^T\mathbf{X}\beta = \mathbf{X}^T\mathbf{y}$: $2p^2$ flops
  - If need standard errors, another $(4/3)p^3$ flops
- Total computational cost is $np^2 + (1/3)p^3$ (without s.e.) or $np^2 + (5/3)p^3$ (with s.e.) flops.

4. QR Decomposition

- `lm()`에서 linear regression 푸는 방식

4.1. Definition

- Assume $\mathbf{X} \in \mathbb{R}^{n\times p}$ has full column rank. Necessarilly $n \geq p$.
- **Full QR decomposition**:

$$\mathbf{X} = \mathbf{Q}\mathbf{R},$$

where

  - $\mathbf{Q} \in \mathbb{R}^{n\times n}$, $\mathbf{Q}^T\mathbf{Q} = \mathbf{Q}\mathbf{Q}^T = \mathbf{I}_n$. In other words, $\mathbf{Q}$ is an orthogonal matrix.
    - First p columns of $\mathbf{Q}$ form an orthonormal basis of $\mathcal{R}(\mathbf{X})$ (**range** or column space of $\mathbf{X}$)
    - Last $n - p$ columns of $\mathbf{Q}$ form an orthonormal basis of $\mathcal{N}(\mathbf{X}^T)$ (**null space** of $\mathbf{X}^T$)
    - Recall that $\mathcal{N}(\mathbf{X}^T) = \mathcal{R}(\mathbf{X})^\perp$ and $\mathcal{R}(\mathbf{X}) \oplus \mathcal{N}(\mathbf{X}^T) = \mathbb{R}^n$.
  - $\mathbf{R} \in \mathbb{R}^{n\times p}$ is upper triangular with positive diagonal entries.
    - The lower $(n - p) \times p$ block of $\mathbf{R}$ is $\mathbf{0}$

- **Reduced QR decomposition**:

$$\mathbf{X} = \mathbf{Q}_1\mathbf{R}_1,$$

where

- $\mathbf{Q}_1 \in \mathbb{R}^{n\times p}$, $\mathbf{Q}_1^T\mathbf{Q}_1 = \mathbf{I}_p$. In other words, $\mathbf{Q}_1$ is a partially orthogonal matrix. Note $\mathbf{Q}_1\mathbf{Q}_1^T \neq \mathbf{I}_n$.
- $\mathbf{R}_1 \in \mathbb{R}^{p\times p}$ is an upper triangular matrix with positive diagonal entries.

- Given QR decomposition $\mathbf{X} = \mathbf{QR}$,

$$\mathbf{X}^T\mathbf{X} = \mathbf{R}^T\mathbf{Q}^T\mathbf{QR} = \mathbf{R}^T\mathbf{R} = \mathbf{R}_1^T\mathbf{R}_1.$$

  - Once we have a (reduced) QR decomposition of $\mathbf{X}$, we automatically have the Cholesky decomposition of the Gram matrix $\mathbf{X}^T\mathbf{X}$.

Least Squares

- Normal equation

$$\mathbf{X}^T\mathbf{X}\beta = \mathbf{X}^T\mathbf{y}$$

  is equivalently written with reduced QR as

$$\mathbf{R}_1^T\mathbf{R}_1\beta = \mathbf{R}_1^T\mathbf{Q}_1^T\mathbf{y}$$

- $\mathbf{R}_1$이 invertible이므로 다음을 풀면 됨 ($\mathbf{R}_1$이 upper-triangular 이므로 backsolve로)
  - Multiplication $\mathbf{Q}_1^T\mathbf{y}$ is done implicitly (see below).

$$\mathbf{R}_1\beta = \mathbf{Q}_1^T\mathbf{y}$$

- 이 방법이 numerically more stable since $\kappa(\mathbf{X}^T\mathbf{X}) = \kappa(\mathbf{X})^2$!
- standard error가 필요한 경우 $\mathbf{R}_1^T\mathbf{R}_1$의 역행렬 계산 → triangular solve 필요

4.2. Gram-Schmidt Procedure

- $\mathbf{X}$는 항상 QR 분해를 갖는다 ($\Leftrightarrow$ Gram-Schmidt Procedure for basis orthonormalization)
- Assume $\mathbf{X} = \left[\mathbf{x}_1|\cdots|\mathbf{x}_p\right] \in \mathbb{R}^{n\times p}$ has full column rank. That is, $\mathbf{x}_1, ..., \mathbf{x}_p$ are linearly independent.
- Gram-Schmidt (GS) procedure produces nested orthonormal basis vectors $\left\{\mathbf{q}_1, ..., \mathbf{q}_p\right\}$ that spans $\mathcal{R}(\mathbf{X})$, i.e.,

$$\begin{aligned}
\text{span }(\{\mathbf{x}_1\}) &= \text{span }(\{\mathbf{q}_1\}) \\
\text{span }(\{\mathbf{x}_1, \mathbf{x}_2\}) &= \text{span }(\{\mathbf{q}_1, \mathbf{q}_2\}) \\
&\vdots \\
\text{span }\left(\left\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_p\right\}\right) &= \text{span }\left(\left\{\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_p\right\}\right)
\end{aligned}$$

  and $\langle \mathbf{q}_i, \mathbf{q}_j \rangle = \delta_{ij}$.

- 알고리즘:

  0. Initialize $\mathbf{q}_1 = \mathbf{x}_1 / \parallel \mathbf{x}_1 \parallel_2$
  1. For $k = 2, ..., p$,

$$\mathbf{v}_k = \mathbf{x}_k - P_{\text{span }(\{\mathbf{q}_1,\ldots,\mathbf{q}_{k-1}\})}(\mathbf{x}_k) = \mathbf{x}_k - \sum_{j=1}^{k-1} \langle \mathbf{q}_j, \mathbf{x}_k \rangle \cdot \mathbf{q}_j$$

$$\mathbf{q}_k = \mathbf{v}_k / \| \mathbf{v}_k \|_2$$

- note.
  - $P_{\text{span }(\{\mathbf{q}_1,\ldots,\mathbf{q}_{k-1}\})}(\mathbf{x}_k)$: $k = 2$인 경우, $\mathbf{q}_1$이 span 하는 subspace로 $\mathbf{x}_2$를 projection 한 것

GS Conducts Reduced QR

- $\mathbf{Q} = [\mathbf{q}_1 | \cdots | \mathbf{q}_p]$이고 $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}_p$임은 자명.
- $\mathbf{R}$의 구성:
  - Let $r_{jk} = \langle \mathbf{q}_j, \mathbf{x}_k \rangle$ for $j < k$, and $r_{kk} = \| \mathbf{v}_k \|_2$.
  - Re-write the above expression:

$$r_{kk}\mathbf{q}_k = \mathbf{x}_k - \sum_{j=1}^{k-1} r_{jk} \cdot \mathbf{q}_j$$

    or

$$\mathbf{x}_k = r_{kk}\mathbf{q}_k + \sum_{j=1}^{k-1} r_{jk} \cdot \mathbf{q}_j.$$

  - If we let $r_{jk} = 0$ for $j > k$, then $\mathbf{R} = (r_{jk})$ is upper triangular and

$$\mathbf{X} = \mathbf{QR}.$$

Classical Gram-Schmidt

```
cgs <- function(X) {  # not in-place
    n <- dim(X)[1]
    p <- dim(X)[2]
    Q <- Matrix(0, nrow=n, ncol=p, sparse=FALSE)
    R <- Matrix(0, nrow=p, ncol=p, sparse=FALSE)
    for (k in seq_len(p)) {
        Q[, k] <- X[, k]
        for (j in seq_len(k-1)) {
            R[j, k] = sum(Q[, j] * X[, k])  # dot product
            Q[, k] <- Q[, k] - R[j, k] * Q[, j]
        }
        R[k, k] <- Matrix::norm(Q[, k, drop=FALSE], "F")
        Q[, k] <- Q[, k] / R[k, k]
    }
    list(Q=Q, R=R)
}
```

- CGS는 unstable (rounding error로 인해 orthogonality 잃을 수 있음 - 열이 거의 collinear한 경우)
- 문제가 발생하는 곳:
  - `Q[, k] <- Q[, k] - R[j, k] * Q[, j]`
  - `Q[, k] <- Q[, k] / R[k, k]`

Modified Gram-Schmidt

- The algorithm:

  0. Initialize $\mathbf{q}_1 = \mathbf{x}_1 / \| \mathbf{x}_1 \|_2$
  1. For $k = 2, ..., p$,

$$\begin{aligned}
\mathbf{v}_k &= \mathbf{x}_k - P_{\text{span}\,(\{\mathbf{q}_1,\ldots,\mathbf{q}_{k-1}\})}(\mathbf{x}_k) = \mathbf{x}_k - \sum_{j=1}^{k-1} \langle \mathbf{q}_j, \mathbf{x}_k \rangle \cdot \mathbf{q}_j \\
&= \mathbf{x}_k - \sum_{j=1}^{k-1} \left\langle \mathbf{q}_j, \mathbf{x}_k - \sum_{l=1}^{j-1} \langle \mathbf{q}_l, \mathbf{x}_k \rangle \mathbf{q}_l \right\rangle \cdot \mathbf{q}_j \\
\mathbf{q}_k &= \mathbf{v}_k / \| \mathbf{v}_k \|_2
\end{aligned}$$

```
mgs <- function(X) {  # in-place
    n <- dim(X)[1]
    p <- dim(X)[2]
    R <- Matrix(0, nrow=p, ncol=p)
    for (k in seq_len(p)) {
        for (j in seq_len(k-1)) {
            R[j, k] <- sum(X[, j] * X[, k])  # dot product
            X[, k] <- X[, k] - R[j, k] * X[, j]
        }
        R[k, k] <- Matrix::norm(X[, k, drop=FALSE], "F")
        X[, k] <- X[, k] / R[k, k]
    }
    list(Q=X, R=R)
}
```

- $\mathbf{X}$ is overwritten by $\mathbf{Q}$ and $\mathbf{R}$ is stored in a separate array.
- CGS 보다 stable 하지만 완전히 immune 하지는 않음
  - 언제 문제 발생?
- CGS와 MGS의 computational cost: $\sum_{k=1}^{p} 4n(k-1) \approx 2np^2$

4.3. QR by Housholder Transform

- LAPACK에 내장된 방법 (R에서 사용됨)
- GS는 다음과 같이 쓸 수 있음

$$\mathbf{X}\mathbf{R}_1\mathbf{R}_2\cdots\mathbf{R}_n = \mathbf{Q}_1$$

where $\mathbf{R}_j$ are a sequence of upper triangular matrices.

- Householder QR:

$$\mathbf{H}_p\cdots\mathbf{H}_2\mathbf{H}_1\mathbf{X} = \begin{pmatrix}\mathbf{R}_1\\ \mathbf{0}\end{pmatrix},$$

where $\mathbf{H}_j \in \mathbf{R}^{n\times n}$ are a sequence of Householder transformation matrices.

  - It yields the **full QR** where $\mathbf{Q} = \mathbf{H}_1\cdots\mathbf{H}_p \in \mathbb{R}^{n\times n}$.
  - Recall that CGS/MGS only produces the **reduced QR** decomposition.

- 방법:

  - For arbitrary $\mathbf{x}, \mathbf{w} \in \mathbb{R}^n$ with $\| \mathbf{x} \|_2 = \| \mathbf{w} \|_2$, we can construct a **Householder matrix** (or **Householder reflector**)

$$\mathbf{H} = \mathbf{I}_n - 2\mathbf{u}\mathbf{u}^T, \quad \mathbf{u} = -\frac{1}{\| \mathbf{x} - \mathbf{w} \|_2}(\mathbf{x} - \mathbf{w}),$$

    that transforms $\mathbf{x}$ to $\mathbf{w}$:

$$\mathbf{H}\mathbf{x} = \mathbf{w}.$$

    - $\mathbf{H}$는 대칭, 직교행렬
    - $\mathbf{u}$ 계산: $3n$ flops
  - Now choose $\mathbf{H}_1$ so that

$$\mathbf{H}_1\mathbf{x}_1 = \begin{pmatrix}\pm \| \mathbf{x}_1 \|_2\\ 0\\ \vdots\\ 0\end{pmatrix}.$$

    That is, $\mathbf{x} = \mathbf{x}_1$ and $\mathbf{w} = \pm \| \mathbf{x}_1 \|_2 \mathbf{e}_1$.
    - Left-multiplying $\mathbf{H}_1$ zeros out the first column of $\mathbf{X}$ below (1, 1).
  - Take $\mathbf{H}_2$ to zero the second column below diagonal:

$$\mathbf{H}_2\mathbf{H}_1\mathbf{X} = \begin{bmatrix}\times & \times & \times & \times\\ 0 & \times & \times & \times\\ 0 & \mathbf{0} & \times & \times\\ 0 & \mathbf{0} & \times & \times\\ 0 & \mathbf{0} & \times & \times\end{bmatrix}$$

- In general, choose the j-th Householder transform $\mathbf{H}_j = \mathbf{I}_n - 2\mathbf{u}_j\mathbf{u}_j^T$, where

$$\mathbf{u}_j = \begin{bmatrix} \mathbf{0}_{j-1} \\ \tilde{u}_j \end{bmatrix}, \quad \tilde{u}_j \in \mathbb{R}^{n-j+1},$$

to zero the j-th column below diagonal. $\mathbf{H}_j$ takes the form

$$\mathbf{H}_j = \begin{bmatrix} \mathbf{I}_{j-1} & \\ & \mathbf{I}_{n-j+1} - 2\tilde{u}_j\tilde{u}_j^T \end{bmatrix} = \begin{bmatrix} \mathbf{I}_{j-1} & \\ & \tilde{H}_j \end{bmatrix}.$$

- Applying a Householder transform $\mathbf{H} = \mathbf{I} - 2\mathbf{u}\mathbf{u}^T$ to a matrix $\mathbf{X} \in \mathbb{R}^{n\times p}$

$$\mathbf{H}\mathbf{X} = \mathbf{X} - 2\mathbf{u}(\mathbf{u}^T\mathbf{X})$$

costs $4np$ flops. **Householder updates never entails explicit formation of the Householder matrices.**

- Note applying $\tilde{H}_j$ to $\mathbf{X}$ only needs $4(n-j+1)(p-j+1)$ flops.

Algorithm

```
for (j in seq_len(p)) {  # in-place
    u <- Householder(X[j:n, j])
    for (i in j:p)
        X[j:n, i] <- X[j:n, i] - 2 * u * sum(u * X[j:n, i])
    }
}
```

- done in place
- Upper triangular part of $\mathbf{X}$ is overwritten by $\mathbf{R}_1$ and the essential Householder vectors ($\tilde{u}_{j1}$ is normalized to 1) are stored in $\mathbf{X}[j:n, j]$.
- Can ensure $u_{j1} \neq 0$ by a clever choice of the sign in determining vector $\mathbf{w}$ above
- At j-th stage

  0. computing the Householder vector $\tilde{u}_j$ costs $3(n-j+1)$ flops
  1. applying the Householder transform $\tilde{H}_j$ to the $\mathbf{X}[j:n, j:p]$ block costs $4(n-j+1)(p-j+1)$ flops

- In total we need $\sum_{j=1}^{p}[3(n-j+1) + 4(n-j+1)(p-j+1)] \approx 2np^2 - \frac{2}{3}p^3$ flops.
- $\mathbf{Q} = \mathbf{H}_1 \cdots \mathbf{H}_p$는 명시적으로 저장되지 않음
  - Accumulating $\mathbf{Q}$ costs another $2np^2 - \frac{2}{3}p^3$ flops.
  - linear equation 풀 때는 $\mathbf{Q}$ 계산할 필요 없음 (regression coefficients와 $\hat{\sigma}^2$); fitted value 계산 시에는 필요

```
set.seed(2020) # seed
```

```r
n <- 5
p <- 3
(X <- matrix(rnorm(n * p), nrow=n)) # predictor matrix
```

```
           [,1]       [,2]       [,3]
[1,]  0.3769721  0.7205735 -0.8531228
[2,]  0.3015484  0.9391210  0.9092592
[3,] -1.0980232 -0.2293777  1.1963730
[4,] -1.1304059  1.7591313 -0.3715839
[5,] -2.7965343  0.1173668 -0.1232602
```

```r
(y <- rnorm(n))    # response vector
```

```
[1]  1.80004312  1.70399588 -3.03876461 -2.28897495  0.05830349
```

```r
qr.solve(X, y)  # coef(lm(y ~ X -1))과 동일
```

```
[1]  0.615117664 -0.008382116 -0.770116370
```

```r
(qrobj <- qr(X, LAPACK=TRUE))
```

```
$qr
           [,1]        [,2]       [,3]
[1,] -3.2460924  0.46519442  0.1837043
[2,]  0.0832302 -2.08463446  0.3784090
[3,] -0.3030648 -0.05061826 -1.7211061
[4,] -0.3120027  0.61242637 -0.4073016
[5,] -0.7718699  0.10474144 -0.3750994

$rank
[1] 3

$qraux
[1] 1.116131 1.440301 1.530697

$pivot
[1] 1 2 3

attr(,"useLAPACK")
[1] TRUE
attr(,"class")
[1] "qr"
```

- `$qr`: a matrix of same size as input matrix
  - The upper triangular part of `qrobj$qr` contains the $\mathbf{R}$ of the decomposition and the lower triangular part contains information on the $\mathbf{Q}$ of the decomposition
- `$rank`: rank of the input matrix
- `$pivot`: pivot vector (for rank-deficient `X`)
- `$aux`: normalizing constants of Householder vectors

```
qr.Q(qrobj)  # extract Q
```

```
           [,1]       [,2]        [,3]
[1,] -0.11613105 -0.3715745  0.40159171
[2,] -0.09289581 -0.4712268 -0.64182039
[3,]  0.33825998  0.1855167 -0.61822569
[4,]  0.34823590 -0.7661458  0.08461993
[5,]  0.86150792  0.1359480  0.19346097
```

```
qr.R(qrobj) # extract R
```

```
         [,1]       [,2]       [,3]
[1,] -3.246092  0.4651944  0.1837043
[2,]  0.000000 -2.0846345  0.3784090
[3,]  0.000000  0.0000000 -1.7211061
```

```
qr.qty(qrobj, y)  # this uses the "compact form"
```

```
          [,1]
[1,] -2.1421018
[2,] -0.2739453
[3,]  1.3254520
[4,] -3.3263425
[5,]  1.7707709
```

- GS에서 작동하지 않았던 분해도 가능

4.4. QR by Givens Rotation

- Householder 행렬 $\mathbf{H}_j$는 vector에 0 다수 존재
- Givens transform은 vector의 한 원소를 0으로 (Givens $(n-1)$번 = Householder 1번)
- Givens/Jacobi rotations:

$$\mathbf{G}(i,k,\theta) = \begin{bmatrix} 1 & & 0 & & 0 & & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & & c & & s & & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & & -s & & c & & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & & 0 & & 0 & & 1 \end{bmatrix},$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$. $\mathbf{G}(i,k,\theta)$ is orthogonal.

- 0을 만드는 원리:
  - Pre-multiplication by $\mathbf{G}(i,k,\theta)^T$ rotates counterclockwise θ radians in the (i,k) coordinate plane.
  - If $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} = \mathbf{G}(i,k,\theta)^T\mathbf{x}$, then

$$y_j = \begin{cases} cx_i - sx_kj = i \\ sx_i + cx_kj = k \\ x_j \qquad\quad j \neq i,k \end{cases}.$$

  - Apparently if we choose $\tan(\theta) = -x_k/x_i$, or equivalently,

$$c = \frac{x_i}{\sqrt{x_i^2 + x_k^2}}, \quad s = \frac{-x_k}{\sqrt{x_i^2 + x_k^2}},$$

    then $y_k = 0$.
- Pre-applying Givens transform: 두 개의 행만 영향 ($6n$ flops)
- Post-applying Givens transform: 두 개의 열만 영향 ($6n$ flops)
- QR by Givens: $\mathbf{G}_t^T \cdots \mathbf{G}_1^T\mathbf{X} = \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix}$
- If $\mathbf{X} \in \mathbb{R}^{n\times p}$, the total cost is $3np^2 - p^3$ flops and $O(np)$ square roots.
- 각각의 transform은 하나의 숫자로 요약 가능: stored in the zeroed entry of $\mathbf{X}$.

4.5. Conditioning of Linear Regression by Least Squares

- condition number for the least squares problem $\mathbf{y} \approx \mathbf{X}\beta$ is more complicated and depends on the residual.
- the condition number is

$$\kappa(\mathbf{X}) + \frac{\kappa(\mathbf{X})^2 \tan\theta}{\eta},$$

where

$$\theta = \cos^{-1}\frac{\| \mathbf{X}\boldsymbol{\beta} \|}{\| \boldsymbol{\beta} \|}, \quad \eta = \frac{\| \mathbf{X} \| \| \boldsymbol{\beta} \|}{\| \mathbf{X}\boldsymbol{\beta} \|}.$$

- $\kappa(\mathbf{X})$가 크면 ($\mathbf{X}$가 collinear) condition number의 크기가 $\kappa(\mathbf{X})^2$에 의해 dominate 됨 → stable algorithm is preferred

4.6. Summary of Linear Regression

- Methods for solving linear regression $\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$:

Method	Flops	Remarks	Software	Stability
Cholesky	$np^2 + p^3/3$			unstable
QR by Householder	$2np^2 - (2/3)p^3$		R, Julia, MATLAB	stable
QR by MGS	$2np^2$	Q_1 available		stable
QR by SVD	$4n^2p + 8np^2 + 9p^3$	$X = UDV^T$		most stable

- remarks
  - $n >> p$이면 Cholesky가 QR보다 2배 빠르고 공간을 절약
  - Cholesky는 gram matrix에 기반 (huge n, moderate p 데이터 handle 가능)
  - QR method는 더 stable하고 numerically 정확한 solution
  - MGS는 Householder에 비해 느리지만 $\mathbf{Q}_1$을 계산

Undetermined Least Squares

- gram matrix가 singular인 경우 ($\text{rank }(\mathbf{X}) < p$, in particular $n < p$)
- require solution $\mathbf{x}$ has a smallest norm
- depends on the choice of the norm
  - ℓ_2 norm: closed form solution
  - ℓ_0 'norm': model selection problem (combinatorial)
  - ℓ_1 norm: convex optimization → tractable

5. Iterative Methods for Solving Linear Equations

5.1. Motivation: PageRank

- PageRank 행렬의 stationary distribution

$$\mathbf{x}^T\mathbf{P} = \mathbf{x}^T.$$

Equivalently it can be cast as a linear equation.

$$(\mathbf{I} - \mathbf{P}^T)\mathbf{x} = \mathbf{0}.$$

- n이 매우 큰 경우 → direct method로는 매우 오래 걸림

5.2. Splitting and Fixed Point Iteration

- key idea: 행렬 $\mathbf{A} = \mathbf{M} - \mathbf{K}$로 split
  - $\mathbf{M}$: 쉽게 invert 할 수 있는 행렬
- Then $\mathbf{Ax} = \mathbf{Mx} - \mathbf{Kx} = \mathbf{b}$ or

$$\mathbf{x} = \mathbf{M}^{-1}\mathbf{Kx} + \mathbf{M}^{-1}\mathbf{b}.$$

  Thus a solution to $\mathbf{Ax} = \mathbf{b}$ is a fixed point of iteration

$$\mathbf{x}^{(t+1)} = \mathbf{M}^{-1}\mathbf{Kx}^{(t)} + \mathbf{M}^{-1}\mathbf{b} = \mathbf{Rx}^{(t)} + \mathbf{c}.$$

- $\mathbf{R}$을 적절하게 선택 → the sequence $\mathbf{x}^{(k)}$ generated by the above iteration converges to a solution $\mathbf{Ax} = \mathbf{b}$
  - converge if and only if $\rho(\mathbf{R}) < 1$
  - $\rho(\mathbf{R}) = \max_{i=1,\dots,n} |\lambda_i(\mathbf{R})|$

Matrix Norms

- operator norm (induced by a vector norm): $\| \mathbf{A} \| = \sup_{\mathbf{y} \neq \mathbf{0}} \frac{\|\mathbf{Ay}\|}{\|\mathbf{y}\|}$
- norm의 성질
  1. $\| \mathbf{x} \| \geq 0$,
  2. $\| \mathbf{x} \| = 0 \Leftrightarrow \mathbf{x} = \mathbf{0}$,
  3. $\| c\mathbf{x} \| = |c| \, \| \mathbf{x} \|$ for all $c \in \mathbb{R}$,
  4. $\| \mathbf{x} + \mathbf{y} \| \leq \| \mathbf{x} \| + \| \mathbf{y} \|$.
  5. $\| \mathbf{AB} \| \leq \| \mathbf{A} \| \| \mathbf{B} \|$.

Convergence of Iterative Methods

- if $\| \mathbf{R} \| < 1$ for a certain (induced) norm, then $\mathbf{x}^{(t)} \to \mathbf{x}^\star$
  - Let $\mathbf{x}^\star$ be a solution of $\mathbf{Ax} = \mathbf{b}$.
  - Equivalently, $\mathbf{x}^\star$ satisfies $\mathbf{x}^\star = \mathbf{Rx}^\star + \mathbf{c}$.
  - Subtracting this from $\mathbf{x}^{(t+1)} = \mathbf{Rx}^{(t)} + \mathbf{c}$, we have

$$\mathbf{x}^{(t+1)} - \mathbf{x}^\star = \mathbf{R}\big(\mathbf{x}^{(t)} - \mathbf{x}^\star\big).$$

  - Take a vector norm on both sides:

$$\| \mathbf{x}^{(t+1)} - \mathbf{x}^\star \| = \| \mathbf{R}\big(\mathbf{x}^{(t)} - \mathbf{x}^\star\big) \| \leq \| \mathbf{R} \| \| \mathbf{x}^{(t)} - \mathbf{x}^\star \|$$

    where $\| \mathbf{R} \|$ is the induced operator norm.
- theorem (pf 생략):
  - The spectral radius $\rho(\mathbf{A})$ of a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ satisfies

$$\rho(\mathbf{A}) \leq \| \mathbf{A} \|$$

for any operator norm.

- for any $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\epsilon > 0$, there is an operator norm $\|\cdot\|$ such that $\| \mathbf{A} \| \leq \rho(\mathbf{A}) + \epsilon$.
- In other words,

$$\rho(\mathbf{A}) = \inf_{\|\cdot\|\text{: operator norm}} \| \mathbf{A} \| .$$

- Thus it is immediate to see

$$\rho(\mathbf{A}) < 1 \Leftrightarrow \| \mathbf{A} \| < 1$$

for some operator norm.

Implications

- When $\mathbf{A}$ is invertible, the iteration $\mathbf{x}^{(t+1)} = \mathbf{R}\mathbf{x}^{(t)} + \mathbf{c}$ converges to $\mathbf{A}^{-1}\mathbf{b}$ for any $\mathbf{x}^{(0)}$ if and only if $\rho(\mathbf{R}) < 1$.
  - ($\Leftarrow$) 위의 정리 ($\mathbf{A}$가 singular여도 성립)
  - ($\Rightarrow$) when $\mathbf{A}$ is invertible (hence $\mathbf{x}^\star$ is unique):

$$\mathbf{x}^{(t)} - \mathbf{x}^\star = \mathbf{R}^t\big(\mathbf{x}^{(0)} - \mathbf{x}^\star\big).$$

    - If LHS converges to zero for any $\mathbf{x}^{(0)}$, then $\mathbf{R}^t \to \mathbf{0}$ as $t \to \infty$.
    - This cannot occur if $|\lambda_i(\mathbf{R})| \geq 1$ for some i:
    - if $\mathbf{x}^{(0)} = \mathbf{v}_i + \mathbf{x}^\star$ for an eigenvector $\mathbf{v}_i \neq \mathbf{0}$ such that $\mathbf{R}\mathbf{v}_i = \lambda_i \mathbf{v}_i$, then $\| R^t\big(\mathbf{x}^{(0)} - \mathbf{x}^\star\big) \| = |\lambda_i|^t \| \mathbf{v}_i \| \geq \| \mathbf{v}_i \| \neq 0$.

5.3. Jacobi's Method

- 어떤 $\mathbf{R}$을 선택해야 하는가의 문제 ($\mathbf{X}$를 어떻게 split?)
- Split $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$ = (strictly lower triangular) + (diagonal) + (strictly upper triangular).
- Take $\mathbf{M} = \mathbf{D}$ (easy to invert!) and $\mathbf{K} = -(\mathbf{L} + \mathbf{U})$:

$$\mathbf{D}\mathbf{x}^{(t+1)} = -(\mathbf{L} + \mathbf{U})\mathbf{x}^{(t)} + \mathbf{b},$$

i.e.,

$$\mathbf{x}^{(t+1)} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}^{(t)} + \mathbf{D}^{-1}\mathbf{b} = -\mathbf{D}^{-1}\mathbf{A}\mathbf{x}^{(t)} + \mathbf{x}^{(t)} + \mathbf{D}^{-1}\mathbf{b}.$$

$$x_i^{(t+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(t)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(t)}}{a_{ii}}.$$

- Convergence is guaranteed if $\mathbf{A}$ is striclty row diagonally dominant: $|a_{ii}| > \sum_{j\neq i} |a_{ij}|$.
- One round costs $2n^2$ flops with an **unstructured** $\mathbf{A}$. Gain over GE/LU if converges in $o(n)$ iterations.
- Saving is huge for **sparse** or **structured** $\mathbf{A}$. By structured, we mean the matrix-vector multiplication $\mathbf{A}\mathbf{v}$ is fast ($O(n)$ or $O(n \log n)$).

- ▸ Often the multiplication is implicit and $\mathbf{A}$ is not even stored, e.g., finite difference: $(\mathbf{Av})_i = v_i - v_{i+1}$.

5.4. Gauss-Seidel Method

- Split $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$ = (strictly lower triangular) + (diagonal) + (strictly upper triangular) as Jacobi.
- Take $\mathbf{M} = \mathbf{D} + \mathbf{L}$ (easy to invert, why?) and $\mathbf{K} = -\mathbf{U}$:

$$(\mathbf{D} + \mathbf{L})\mathbf{x}^{(t+1)} = -\mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}$$

  i.e.,

$$\mathbf{x}^{(t+1)} = -(\mathbf{D} + \mathbf{L})^{-1}\mathbf{U}\mathbf{x}^{(t)} + (\mathbf{D} + \mathbf{L})^{-1}\mathbf{b}.$$

- Equivalent to

$$\mathbf{D}\mathbf{x}^{(t+1)} = -\mathbf{L}\mathbf{x}^{(t+1)} - \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}$$

  or

$$\mathbf{x}^{(t+1)} = \mathbf{D}^{-1}\big(-\mathbf{L}\mathbf{x}^{(t+1)} - \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}\big)$$

  leading to the iteration.

$$x_i^{(t+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(t+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(t)}}{a_{ii}}.$$

- "Coordinate descent(성분별)" version of Jacobi. → 바로 이전 성분을 사용하므로 수렴 속도 더 빠름
- Convergence is guaranteed if $\mathbf{A}$ is stricly row diagonally dominant.

5.5. Successive Over-Relaxation (SOR)

- Split $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$ = (strictly lower triangular) + (diagonal) + (strictly upper triangular) as before.
- Take $\mathbf{M} = \frac{1}{\omega}\mathbf{D} + \mathbf{L}$ (easy to invert, why?) and $\mathbf{K} = \frac{1-\omega}{\omega}\mathbf{D} - \mathbf{U}$:

$$\begin{aligned}(\mathbf{D} + \omega\mathbf{L})\mathbf{x}^{(t+1)} &= [(1-\omega)\mathbf{D} - \omega\mathbf{U}]\mathbf{x}^{(t)} + \omega\mathbf{b} \\ \mathbf{D}\mathbf{x}^{(t+1)} &= (1-\omega)\mathbf{D}\mathbf{x}^{(t)} + \omega\big(-\mathbf{U}\mathbf{x}^{(t)} - \mathbf{L}\mathbf{x}^{(t+1)} + \mathbf{b}\big) \\ \mathbf{x}^{(t+1)} &= (1-\omega)\mathbf{x}^{(t)} + \omega\mathbf{D}^{-1}\big(-\mathbf{U}\mathbf{x}^{(t)} - \mathbf{L}\mathbf{x}^{(t+1)} + \mathbf{b}\big)\end{aligned}$$

  - ▸ $\omega = 1$: Gauss-Seidel; $\omega \in (0, 1)$: underrelaxation; $\omega > 1$: overrelaxation
  - ▸ Relaxation in hope of faster convergence

5.6. Conjugate Gradient Method

- huge, structured linear system 풀기에 가장 좋은 방법

- Key idea: solving $\mathbf{Ax} = \mathbf{b}$ is equivalent to minimizing the quadratic function $\frac{1}{2}\mathbf{x}^T\mathbf{Ax} - \mathbf{b}^T\mathbf{x}$ if $\mathbf{A}$ is positive definite.

5.7. Numerical Examples

Generate Test Matrix

- Poisson matrix: block tridiagonal matrix from Poisson's equation $\nabla^2\psi = f$ in on the plane
  - sparse, symmetric positive definite and has known eigenvalues

Result

- Cholesky → awfully slow
- iterative: Jacobi, Gauss-Seidel, SOR, Conjugate Gradient

6. Nonlinear Equation Solve

6.1. Problem

- Find $\mathbf{x}$ such that

$$f(\mathbf{x}) = 0.$$

  - f가 non-linear이면 closed form solution 존재하지 않음 → 직접법 사용 불가
  - iterative, numerical method 사용
- nonlinear equation과 optimization 유사 (g가 differentiable 하다면..)
  - maximizer of $g \Leftrightarrow$ solving $f(x) = g'(x) = 0$

6.2. Univariate Problems

- $f : \mathbb{R} \to \mathbb{R}$

Bisection

- f가 $[l, r]$에서 연속, $f(l)f(r) \leq 0$
- 중간값 정리에 의해 $\exists x^\star$ such that $f(x^\star) = 0$.
- consider the midpoint $m = \frac{l+r}{2}$
  - if $f(m) = 0$, done
  - otherwise, $r \leftarrow m$ if $f(l)f(m) < 0$, $l \leftarrow m$ if $f(m)f(r) < 0$
- n번 반복 이후 최종 interval의 길이: $\left(r^{(0)} - l^{(1)}\right)/2^n$
- algorithm

```
bisect <- function(f, l, r, tol=1e-6) {
    iter <- 0
    while ( r - l > tol ) {
        m <- (l + r) / 2
```

```
        if ( f(l) * f(m) < 0 ) {
            r <- m
        } else {
            l <- m
        }
        iter <- iter + 1
    }
    ret <- list(val = (l + r) / 2, iter = iter)
}
```

- example: quantile of continuous distribution

```
F <- function(x) {pt(x, 5) - 0.95} # Student's t distribution
ret <- bisect(F, 1.291, 2.582)
```

Fixed-Point Iteration

- def: $x \in \mathbb{R}$ is a fixed point of function $F : \mathbb{R} \to \mathbb{R}$ if $F(x) = x$
- strategy
  - determine F such that $f(x) = 0 \Leftrightarrow F(x) = x$
  - apply FPI: $x^{(k+1)} = F\left(x^{(k)}\right)$
- example: find a root of $f(x) = x^4 - \sin x$ on $(0, 1]$
  - method 1: $F_1(x) = (\sin x)^{1/4}$, since $f(x) = 0 \Leftrightarrow x = (\sin x)^{1/4}$
  - method 2: $F_2(x) = \frac{\sin x}{x^3}$ since $f(x) = 0 \Leftrightarrow x = \frac{\sin x}{x^3}$ → 발산
- contraction mapping theorem: Suppose the function F defined on a closed interval $I \subset \mathbb{R}$ satisfies

  1. $F(x) \in I$ whenever $x \in I$
  2. $|F(x) - F(y)| \leq L|x - y|$ for all $x, y \in I$ (Lipschitz continuity).

  Then, if $L \in [0, 1)$, F has a unique fixed point $x^\star \in I$. Furthermore, fixed-point iteration $x^{k+1} = F\left(x^k\right)$ converges to $x^\star$ regardless of the initial point $x^{(0)}$.

  - pf 참고
  - remark. rate of convergence (geometric) → linear convergence

$$\left|x^{(n)} - x^\star\right| \leq \frac{L^n}{1 - L}\left|x^{(1)} - x^{(0)}\right|$$

- Lipschitz modulus of a differentiable function: $L = \sup_{x \in I}|F'(x)|$
- recall example
  - F_1 maps $I = [0.2, 1]$ to a smaller interval. $F_1'(x) = \frac{\cos x}{4 \sin^{3/4}(x)}$ is decreasing on I and $1 > F_1'(0.2) > F_1'(1) > 0$.

- F_2 expands $I = [0.9, 1]$ to $[0.841, 1.075]$ and also $F_{2'}(x) = \frac{x\cos x - 3\sin x}{x^4}$ is increasing on I with $F_{2'}(1) < -1$.

Obvious Choice of F

- $F(x) = f(x) + x$ → iteration $x^{(k+1)} = x^{(k)} + f\big(x^{(k)}\big)$

Newton's Method

- iteration:

$$x^{(k+1)} = x^{(k)} - \frac{f\big(x^{(k)}\big)}{f'\big(x^{(k)}\big)}$$

- motivation: 1차 테일러 전개
  - if $x^{(k)} \simeq x^\star$ (root of f),

$$0 = f(x^\star) = f\big(x^{(k)}\big) + f'(\tilde{x})\big(x^\star - x^{(k)}\big)$$

    for some $\tilde{x}$ on the line segment connecting $x^{(k)}$ and $x^\star$
  - substitute $\tilde{x}$ with $x^{(k)}$ and $x^\star$ with $x^{(k+1)}$

$$0 = f\big(x^{(k)}\big) + f'\big(x^{(k)}\big)\big(x^{(k+1)} - x^{(k)}\big),$$

- Newton's method is a FPI with

$$F(x) = x - \frac{f(x)}{f'(x)}.$$

- local convergence is governed by

$$F'(x) = 1 - \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2}$$

  - If $f'(x^\star) \neq 0$ and $|f''(x^\star)| < \infty$, then $F'(x^\star) = 0$. So there is a neighborhood of $x^\star$ in which $|F'(x)| < 1$.
  - With this assumption, apply 2nd-order Taylor expansion of F around $x^\star$: for $\tilde{x}$ on the line segment connecting $x^{(k-1)}$ and $x^\star$.
  - If furthermore F'' is continuous and $x^{(0)}$ is sufficiently close to $x^\star$, then $x^{(k)} \to x^\star$ and

$$\lim_k \frac{\big|x^{(k)} - x^\star\big|}{\big|x^{(k-1)} - x^\star\big|^2} = \frac{1}{2}|F''(x^\star)|.$$

    Thus Newton's method has a locally quadratic convergence property (under suitable contidions).

- The local convergence theory requires a good starting point $x^{(0)}$ ("sufficiently close to $x^\star$"). With a good starting point, Newton-Rephson will converge very fast to the nearby solution. Otherwise, it can fail miserably.

6.3. Multivariate Problems

Newton's Method

- linearize f around $\mathbf{x} = (x_1, ..., x_n)$ → univariate version과 같은 아이디어
- Iteration:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \left[Jf\left(\mathbf{x}^{(k)}\right)\right]^{-1} f\left(\mathbf{x}^{(k)}\right)$$

  where

$$Jf(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_n(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

- Remember we should solve a linear system

$$\left[Jf\left(\mathbf{x}^{(k)}\right)\right]\Delta\mathbf{x} = -f\left(\mathbf{x}^{(k)}\right)$$

  to get $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}$, using LU etc, for each iteration.

Variants

- Modified Newton: replace $Jf\left(\mathbf{x}^{(k)}\right)$ with $J = Jf\left(\mathbf{x}^{(0)}\right)$.
- Jacobi's method: replace $Jf\left(\mathbf{x}^{(k)}\right)$ with $\operatorname{diag}\left(Jf\left(\mathbf{x}^{(k)}\right)\right)$.
- Gauss-Seidel: replace $Jf\left(\mathbf{x}^{(k)}\right)$ with $\operatorname{lowertri}\left(Jf\left(\mathbf{x}^{(k)}\right)\right)$.
- SOR: replace $Jf\left(\mathbf{x}^{(k)}\right)$ with $\frac{1}{\omega}\operatorname{diag}\left(Jf\left(\mathbf{x}^{(k)}\right)\right) + \operatorname{strictlowertri}\left(Jf\left(\mathbf{x}^{(k)}\right)\right)$.

7. Optimization

-problem:

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x})$$

7.1. Golden Section Search

- $d = 1$, 실직선의 구간에서 정의된 함수에 적용
- assumption: f is continuous and strictly unimodal
  - A function $f : [a, b] \to \mathbb{R}$ is strictly unimodal if

    $f(\lambda x + (1-\lambda)y) < \max\{f(x), f(y)\}, \quad x, y \in [a, b], x \neq y, \lambda \in (0, 1).$

- idea: keep track of three points $a < x_1 < b$ such that $f(x_1) < \min\{f(a), f(b)\}$
- choose a new point x_0 such that x_0 belongs to the longer of intervals (a, x_1) and (x_1, b)
- WLOG, suppose $a < x_0 < x_1$
  - If $f(x_0) > f(x_1)$, then the minimum lies in the interval (x_0, b). Hence set $(a, x_1, b) := (x_0, x_1, b)$.
  - Otherwise, the minimum lies in the interval (a, x_1). Hence set $(a, x_1, b) := (a, x_0, x_1)$.
- x_0의 선택: 두 candidate interval의 길이를 같게, 그리고 중간점의 상대적 위치를 같게 유지
  - 두 길이를 같게: $b - x_0 = x_1 - a$
  - $\alpha = \frac{x_0 - a}{b-a}$, $y = \frac{x_1 - x_0}{b-a}$로 두면 $\frac{b-x_1}{b-a} = \alpha$이고

$$1 : \alpha = y + \alpha : y, \quad \alpha + y + \alpha = 1$$

  - 풀면 $\alpha = \frac{3-\sqrt{5}}{2}$, 즉 $1 - \alpha = \frac{\sqrt{5}-1}{2}$ (**golden section**, 황금분할)
- algorithm

```
goldensection <- function(f, a, b, tol=1e-6) {
    alpha <- (3 - sqrt(5)) * 0.5
    iter <- 0
    while ( b - a > tol ) {
        x0 <- a + alpha * (b - a)
        x1 <- b - alpha * (b - a)
        if ( f(x0) > f(x1) ) {
            a <- x0
        } else {
            b <- x1
            x1 <- x0
        }
        iter <- iter + 1
    }
    list(val = (a + b) / 2, iter = iter)
}
```

- 수렴은 **느리지만 확실** (assumption 만족 시 global minimum 보장)

7.2. Newton's Method

- first-order optimality condition: $\nabla f(\mathbf{x}^\star) = \mathbf{0}$
  - $g(\mathbf{x}) = \nabla f(\mathbf{x})$에 Newton-Raphson 적용하는 것과 동일
- idea: iterative quadratic approximation (2차 테일러 전개)

$$f(\mathbf{x}) \approx f\big(\mathbf{x}^{(t)}\big) + \nabla f\big(\mathbf{x}^{(t)}\big)^T \big(\mathbf{x} - \mathbf{x}^{(t)}\big) + \frac{1}{2}\big(\mathbf{x} - \mathbf{x}^{(t)}\big)^T \big[\nabla^2 f\big(\mathbf{x}^{(t)}\big)\big]\big(\mathbf{x} - \mathbf{x}^{(t)}\big)$$

  - gradient를 0으로 두면 $\big(\nabla f\big(\mathbf{x}^{(t)}\big) + \big[\nabla^2 f\big(\mathbf{x}^{(t)}\big)\big]\big(\mathbf{x} - \mathbf{x}^{(t)}\big) = \mathbf{0}\big)$ 다음 iterate

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \left[\nabla^2 f\left(\mathbf{x}^{(t)}\right)\right]^{-1} \nabla f\left(\mathbf{x}^{(t)}\right)$$

- gold standard: 빠른 **quadratic convergence**

$$\frac{\| \mathbf{x}^{(t+1)} - \mathbf{x}^{\star} \|}{\| \mathbf{x}^{(t)} - \mathbf{x}^{\star} \|^2} \to \text{constant}$$

  - iteration마다 정확한 자릿수가 약 2배씩 증가
- 위를 **naive Newton's method**라 함
- **stability issue**: naive Newton은 descent algorithm이 아님 ($f\left(\mathbf{x}^{(t+1)}\right) \leq f\left(\mathbf{x}^{(t)}\right)$ 보장 X)
  - 내리막/오르막 가리지 않음 → 시작점에 따라 local max로 수렴, local min으로 수렴, 또는 발산

7.3. Practical Newton

- instability에 대한 해결책:
  1. $\nabla^2 f\left(\mathbf{x}^{(t)}\right)$를 양정치 행렬 $\mathbf{H}^{(t)}$로 근사 (양정치 아닐 경우)
  2. line search (backtracking)로 descent property 보장
- 왜 양정치 근사가 필요? **Newton direction** 정의:

$$\Delta \mathbf{x}^{(t)} = \left[\mathbf{H}^{(t)}\right]^{-1} \nabla f\left(\mathbf{x}^{(t)}\right)$$

  - 1차 테일러 전개:

$$f\left(\mathbf{x}^{(t)} + s\Delta \mathbf{x}^{(t)}\right) - f\left(\mathbf{x}^{(t)}\right) = -s \cdot \nabla f\left(\mathbf{x}^{(t)}\right)^T \left[\mathbf{H}^{(t)}\right]^{-1} \nabla f\left(\mathbf{x}^{(t)}\right) + o(s)$$

  - $\mathbf{H}^{(t)}$가 양정치이면 충분히 작은 s에 대해 위 값이 강하게 음수 → 감소 보장
  - $\left\{\nabla f\left(\mathbf{x}^{(t)}\right)^T \left[\mathbf{H}^{(t)}\right]^{-1} \nabla f\left(\mathbf{x}^{(t)}\right)\right\}^{1/2}$를 **Newton decrement**라 함
- **practical Newton-type algorithm**:

$$\boxed{\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - s\left[\mathbf{H}^{(t)}\right]^{-1} \nabla f\left(\mathbf{x}^{(t)}\right) = \mathbf{x}^{(t)} + s\Delta \mathbf{x}^{(t)}}$$

  - $\mathbf{H}^{(t)}$: $\nabla^2 f\left(\mathbf{x}^{(t)}\right)$에 대한 양정치 근사, s: step size
- strictly convex f이면 $\nabla^2 f$가 항상 양정치 → **damped Newton**
  - 그래도 수렴 보장 위해 line search는 (유한 번) 필요

Line Search

- Newton direction을 계산한 뒤 $f\left(\mathbf{x}^{(t)} + s\Delta \mathbf{x}^{(t)}\right)$를 최소화하는 s 탐색
  - direction은 한 번만 계산, 비용은 주로 함수값 평가
- full line search: $s = \arg\min_{\alpha} f\left(\mathbf{x}^{(t)} + \alpha \Delta \mathbf{x}^{(t)}\right)$ (e.g. golden section search 사용)
- approximate line search: step-halving ($s = 1, 1/2, \ldots$), Armijo rule, ⋯
- **backtracking line search (Armijo rule)**

```
# given: descent direction Δx, x ∈ domf, α ∈ (0, 0.5), β ∈ (0, 1)
t <- 1.0
while (f(x + t * delx) > f(x) + alpha * t * sum(gradf(x) * delx)) {
    t <- beta * t
}
```

- $\nabla^2 f(\mathbf{x})$를 어떻게 근사? science보다는 art → 문제별 분석 필요
- $\mathbf{H}^{(t)} = \mathbf{I}$로 두면 **gradient descent** (7.6 참조)

7.4. Fisher Scoring

- MLE에서 $f(\mathbf{x}) = -\ell(\boldsymbol{\theta})$, ℓ은 log-likelihood
- **Fisher scoring**: $-\nabla^2 \ell(\boldsymbol{\theta})$를 expected Fisher information matrix로 대체

$$\mathbf{I}(\theta) = \mathbf{E}\big[-\nabla^2 \ell(\boldsymbol{\theta})\big] = \mathbf{E}\big[\nabla \ell(\boldsymbol{\theta}) \nabla \ell(\boldsymbol{\theta})^T\big] \succcurlyeq \mathbf{0}$$

  - 기댓값과 미분의 교환 가능성 하에 성립 (대부분의 흔한 분포에서 참)
  - $\mathbf{H}^{(t)} = \mathbf{I}\big(\boldsymbol{\theta}^{(t)}\big)$로 두면

$$\boxed{\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + s\big[\mathbf{I}\big(\boldsymbol{\theta}^{(t)}\big)\big]^{-1} \nabla \ell\big(\boldsymbol{\theta}^{(t)}\big)}$$

  - 양정치($\succcurlyeq 0$)이므로 line search와 결합하면 descent algorithm

Example: Logistic Regression

- binary $y_i \in \{0, 1\}$, $\mathbf{x}_i \in \mathbb{R}^p$, $y_i \sim$ Bernoulli(p_i)

$$p_i = g^{-1}(\eta_i) = \frac{e^{\eta_i}}{1 + e^{\eta_i}}, \quad \eta_i = \mathbf{x}_i^T \beta = g(p_i) = \log \frac{p_i}{1 - p_i} \quad \text{(logit link)}$$

- log-likelihood와 미분:
- 따라서 이 문제에선 **Newton's method == Fisher scoring**:

$$\begin{aligned} \beta^{(t+1)} &= \beta^{(t)} + s\big(\mathbf{X}^T \mathbf{W}^{(t)} \mathbf{X}\big)^{-1} \mathbf{X}^T \big(\mathbf{y} - \mathbf{p}^{(t)}\big) \\ &= \big(\mathbf{X}^T \mathbf{W}^{(t)} \mathbf{X}\big)^{-1} \mathbf{X}^T \mathbf{W}^{(t)} \mathbf{z}^{(t)} \end{aligned}$$

  - working response $\mathbf{z}^{(t)} = \mathbf{X}\beta^{(t)} + s\big[\mathbf{W}^{(t)}\big]^{-1}\big(\mathbf{y} - \mathbf{p}^{(t)}\big)$
  - 한 번의 Newton iteration = weighted least squares $\min_\beta \sum_i w_i \big(z_i - \mathbf{x}_i^T \beta\big)^2$ 푸는 것
  - → **IRLS (iteratively re-weighted least squares)**
  - **IRLS == Fisher scoring == Newton's method**

Example: Poisson Regression

- count $y_i \in \{0, 1, 2, \ldots\}$, $y_i \sim$ Poisson(λ_i)

$$\lambda_i = e^{\eta_i}, \quad \eta_i = \mathbf{x}_i^T \beta = \log \lambda_i \quad \text{(log link)}$$

- log-likelihood와 미분:

- 역시 **Newton == Fisher scoring == IRLS**, working response $\mathbf{z}^{(t)} = \mathbf{X}\beta^{(t)} + s\left[\mathbf{W}^{(t)}\right]^{-1}\left(\mathbf{y} - \boldsymbol{\lambda}^{(t)}\right)$

```r
# Poisson regression via Fisher scoring (IRLS) + step halving
# deaths ~ quarter, lambda = exp(beta0 + beta1 * quarter)
poissonreg <- function(x, y, maxiter=10, tol=1e-6) {
    beta <- matrix(0, nrow=2, ncol=1); betaold <- beta
    lik <- function(bet) {eta <- bet[1] + bet[2] * x; sum(y * eta - exp(eta))}
    likold <- lik(betaold); iter <- 1; stop <- FALSE
    while ((!stop) && (iter < maxiter)) {
        eta <- beta[1] + x * beta[2]
        w <- exp(eta)            # lambda
        s <- 1.0                 # line search by step halving
        for (i in 1:5) {
            z <- eta + s * (y / w - 1)   # working response
            m <- lm(z ~ x, weights=w)    # weighted least squares
            beta <- as.matrix(coef(m)); curlik <- lik(beta)
            if (curlik > likold) break
            s <- s * 0.5
        }
        if (norm(beta - betaold, "F") < tol) stop <- TRUE
        likold <- curlik; betaold <- beta; iter <- iter + 1
    }
    list(val=as.numeric(beta), iter=iter)
}
```

Generalized Linear Models (GLM)

- 위 두 예의 **IRLS == Fisher scoring == Newton** 는 우연이 아님
- **exponential family**: density

$$p(y \mid \eta, \phi) = \exp\left\{\frac{y\eta - b(\eta)}{a(\phi)} + c(y, \phi)\right\}$$

  - η: natural parameter, ϕ: dispersion parameter
  - mean $\mu = b'(\eta)$; canonical link $g(\cdot) = [b']^{-1}(\cdot)$
  - variance $\mathbf{Var}(Y) = b''(\eta)a(\phi)$

Family	Canonical Link	Variance Function
Normal	$\eta = \mu$	1
Poisson	$\eta = \log \mu$	μ
Binomial	$\eta = \log(\mu/(1-\mu))$	$\mu(1-\mu)$
Gamma	$\eta = \mu^{-1}$	μ^2
Inverse Gaussian	$\eta = \mu^{-2}$	μ^3

- **GLM**: 조건부 평균 $\mu = \mathbf{E}(Y \mid \mathbf{x})$를 strictly increasing link로 모델링

$$g(\mu) = \mathbf{x}^T\beta, \quad \mu = g^{-1}(\mathbf{x}^T\beta) = b'(\eta)$$

- score, Hessian, information ($a(\phi) \equiv 1$ 가정):
  - Fisher scoring = IRLS, weights $w_i = [1/g'(\mu_i)]^2/\sigma_i^2$
- **canonical link**의 경우 ($\mathbf{x}^T\beta = g(\mu) = \eta$):
  - Hessian의 두 번째 항이 사라짐 → Hessian = Fisher information
  - **IRLS == Fisher scoring == Newton's method**, MLE는 convex optimization
- **non-canonical link**: **Fisher scoring != Newton**, 일반적으로 non-convex
  - e.g. probit regression ($p_i = \Phi(\mathbf{x}_i^T\beta)$) — 단, 이 경우는 예외적으로 convex
- R의 `glm()`이 GLM에 대해 Fisher scoring (IRLS) 구현

7.5. Nonlinear Regression: Gauss-Newton Method

- 역사: 1801년 Gauss가 24개 관측으로 6-parameter 궤도를 fitting하여 dwarf planet Ceres의 위치를 재발견 (method of least squares)
- nonlinear least squares (curve fitting):

$$\text{minimize} f(\beta) = \frac{1}{2}\sum_{i=1}^{n}[y_i - \mu_i(\mathbf{x}_i, \beta)]^2$$

  - μ_i가 β에 대해 선형이면 보통의 least squares로 환원
- "score"와 "information":
  - $\mathbf{J}(\beta)^T = [\nabla\mu_1, ..., \nabla\mu_n] \in \mathbb{R}^{p\times n}$
- **Gauss-Newton** (= Fisher scoring): $\mathbf{I}(\beta) = \mathbf{J}^T\mathbf{J}$ 사용 (항상 양반정치)

$$\boxed{\beta^{(t+1)} = \beta^{(t)} - s\left[\mathbf{I}\left(\beta^{(t)}\right)\right]^{-1}\nabla f\left(\beta^{(t)}\right)}$$

- justification:
  1. residual $y_i - \mu_i$가 작거나 μ_i가 거의 선형 → 둘째 항 무시 가능
  2. $y_i \sim N(\mu_i(\mathbf{x}_i, \beta), \sigma^2)$ (σ^2 알려짐)로 두면 $\mathbf{E}[-\nabla^2\ell(\beta)] = \sum_i \nabla\mu_i\nabla\mu_i^T$
  - 사실 Gauss는 이 방법을 정당화하기 위해 정규분포(Gaussian)를 발명함

7.6. Gradient Descent Method

- iteration:

$$\boxed{\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \gamma_t\nabla f\left(\mathbf{x}^{(t)}\right)}$$

  - Newton-type에서 $\mathbf{H}_t = \mathbf{I}$인 특수경우

- idea: iterative linear approximation (1차 테일러 전개)를 compact set 위에서 최소화

$$\min_{\|\Delta\mathbf{x}\|_2 \le 1} \nabla f\big(\mathbf{x}^{(t)}\big)^T \Delta\mathbf{x} \Rightarrow \Delta\mathbf{x} \propto -\nabla f\big(\mathbf{x}^{(t)}\big)$$

- step size는 descent property 유지하도록 선택 (e.g. line search)
  - Newton과 달리 1차 근사는 항상 unbounded below → step size를 반드시 선택해야 함
- pros
  - 각 iteration이 저렴
  - Hessian 유도/계산/저장/역행렬 불필요 → 대규모 문제에 매력적
- cons
  - 느린 수렴 (zigzagging)
  - non-smooth 문제에는 작동 X

Convergence

- 다음 가정 하에:
  1. f convex, differentiable on $\mathbb{R}^d$
  2. f가 L-Lipschitz gradient: $\| \nabla f(\mathbf{x}) - \nabla f(\mathbf{y}) \|_2 \le L \| \mathbf{x} - \mathbf{y} \|_2$
  3. $p^\star = \inf_{\mathbf{x}} f(\mathbf{x}) > -\infty$, $\mathbf{x}^\star$에서 달성

  상수 step size $\gamma_t = \gamma \in (0, 1/L]$에 대해

$$f\big(\mathbf{x}^{(t)}\big) - p^\star \le \frac{1}{2\gamma t} \| \mathbf{x}^{(0)} - \mathbf{x}^\star \|_2^2 = O(1/t)$$

  - sublinear convergence (line search도 유사한 bound)
  - 일반적으로 gradient descent의 최선은 linear convergence (cf. Newton의 quadratic)

Examples

- least squares $f(\beta) = \frac{1}{2} \| \mathbf{X}\beta - \mathbf{y} \|_2^2$
  - $\nabla f(\beta) = \mathbf{X}^T\mathbf{X}\beta - \mathbf{X}^T\mathbf{y}$는 $\| \mathbf{X}^T\mathbf{X} \|_2$-Lipschitz → $\gamma \in (0, 1/\sigma_{\max}(\mathbf{X})^2]$
- logistic regression
  - $\nabla f(\beta) = -\mathbf{X}^T(\mathbf{y} - \mathbf{p})$는 $\frac{1}{4} \| \mathbf{X}^T\mathbf{X} \|_2$-Lipschitz
  - 한 클래스만 있으면 unbounded below 가능 → ridge penalty $\frac{\rho}{2} \| \beta \|_2^2$로 identifiable
- Poisson regression
  - $\nabla f(\beta) = -\mathbf{X}^T(\mathbf{y} - \boldsymbol{\lambda})$는 Lipschitz가 아님 (HW)

7.7. Coordinate Descent

- idea: 좌표별 최소화가 전차원 최소화보다 쉬운 경우가 많음

$$x_j^{(t+1)} = \arg\min_{x_j} f\left(x_1^{(t+1)}, ..., x_{j-1}^{(t+1)}, x_j, x_{j+1}^{(t)}, ..., x_d^{(t)}\right)$$

  - 선형방정식의 Gauss-Seidel과 유사
- block descent: coordinate descent의 벡터 버전 (좌표 대신 블록 $\mathbf{x}_j$ 갱신)
- Q: 왜 목적함수 값이 수렴? → A: descent property
- 주의: 목적함수가 convex여도 coordinate descent가 global minimum이 아닌 suboptimal point로 수렴할 수 있음 (특히 non-smooth한 경우)

8. Numerical Integration

8.1. Goal

- $\int_a^b f(x)dx$, $f : [a, b] \to \mathbb{R}$ 계산
  - 해석적으로 적분 가능한 함수는 소수 → 나머지는 수치적 근사
- 적분이 필요한 이유
  - expectation: $\mathbf{E}[Y] = \int g(x) f_X(x) dx$
  - Bayesian inference: posterior $f_{\Theta \mid X}(\theta \mid x) = \frac{f_{X \mid \Theta}(x \mid \theta) f_\Theta(\theta)}{\int f_{X \mid \Theta}(x \mid \theta') f_\Theta(\theta') d\theta'}$ (분모 적분이 closed form 아닐 수 있음)
  - structured data의 MLE (e.g. GLMM의 random intercept): likelihood/score가 closed form 없는 적분으로 구성
- 두 접근: **quadrature** (Archimedes 이래) / **Monte Carlo** (20세기, 9~11장)

8.2. Newton-Côtes Quadrature

- key idea:
  1. $[a, b]$를 길이 $h = \frac{b-a}{n}$의 **등간격** subinterval $[x_i, x_{i+1}]$로 분할
  2. 각 구간에서 f를 m차 **interpolating polynomial** p_i로 근사
  3. $I_i = \int_{x_i}^{x_{i+1}} f \approx \int_{x_i}^{x_{i+1}} p_i$ (정확히 계산 가능)
  4. $I \approx \sum_{i=0}^{n-1} I_i$
- 차수 m의 선택:
  - $m = 0$: Riemann (rectangular) rule
  - $m = 1$: Trapezoidal rule
  - $m = 2$: Simpson's rule

Riemann Rule

- f를 상수 $f(x_i)$로 근사: $I_i \approx h f(x_i)$
- 정확도: error $= O(n^{-1})$

Trapezoidal Rule

- f를 $(x_i, f(x_i))$, $(x_{i+1}, f(x_{i+1}))$을 잇는 일차함수로 근사

$$I \approx h\left(\frac{1}{2}f(a) + f(x_1) + \cdots + f(x_{n-1}) + \frac{1}{2}f(b)\right) =: T(n)$$

- 정확도: error $= O(n^{-2})$

Simpson's Rule

- f를 $(x_i, f(x_i))$, 중점, $(x_{i+1}, f(x_{i+1}))$을 지나는 이차함수로 근사

$$I_i \approx \frac{h}{6}\left[f(x_i) + 4f\left(\frac{x_i + x_{i+1}}{2}\right) + f(x_{i+1})\right]$$

- 정확도: error $= O(n^{-4})$ (Trapezoidal의 $O(n^{-2})$에서 큰 도약!)
  - 단, Simpson은 점을 **2배** 사용 → 공정한 비교는 동일 점 개수로

```r
trapezoidal <- function(f, a, b, n) {
    h <- (b - a) / n
    xi <- seq.int(a, b, length.out = n + 1); xi <- xi[-c(1, n + 1)]
    h * (0.5 * f(a) + sum(f(xi)) + 0.5 * f(b))
}
```

General m

- Lagrange polynomial $L_{ij}^m(x) = \prod_{k \neq j} \frac{x - x_{ik}^*}{x_{ij}^* - x_{ik}^*}$ 사용 → $p_i(x) = \sum_j f(x_{ij}^*) L_{ij}^m(x)$
- **interpolation theorem**: 서로 다른 $m+1$개의 점을 지나는 m차 이하 다항식은 유일하게 존재
- **Corollary**: f가 m차 이하 다항식이면 $\int_a^b f(x)dx = \sum_{j=0}^m w_j f(x_j)$, $w_j = \int_a^b L_j^m(x)dx$ (정확)

8.3. Gaussian Quadrature

- Newton-Côtes: 점 x_{ij}^*를 **등간격**으로 고정, w_{ij}만 최적화 → m차 다항식까지 exact
- Gaussian: 등간격 제약 제거, w_{ij}와 x_{ij}^*를 **모두** 최적 선택
- **Theorem**: f가 $2m+1$차 이하 다항식이면, $m+1$개의 점/가중치로 $\int_{x_i}^{x_{i+1}} f = \sum_{j=0}^m w_{ij} f(x_{ij}^*)$가 exact ($w_{ij} > 0$)
  - $m+1$개 점으로 $2m+1$차까지 exact → Newton-Côtes 대비 큰 개선

핵심 아이디어: 직교다항식

- $\mathcal{P}^m(a,b)$ (구간 위 m차 이하 다항식)에 inner product $\langle p, q \rangle = \int_a^b p(x)q(x)dx$ 도입 → inner product space
- $\{1, x, \ldots, x^m\}$에 Gram-Schmidt → orthonormal basis = **Legendre polynomial** (uniform 분포에 대응)
- 증명 아이디어: $f = g \cdot q_{m+1} + r$ (다항식 나눗셈), $\langle g, q_{m+1} \rangle = 0$이므로 $\int f = \int r$; 점 x_j를 q_{m+1}의 근으로 잡으면 $f(x_j) = r(x_j)$
  - **Lemma**: q_k의 근은 모두 실수이고 서로 다름
  - 차수 $2m+1$이 최대 (반례: $f(x) = \prod (x - x_j)^2$는 $2m+2$차인데 $\sum w_j f(x_j) = 0$)

Weighted Orthogonal Polynomials

- inner product를 $\langle p, q \rangle = \int_a^b p(x)q(x)w(x)dx$로 바꾸면 w-orthogonal polynomial 생성

orthogonal poly	weight $w(x)$	domain	distribution
Legendre	1	$(-1,1)$	uniform
Laguerre	$x^\alpha e^{-x}$	$(0,\infty)$	gamma
Hermite	e^{-x^2}	$(-\infty,\infty)$	normal
Jacobi	$(1-x)^\alpha(1+x)^\beta$	$(-1,1)$	beta

- 근은 보통 Newton's method로 탐색; 3-term recursion, interlacing property 활용

Gauss-Hermite Quadrature (GHQ)

- $(-\infty,\infty)$ 적분에 Hermite polynomial 사용:

$$\int_{-\infty}^{\infty} f(x)dx = \int_{-\infty}^{\infty}\left(\frac{f(x)}{e^{-x^2}}\right)e^{-x^2}dx \approx \sum_{j=0}^{m} w_j f(x_j)e^{x_j^2}$$

- $\hat{f}(x) = f(x)e^{x^2}$가 다항식에 가까우면 정확
  - f의 mass가 원점에서 멀면 ($\mu \gg 0$ 또는 $\sigma^2 \gg 1$) 점 x_j가 엉뚱한 곳에 → 나쁜 근사
  - **remedy**: change of variable로 적분의 peak를 원점/적절한 scale로 이동 (e.g. mode θ^* 중심으로)

8.4. Laplace Approximation

- $\int_{-\infty}^{\infty} f(x)e^{-ag(x)}dx$를 **최적화**로 대체 ($g$의 최소점 가정 WLOG 0)
- $g(x) = g(0) + \frac{1}{2}g''(0)x^2 + o(x^2)$ 테일러 전개 후

$$\int_{-\infty}^{\infty} f(x)e^{-ag(x)}dx \asymp f(0)e^{-ag(0)}\sqrt{\frac{2\pi}{ag''(0)}} \quad (a\to\infty)$$

  - $a\to\infty$이면 0 근처 기여가 dominate → $N(0, 1/[ag''(0)])$에 대한 적분으로 환원

Applications

- **Stirling's formula**: $\Gamma(t+1) = t^{t+1}\int_0^\infty e^{-tg(z)}dz$, $g(z) = z - \log z$ (최소 $z=1$) → $\Gamma(t+1) \asymp \sqrt{2\pi}t^{t+1/2}e^{-t}$
- **posterior expectation**: n이 크면 log posterior가 mode $\hat{\theta}$ 근처에 sharply peaked → 분자/분모 각각 Laplace 근사
- beta 분포 적률 등에도 적용
- **multidimensional**: Laplace approximation은 자연스럽게 다차원 확장; 반면 quadrature는 점 개수가 차원에 지수적으로 증가 (Monte Carlo가 이를 완화 — 다음 장)

9. Random Number Generation

9.1. Uniform Random Number Generation

- goal: $U_i \overset{iid}{\sim} \text{unif}\ (0,1)$ 생성 — 모든 난수 생성의 기초

- **사실: 컴퓨터에 진짜 난수는 없음** → 난수처럼 보일 뿐 (**pseudo-random**)

Congruential Generator

$$x_{i+1} = ax_i \bmod m, \quad u_i = x_i/m$$

- modulus m: 큰 소수, multiplier a: $2 \le a \le m-1$ - $f : x \mapsto ax \bmod m$은 $\{1, ..., m-1\}$ 위 일대일 대응 (onto) - $x_n = a^n x_0 \bmod m$ → $a^n = 1 \bmod m$이면 수열 반복; 이 n이 **period**, x_0가 **seed**

- **primitive root**: Fermat's little theorem ($a^{m-1} = 1 \bmod m$)로 period $n \le m-1$; $n = m-1$ 이게 하는 a
  - e.g. $m = 2^{31} - 1$ (Mersenne prime), $a = 7^5 = 16807$ → period $2^{31} - 2$ (초기 MATLAB)
- 좋은 RNG: 긴 period + uniform하게 보이는 표본 (표본 크기 $\ll$ period이면 random하게 보임)

Autocorrelation

- 이상적 PRNG: p개씩 묶은 벡터가 p차원 hypercube를 uniform하게 채워야 함
- 그러나 congruential generator는 이전 값에 의존 → correlated, 일부 p에서 저차원 hyperplane에 집중
  - e.g. RANDU ($a = 65539$, $m = 2^{31}$)의 악명 높은 실패

R's RNG

- 기본값 **Mersenne-Twister** (Matsumoto & Nishimura, 1998), period $2^{19937} - 1$; 623차원까지 equidistributed

9.2. Transformation Methods

- 이후 uniform 난수 생성은 해결되었다고 가정

Inverse CDF Method

- inverse CDF: $F^{-1}(u) = \inf\{x : F(x) \ge u\}$ (strictly increasing이면 보통 역함수와 일치)
- **Proposition**: (1) F 연속이면 $U = F(X) \sim \text{unif } (0,1)$; (3) $U \sim \text{unif } (0,1)$이면 $F^{-1}(U)$의 CDF는 F
  - 따라서 U를 뽑아 $F^{-1}(U)$를 계산하면 F를 따르는 표본
- 예시:
  - exponential: $F^{-1}(u) = -\frac{1}{\lambda}\log u$
  - Cauchy: $F^{-1}(u) = \mu - \frac{\beta}{\tan(\pi u)}$
  - discrete uniform $\{1, ..., k\}$: $F^{-1}(u) = \lceil ku \rceil$
  - geometric: $X = \lceil \frac{\log U}{\log(1-p)} \rceil$ ($\lceil \text{Exp } (\lambda) \rceil$, $\lambda = -\log(1-p)$)

Normal Random Numbers

- Φ^{-1}이 closed form 없음 → 별도 방법
- **Box-Muller**: $(X, Y) = (R\cos\Theta, R\sin\Theta)$로 직교→극좌표 변환
  - $R^2 \sim \text{Exp } (1/2)$, $\Theta \sim \text{unif } (0, 2\pi)$ 독립
  - $U, V \sim \text{unif } (0,1)$에서 $R = \sqrt{-2\log U}$, $\Theta = 2\pi V$ → (X, Y) iid $N(0,1)$
- **Marsaglia (polar)**: 단위원에서 (U, V) 추출, $T^2 = U^2 + V^2 \sim \text{unif } (0,1)$ 이용

- $X = \sqrt{-2\log T^2}\frac{U}{T}$, $Y = \sqrt{-2\log T^2}\frac{V}{T}$
- 삼각함수 평가 회피, 대신 평균 $4/\pi$배 더 많은 쌍 필요 (acceptance-rejection)

9.3. Random Numbers by Definition

- 정의/관계를 이용해 다른 분포로부터 생성
  - **Bernoulli**: $X = \mathbb{I}_{\{U \le p\}}$
  - **Binomial**: $\sum_{i=1}^{n} \text{Ber}\ (p)$
  - **negative binomial**: $\sum_{i=1}^{r} \text{Geom}\ (p)$
  - **Poisson**: $\prod_{i=1}^{N} U_i \ge e^{-\lambda} > \prod_{i=1}^{N+1} U_i$인 N (Poisson process의 대기시간 $-\frac{1}{\lambda}\log U_i \sim \text{Exp}\ (\lambda)$ 이용)
  - **chi-square** $\chi^2(\nu) = \sum_{i=1}^{\nu} Z_i^2$; 짝수 ν는 $-2\log\prod U_i$ ($\because \chi^2(\nu) = \text{Gamma}\ (\nu/2, 2)$)
  - **Student's t**: $T = Z/\sqrt{X/\nu}$, $Z \sim N(0,1)$, $X \sim \chi^2(\nu)$
  - ***F***: $F = \frac{X_1/\nu_1}{X_2/\nu_2}$, $X_i \sim \chi^2(\nu_i)$

9.4. Acceptance-Rejection Sampling

- 복잡한 density f에서 표본 추출 (CDF/inverse 사용 불가 시), von Neumann 제안
  1. **envelope** 찾기: 간단한 density g, 상수 c로 $f(x) \le cg(x) =: h(x)$
  2. $X \sim g$ 추출
  3. $U \le \frac{f(X)}{h(X)}$이면 **accept** (독립 $U \sim \text{unif}\ (0,1)$)
  4. 아니면 reject 후 2로
- **왜 작동?** (body $B_f = \{(x,y) : 0 \le y \le f(x)\}$)
  - **Theorem**: $(X, cg(X)U)$는 B_{cg} 위 uniform
  - accept된 (X,Y)는 B_f 위 uniform → X의 주변분포는 f
  - 전체 acceptance ratio $= P\Big(U \le \frac{f(X)}{cg(X)}\Big) = \frac{1}{c}$ → c가 작을수록 효율적
- 예: Marsaglia (단위원, $c = 4/\pi$); gamma 난수 ($0 < \alpha < 1$, 조각별 envelope)

9.5. Multivariate Random Numbers

Batch Methods

- **multinomial**: (1) pmf에서 n번 독립 추출, 또는 (2) 독립 Poisson 추출 후 합이 n이면 조건부가 multinomial
- **multivariate normal** $N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$: $\mathbf{L}^T\mathbf{L} = \boldsymbol{\Sigma}$인 $\mathbf{L}$ 찾아 $\mathbf{L}^T\mathbf{Z} + \boldsymbol{\mu}$, $\mathbf{Z} \sim N(0, \mathbf{I})$
  - $\mathbf{L}$: Cholesky factor 또는 matrix square root $\boldsymbol{\Sigma}^{1/2} = \mathbf{Q}\boldsymbol{\Lambda}^{1/2}\mathbf{Q}^T$
- **multivariate t**: $\mathbf{T} = \frac{1}{\sqrt{\chi^2_\nu/\nu}}\mathbf{X} + \boldsymbol{\mu}$, $\mathbf{X} \sim N(0, \boldsymbol{\Sigma})$

Sequential Sampling

- 연쇄법칙으로 성분별로 조건부 추출:

$$p(x_1, ..., x_p) = p(x_1)\prod_{j=2}^{p} p(x_j \mid x_1, ..., x_{j-1})$$

  - multinomial: $X_1 \sim B(n, p_1)$, $X_2 \mid X_1 \sim B\Big(n - x_1, \frac{p_2}{1-p_1}\Big)$, ⋯
  - MVN: $X_1 \sim N(\mu_1, \sigma_{11})$, 이후 조건부 정규분포 공식 사용

10. Monte Carlo Integration

10.1. Naïve Monte Carlo

- goal: $\int_A g(x)dx$ (또는 $\int_A g(x)w(x)dx$), $g:\mathbb{R}^d \to \mathbb{R}$
  - quadrature를 d차원으로 확장 시 n^d 점 필요 (**curse of dimensionality**)
- 정규화 후 $\int_A g(x)f(x)dx = \mathbf{E}[g(X)]$, $X \sim F$. 대수의 법칙으로

$$\mathbf{E}[g(X)] \approx \hat{I}_n(f) = \frac{1}{n}\sum_{i=1}^{n} g(X_i), \quad X_i \overset{iid}{\sim} F$$

  - **unbiased** estimator, variance $= \frac{1}{n}\operatorname{Var}[g(X)]$
  - CLT로 신뢰구간 $\hat{I}_n \pm z_{1-\alpha/2}\frac{\operatorname{Sd}[g(X)]}{\sqrt{n}}$
- MC integration의 핵심 특징:
  1. error가 공간 차원에 **직접 의존하지 않음**
  2. error가 느린 $O(n^{-1/2})$로 감소
  - quadrature는 1차원에서 $O(n^{-k})$ ($k \geq 2$), d차원에선 $O(n^{-k/d})$ → $d \geq 4$부터 MC가 유리
  - MC의 매력: 구현이 매우 간단 (sample & average)
- 예: $\int_0^1 \frac{1}{\sqrt{x}}dx = 2$이지만 $\operatorname{Var}\left[1/\sqrt{X}\right] = \infty$ → naïve MC 부정확
  - change of variable ($t = \sqrt{x}$)로 피적분함수를 상수에 가깝게 → 정확해짐
  - 일반적으로 피적분함수가 상수에 가까울수록 MC 정확
- Buffon's needle: $P(\textbf{교차}) = \frac{2L}{\pi D}$ → π의 MC 추정

10.2. Variance Reduction Methods

- $O(n^{-1/2})$ rate는 유지, 상수 $\sqrt{\operatorname{Var}[g(X)]}$를 줄이는 것이 목표

Importance Sampling

- $I = \mathbf{E}_f[g(X)] = \mathbf{E}_h\left[\frac{g(X)f(X)}{h(X)}\right]$ (다른 density h, $gf \neq 0$인 곳에서 $h > 0$)

$$\hat{I}_{\text{imp}} = \frac{1}{n}\sum_{i=1}^{n}\frac{g(X_i)f(X_i)}{h(X_i)}, \quad X_i \overset{iid}{\sim} h$$

- 둘 다 unbiased; $\hat{I}_{\text{imp}}$의 분산이 더 작을 조건 (Cauchy-Schwarz)
  - **최적 $h \propto |g(x)|f(x)$**: $g \geq 0$이면 분산을 0으로 만들 수 있음
  - 단 $\int |g|f$는 미지 → h를 $|g|f$에 비례에 가깝게 잡으면 분산 감소
- 예: 정규분포 tail 확률 $\pi(c) = \Phi(-c)$ — $N(c,1)$로 importance sampling 시 큰 c에서 극적 개선

Antithetic Variable Method

- unbiased estimator I_1, I_2가 **음의 상관**이면 $\operatorname{Var}\left[\frac{I_1+I_2}{2}\right] \leq \frac{1}{4}(\operatorname{Var}[I_1] + \operatorname{Var}[I_2])$
- **Proposition**: g_1, g_2가 같은 방향으로 단조이면 $\operatorname{Cov}(g_1(X), g_2(X)) \geq 0$
- 구현: $I_1 = g(F^{-1}(U))$, $I_2 = g(F^{-1}(1-U))$ (같은 U) → $g(F^{-1}(u))$는 비감소, $g(F^{-1}(1-u))$는 비증가 → $\operatorname{Cov} \leq 0$
  - 또는 대칭성 이용: f가 μ 대칭이면 $I_2 = g(2\mu - X_i)$
- 예: $\int_0^1 e^u du$ — naïve $\frac{0.242}{n}$ vs antithetic $\frac{0.0039}{n}$

Control Variable Method

- 기댓값 μ를 아는 Y 사용: $\hat{I} = f(X) - c(Y - \mu)$ (모든 c에 unbiased)
  - 최적 $c^\star = \frac{\mathrm{Cov}\,(f(X),Y)}{\mathrm{Var}\,[Y]}$, 최소 분산 $\mathrm{Var}\,[f(X)] - \frac{\mathrm{Cov}\,(f(X),Y)^2}{\mathrm{Var}\,[Y]}$
  - $f(X)$와 Y가 강하게 상관될수록 분산 크게 감소 (c는 회귀로 추정 가능)

11. Markov Chain Monte Carlo

11.1. Introduction

- MCMC: 극한분포가 목표분포 π와 같은 Markov chain을 생성하여 π를 simulate하는 방법

11.2. Review: Markov Chain

- (이산시간·이산상태) Markov chain: $\Pr(X_{n+1} = j \mid X_n = i, \ldots) = \Pr(X_{n+1} = j \mid X_n = i) = P_{ij}$ (Markov property)
  - time-homogeneous: $P(n) = P$; transition matrix $P = \big(P_{ij}\big)$, $\sum_j P_{ij} = 1$
- **Chapman-Kolmogorov**: $P^{(m+n)} = P^{(m)} P^{(n)} \to P^{(n)} = P^n$
- **communication**: $i \leftrightarrow j$; **irreducible**: 단일 communication class
- **stationary distribution**: $\pi = \pi P$, $\mathbf{1}^T \pi = 1$, $\pi \geq 0$ (= invariant); 극한분포가 존재하면 stationary
- 특성:
  - **period** $d(j) = \gcd\big\{n : P_{jj}^n > 0\big\}$; **aperiodic**: 모든 $d(i) = 1$
  - **recurrent** ($\tau_{ii} < \infty$ a.s.): positive ($\mathbf{E}[\tau_{ii}] < \infty$) / null; otherwise **transient**
  - **ergodic** = aperiodic + positive recurrent (유한 상태공간 + irreducible이면 ergodic)
- **limit theorem**: irreducible + ergodic이면 $\lim_n P_{ij}^{(n)} = \pi_j$ (i 무관), π 유일, $\pi_j = 1/\mathbf{E}\big[\tau_{jj}\big]$
- **ergodic theorem**: $\frac{1}{n}\sum_{k=1}^n g(X_k) \to \mathbf{E}_\pi[g(X)]$ a.s.
  - i.i.d. SLLN을 Markov 수열로 확장 → MC integration을 비-i.i.d. 수열로 확장 = **MCMC**
- **MCMC 전략**: stationary distribution이 목표 f인 irreducible, aperiodic, positive recurrent chain 구성
- **detailed balance** (충분조건): $\pi_i P_{ij} = \pi_j P_{ji}$ → π가 stationary (reversible chain)
- 연속상태공간: $P_{ij} \to$ transition density $K(x,y)$, detailed balance $\pi(x)K(x,y) = \pi(y)K(y,x)$

11.3. Metropolis-Hastings Algorithm

- 각 t에 대해:
  1. proposal $g\big(\cdot \mid X^{(t)}\big)$에서 후보 $\tilde{X}$ 추출
  2. MH ratio $R\big(X^{(t)}, \tilde{X}\big) = \frac{f(\tilde{X})\, g(X^{(t)} \mid \tilde{X})}{f(X^{(t)})\, g(\tilde{X} \mid X^{(t)})}$
  3. acceptance probability $\alpha = \min\{R, 1\}$로 $X^{(t+1)} = \tilde{X}$, 아니면 $X^{(t+1)} = X^{(t)}$
  - Metropolis (1953): symmetric proposal $g(u \mid v) = g(v \mid u)$

- **stationary distribution**: kernel $K(x,y) = \alpha(x,y)g(y \mid x) + (1 - r(x))\delta(y - x)$
  - $f(x)\alpha(x,y)g(y \mid x) = \min\{f(x)g(y \mid x), f(y)g(x \mid y)\}$ 대칭 → **detailed balance** 만족 → f가 stationary
  - irreducibility/aperiodicity는 g에 의존, 사용자가 확인 (aperiodicity는 보통 rejection 확률 > 0이라 성립)
- **posterior sampling**: 정규화상수 c가 미지여도 MH ratio에서 상쇄됨 → MCMC의 강점
  - independent chain ($g = $ prior)이면 ratio가 likelihood 비율로 단순화

11.4. Gibbs Sampler

- 다차원 목표분포에서 **단변량 조건부분포**(흔히 closed form)를 순차적으로 추출

$$X_i^{(t+1)} \sim f_{X_i \mid X_{-i}}\left(\cdot \mid x_1^{(t+1)}, ..., x_{i-1}^{(t+1)}, x_{i+1}^{(t)}, ..., x_p^{(t)}\right)$$

  - cf. Gauss–Seidel

- 예: bivariate normal — $X_1 \mid X_2 \sim N(\rho x_2, 1 - \rho^2)$, $X_2 \mid X_1 \sim N(\rho x_1, 1 - \rho^2)$ 교대 추출
- **MH와의 관계**: Gibbs는 proposal $g_i(x \mid y) = f_{X_i \mid X_{-i}}(x_i \mid y_{-i})\delta(x_{-i} - y_{-i})$를 쓰는 MH
  - MH ratio $R = 1$ → proposal이 **항상 accept**되는, time-varying proposal의 MH sampler
- 예: Ising model — 정규화상수 Z가 2^p개 항의 합이라 직접 계산 불가하지만, Gibbs 조건부는 logistic 형태로 간단

11.5. Practical Considerations

1. **mixing**: MC가 언제 stationary state에 도달? 언제 멈추나?
2. **burn-in**: 초기 일부 표본 폐기 — 얼마나?
3. i.i.d. 표본:
   - 본질적으로 표본이 correlated
   - 이상적으로는 N개 병렬 chain을 돌려 종단값만 사용
   - **thinning**: $Y_m = X_{mk}$ ($k > 1$)로 subsample하여 상관 감소