Statistical Computation

Derek Li

Contents

1 Review

1.1 Complex Number

Definition 1.1. A complex number z consists of two component, real and imaginary:

$$z = x + \iota y$$

where $\iota = \sqrt{-1}$.

Property 1.1. If $z_1 = x_1 + \iota y_1, z_2 = x_2 + \iota y_2$ then

$$\begin{aligned} z_1 + z_2 &= (x_1 + x_2) + \iota(y_1 + y_2) \\ z_1 z_2 &= (x_1 x_2 - y_1 y_2) + \iota(x_1 y_2 + x_2 y_1) \end{aligned}$$

Property 1.2. $\exp(\iota\theta) = \cos(\theta) + \iota\sin(\theta)$.

Property 1.3. $z = x + \iota y = r\exp(\iota\theta)$ where $r = |z| = \sqrt{x^2 + y^2}, x = r\cos(\theta), y = r\sin(\theta)$.

Property 1.4. $\exp(\iota(\theta_1 + \theta_2)) = \cos(\theta_1 + \theta_2) + \iota\sin(\theta_1 + \theta_2)$.

1.2 Markov Chain

Definition 1.2 (Transition Density)**.** Transition density $q(\mathbf{x}, \mathbf{y})$ is the conditional density of $\mathbf{X}_i$ given $\mathbf{X}_{i-1} = \mathbf{x}$, i.e.,

$$q(\mathbf{x}, \mathbf{y}) = q(\mathbf{x} \to \mathbf{y}) = q(\mathbf{y}|\mathbf{x})$$

Definition 1.3 (Stationary)**.** The Markov chain is stationary with stationary (invariant) density $f(\mathbf{x})$ if

$$f(\mathbf{y}) = \int \cdots \int q(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) \mathrm{d}\mathbf{x}$$

Definition 1.4 (Reversibility)**.** A transition density $q(\mathbf{x}, \mathbf{y})$ will have $f(\mathbf{x})$ as its stationary density if we have

$$f(\mathbf{x})q(\mathbf{x}, \mathbf{y}) = f(\mathbf{y})q(\mathbf{y}, \mathbf{x}) \text{ (Reversibility condition)}$$

1.3 Linear Algebra

Definition 1.5 (Lower Triangular Matrix)**.** We define lower triangular matrix to be

$$L = \begin{pmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \cdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix}$$

where $a_{ij} = 0$ for $i < j$.

Definition 1.6 (Upper Triangular Matrix)**.** We define upper triangular matrix to be

$$U = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{pmatrix}$$

where $a_{ij} = 0$ for $i > j$.

Note. For upper and lower triangular matrices, A^{-1} exists iff $a_{11}, \cdots, a_{nn}$ are all non-zero.

Example 1.1. Suppose we solve $A\mathbf{x} = \mathbf{b}$ for lower triangular matrix A, where $\mathbf{b} = (b_1 \quad \cdots \quad b_n)^T$ and $\mathbf{x} = (x_1 \quad \cdots \quad x_n)^T$ Then

$$\begin{aligned}
x_1 &= \frac{b_1}{a_{11}} \\
x_2 &= \frac{b_2 - a_{21}x_1}{a_{22}} \\
x_3 &= \frac{b_3 - a_{31}x_1 - a_{32}x_2}{a_{33}} \\
&\vdots \\
x_n &= \frac{b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1}}{a_{nn}}
\end{aligned}$$

Note 1. The algorithm for upper triangular matrix is similar.

Note 2. In R, we use `backsolve` and `forwardsolve`.

Definition 1.7 (Norm)**.** Suppose $\mathbf{x} = (x_1 \quad \cdots \quad x_n)^T$ is a vector. A norm $\|\mathbf{x}\|$ satisfies the following conditions:

1. $\|\mathbf{x} + \mathbf{y}\| \leqslant \|\mathbf{x}\| + \|\mathbf{y}\|$.
2. $\|a\mathbf{x}\| = |a|\|\mathbf{x}\|$.
3. $\|\mathbf{x}\| = 0$ implies $\mathbf{x} = \mathbf{0}$.

Note. $\|\mathbf{x}\|$ gives a measure of the size or length of $\mathbf{x}$.

Example 1.2 (General L_p Norm)**.**

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}, p \geqslant 1$$

Example 1.3 (Euclidean Norm)**.**

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} x_i^2\right)^{1/2}$$

Example 1.4 (Manhattan Distance)**.**

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$$

Example 1.5 (L_∞ Norm)**.**

$$\|\mathbf{x}\|_\infty = \max_{1 \leqslant i \leqslant n} |x_i|$$

Definition 1.8 (Frobenius Norm)**.** The Frobenius norm of an $m \times n$ matrix is

$$\|A\|_F = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2\right)^{1/2}$$

Definition 1.9 (L_p Norm for Matrix)**.** Suppose A is an $m \times n$ matrix. We define the L_p norm of A as

$$\|A\|_p = \sup_{\mathbf{x}} \frac{\|A\mathbf{x}\|_p}{\|\mathbf{x}\|_p} = \sup_{\mathbf{x}:\|\mathbf{x}\|_p = 1} \|A\mathbf{x}\|_p$$

Property 1.5.

1. $\|AB\|_p \leqslant \|A\|_p \|B\|_p$.
2. $\|A^k\|_p \leqslant \|A\|_p^k$.
3. $\|I\|_p = \|AA^{-1}\|_p = 1 \Rightarrow \|A^{-1}\|_p \geqslant \dfrac{1}{\|A\|_p}$.
4. For any vector $\mathbf{x}$, $\|A\mathbf{x}\|_p \leqslant \|A\|_p \|\mathbf{x}\|_p$.
5. When $p = 2$,

$$\|A\|_2 = \sqrt{\text{Maximum eigenvalue of } A^T A}$$

Note. If A is symmetric then $\|A\|_2$ is the maximum absolute eigenvalue of A.

Example 1.6 (L_∞ Norm for Matrix)**.** Consider vectors $\mathbf{x}$ whose elements are all ± 1, i.e., $\|\mathbf{x}\|_\infty = 1$. We maximize $\|A\mathbf{x}\|_\infty$ by taking $\mathbf{x}$ so that $\sum\limits_{j=1}^{n} a_{ij} x_j$ is maximized and

$$\|A\|_\infty = \max_{1 \leqslant i \leqslant m} \sum_{j=1}^{n} |a_{ij}|$$

Example 1.7 (L_1 Norm for Matrix)**.** Consider vectors $\mathbf{x}$ whose elements are one 1 and $(n-1)$ 0s, i.e., $\|\mathbf{x}\|_1 = 1$. $A\mathbf{x}$ picks out one column of A and

$$\|A\|_1 = \max_{1 \leqslant j \leqslant n} \sum_{i=1}^{m} |a_{ij}|$$

Theorem 1.1. Suppose A is an $n \times n$ matrix with real-valued eigenvalues $\lambda_1, \cdots, \lambda_n$. Then

$$\max_{1 \leqslant k \leqslant n} |\lambda_k| \leqslant \|A\|_p, p \geqslant 1$$

Proof. Suppose that $A\mathbf{v} = \lambda_k \mathbf{v}$ where $\|\mathbf{v}\|_p = 1$. Then for all $k = 1, \cdots, n$,

$$|\lambda_k| = \|A\mathbf{v}\|_p \leqslant \|A\|_p$$

□

Note. The result holds if there are complex-valued eigenvalues where $|\lambda_k|$ is the modulus if λ_k is complex-valued.

Definition 1.10 (Diagonally Dominant Matrix)**.** An $n \times n$ matrix A is (row) diagonally dominant if

$$|a_{ii}| \geqslant \sum_{j \neq i} |a_{ij}|, i = 1, \cdots, n$$

If $\geqslant$ is replaced by $>$ then A is strictly diagonally dominant.

Property 1.6. A strictly diagonally dominant matrix is invertible.

Theorem 1.2 (Gershgorin Circle Theorem)**.** Suppose A is an $n \times n$ matrix with elements $\{a_{ij}\}$. Define $r_i = \sum\limits_{j \neq i} |a_{ij}|$ and

$$C_i = \{z \in \mathbb{C} : |z - a_{ii} \leqslant r_i|\}$$

which is a circle on the complex plane centered at a_{ii} with radius r_i. If λ is an eigenvalue of A, then $\lambda \in C_i$ for some i.

Theorem 1.3. Suppose A and B are matrices s.t. both AB and BA are well-defined, then

$$\mathrm{trace}(BA) = \mathrm{trace}(AB)$$

Example 1.8. If $\mathbf{u}$ and $\mathbf{v}$ are vectors then

$$\mathbf{u}^T A\mathbf{v} = \mathrm{trace}(A\mathbf{v}\mathbf{u}^T) = \mathrm{trace}(\mathbf{v}\mathbf{u}^T A)$$

Definition 1.11 (Determinant)**.** Suppose A is $n \times n$ matrix with eigenvalues $\lambda_1, \cdots, \lambda_n$, the determinant of A is

$$\det(A) = \prod_{i=1}^{n} \lambda_i$$

Theorem 1.4. Suppose A is symmetric positive definite, then $\lambda_1, \cdots, \lambda_n$ are strictly positive. We have

$$\det(A) = \exp(\mathrm{trace}[\ln(A)])$$

1.3.1 Application: Condition Numbers

Theorem 1.5. If $\|B\|_p < 1$ for some p then

$$(I - B)^{-1} = I + B + B^2 + \cdots = \sum_{k=0}^{\infty} B^k$$

We use $(A+E)^{-1} - A^{-1}$ to check the sensitivity of A^{-1} to round-off error, where E is small relative to A.

We have

$$\begin{aligned}(A+E)^{-1} &= A^{-1}(I + EA^{-1})^{-1} \\ &= A^{-1}[I - EA^{-1} + (EA^{-1})^2 + \cdots] \\ &\approx A^{-1} - A^{-1}EA^{-1}\end{aligned}$$

and thus

$$(A+E)^{-1} - A^{-1} \approx -A^{-1}EA^{-1}$$

Then

$$\|(A+E)^{-1} - A^{-1}\|_p \approx \|A^{-1}EA^{-1}\|_p \leqslant \|A^{-1}\|_p^2 \|E\|_p$$

and

$$\frac{\|(A+E)^{-1} - A^{-1}\|_p}{\|A^{-1}\|_p} \leqslant \|A^{-1}\|_p \|E\|_p = \kappa_p(A)\frac{\|E\|_p}{\|A\|_p}$$

where $\kappa_p(A) = \|A\|_p\|A^{-1}\|_p \geqslant 1$ is called the condition number of A. If the condition number of A is large then the numerical solution of $A\mathbf{x} = \mathbf{b}$ may be unstable.

Example 1.9. Let

$$A = \begin{pmatrix} 1 & 1-\varepsilon \\ 1+\varepsilon & 1 \end{pmatrix}$$

then

$$A^{-1} = \begin{pmatrix} \varepsilon^{-2} & \varepsilon^{-1} - \varepsilon^{-2} \\ -\varepsilon^{-1} - \varepsilon^{-2} & \varepsilon^{-2} \end{pmatrix}$$

for $\varepsilon > 0$ and small. We have $\|A\|_1 = \|A\|_\infty = 2 + \varepsilon$ and $\|A^{-1}\|_1 = \|A^{-1}\|_\infty = 2\varepsilon^{-2} + \varepsilon^{-1}$. Thus

$$\kappa(A) = 4\varepsilon^{-2} + 4\varepsilon^{-1} + 1 \approx 4\varepsilon^{-2}$$

for small ε.

1.3.2 Gram-Schmidt Orthogonalization

Given vectors $\mathbf{v}_1, \cdots, \mathbf{v}_r$, suppose we want to find orthonormal vectors $\mathbf{q}_1, \cdots, \mathbf{q}_r$ with the same span, i.e., if $\mathbf{x} = \sum_{i=1}^{r} a_i \mathbf{v}_i$, then

$$\mathbf{x} = \sum_{i=1}^{r} b_i \mathbf{q}_i$$

for some $b_1, \cdots, b_r$.

The Gram-Schmidt algorithm is:

1. Define $\mathbf{u}_1 = \mathbf{v}_1$.
2. For $i = 2, \cdots, r$, define

$$\mathbf{u}_i = \mathbf{v}_i - \sum_{j=1}^{i-1} \frac{\mathbf{v}_i^T \mathbf{u}_j}{\mathbf{u}_j^T \mathbf{u}_j} \mathbf{u}_j$$

3. Set $\mathbf{q}_i = \dfrac{\mathbf{u}_i}{\|\mathbf{u}_i\|}$ for $i = 1, \cdots, r$.

2 Basics

2.1 Floating Point

2.1.1 Floating Point Representation

Definition 2.1. A ***floating point number*** is represented by three components: (S, F, E) where S is the sign of the number (± 1), F is a fraction (lying between 0 and 1), E is an exponent. S, F, E are all represented as binary digits (bits). The ***floating point representation*** of x, $\mathrm{fl}(x)$ is

$$\mathrm{fl}(x) = S \times F \times 2^E$$

Note. x and $\mathrm{fl}(x)$ need not be the same, since $\mathrm{fl}(x)$ is a binary approximation to x, and there are only a finite number of floating point numbers.

2.1.2 Round-Off Error

Mathematical operations introduce further approximation errors

$$f(\mathrm{fl}(x)) = f(x + \varepsilon) \approx f(x) + \varepsilon f'(x)$$

and the goal is to make the round-off error $|f(x) - \mathrm{fl}(f(\mathrm{fl}(x)))|$ as small as possible.

2.1.3 Machine Epsilon and Other Constants

For a given real number x, we have

$$|\mathrm{fl}(x) - x| \leqslant U|x| \text{ or } \mathrm{fl}(x) = x(1 + u), |u| \leqslant U$$

where U is ***machine epsilon*** or ***machine unit***. U is machine dependent but very small. In R, $U = 2^{-52} = 2.220 \times 10^{-16}$.

Other machine dependent constants include:

1. The minimum and maximum positive floating point numbers: $x_{\min} = 2^{-1022} = 2.225 \times 10^{-308}$ and $x_{\max} = 2^{1024} - 1 = 1.798 \times 10^{308}$.
2. The maximum integer: $2147383647 = 2^{31} - 1$.

2.1.4 Overflow and Underflow Error

Definition 2.2. If the result of a floating point operation exceeds $x_{\max}$, then the value returned is `Inf`.

Note. `Inf` indicates an ***overflow error***.

Definition 2.3. If the result of a floating point operation is undefined then `NaN` is returned.

Definition 2.4 (Underflow Error)**.** An underflow error occurs when the result of a floating point calculation is smaller (in absolute value) than $x_{\min}$.

Note. There are two possible outcomes: an error is reported or an exact 0 is returned. The latter outcome may cause problems in subsequent computations (e.g., division by 0).

Note. There are some ways to avoid overflow and underflow errors:

1. Use logarithmic scale: Changes multiplication/division into addition/subtraction, e.g., `lgamma`, `lfactorial`, `lchoose`.
2. Use series expansions (e.g., Taylor series).

Example 2.1. For x close to 0, $\dfrac{\exp(x)-1}{x} \approx 1$. Naive computation of $\dfrac{\exp(x)-1}{x}$ is problematic for x close to 0 due to possible round-off and underflow errors:

$$\frac{\mathrm{fl}(\exp(x)-1)}{\mathrm{fl}(x)} \neq \frac{\exp(x)-1}{x}$$

We solve the problem by using a series approximation, for $|x| \leqslant \varepsilon$,

$$\frac{\exp(x)-1}{x} = \frac{x + x^2/2 + x^3/6 + \cdots}{x} = 1 + \frac{x}{2} + \frac{x^2}{6} + \cdots$$

2.1.5 Catastrophic Cancellation

Suppose $z_1 = g_1(x_1, \cdots, x_n)$ and $z_2 = g_2(x_1, \cdots, x_n)$. We want to compute $y = z_1 - z_2$. What we actually compute is

$$y^* = \mathrm{fl}(\mathrm{fl}(z_1) - \mathrm{fl}(z_2))$$

where $\mathrm{fl}(z_1) = z_1(1+u_1)$ and $\mathrm{fl}(z_2) = z_2(1+u_2)$. We have

$$\mathrm{fl}(z_1) - \mathrm{fl}(z_2) = \underbrace{z_1 - z_2}_{y} + \underbrace{z_1u_1 - z_2u_2}_{\text{error}}$$

If z_1 and z_2 are large but $y = z_1 - z_2$ is small then the magnitude of the error may be larger than the magnitude of y - ***catastrophic cancellation***.

2.2 Sparse Matrices

Definition 2.5 (Sparse Matrix)**.** We say an $n \times n$ matrix is sparse if it has $k \times n$ non-zero elements where $k \ll n$.

Note 1. An $n \times n$ matrix needs at least n non-zero elements to be invertible.

Note 2. Sparse matrices are useful because we need only store non-zero elements and their row and column indices; multiplication by and addition to 0 are free operations.

2.3 Application: Computation of Probability Distributions

Question: Suppose X_i are independent discrete r.v.s. taking values $0, \cdots, l$ with

$$P(X_i = x) = p(x), x = 0, \cdots, l$$

Define $S = X_1 + \cdots + X_n$ and find the probability distribution of S.

2.3.1 Brute Force Approach

Start with $n = 2$ and proceed inductively:

$$p_2(x) := P(X_1 + X_2 = x) = \sum_{y=0}^{x} P(X_1 = y, X_2 = x - y)$$

$$p_3(x) := P(X_1 + X_2 + X_3 = x) = \sum_{y=0}^{x} P(X_1 + X_2 = y, X_3 = x - y)$$

$$\vdots$$

$p_k(x)$ requires $x+1$ multiplications and to evaluate $p_k(x)$ for $x = 0, \cdots, kl$, we need

$$N(k) = \sum_{x=0}^{kl} (x+1) \approx \frac{(kl)^2}{2} \text{ multiplications}$$

Thus the total number of multiplications is

$$\sum_{k=2}^{n} N(k) \approx \frac{n^3 l^2}{6} = O(n^3 l^2)$$

2.3.2 Probability Generating Function

Definition 2.6 (Probability Generating Function)**.** If X is a discrete r.v. taking values $0, 1, \cdots$, then its probability generating function is

$$\phi(t) = \mathbb{E}[t^X] = \sum_{x=0}^{\infty} P(X = x) t^x$$

Note. If X takes values $0, \cdots, l$, then $P(X = x)$ can be recovered from evaluating $\phi(t)$ at $l+1$ distinct (non-zero) points $t_0, \cdots, t_l$.

If $\phi(t) = \mathbb{E}[t^{X_i}]$, then the probability generating function of S is

$$\mathbb{E}[t^S] = \mathbb{E}[t^{X_1 + \cdots + X_n}] = [\phi(t)]^n$$

Thus we can recover $P(S = x)$ for $x = 0, \cdots, nl$ by evaluating $[\phi(t)]^n$ at $t_0, \cdots, t_{nl}$. We have $nl+1$ linear equations in $nl+1$ unknowns, and solving typically requires $O(n^3 l^3)$ operations, which is slower than the brute force approach.

2.3.3 Discrete Fourier Transform (DFT)

A choice for $t_0, \cdots, t_{nl}$ are complex exponentials

$$t_j = \exp\left(-2\pi\iota \frac{j}{nl+1}\right), j = 0, \cdots, nl$$

where $\iota = \sqrt{-1}$. Since $p(x) = 0$ for $x = l+1, \cdots, nl$, we have

$$\phi(t_j) = \sum_{x=0}^{l} p(x) \exp\left(-2\pi\iota \frac{jx}{nl+1}\right) = \sum_{x=0}^{nl} p(x) \exp\left(-2\pi\iota \frac{jx}{nl+1}\right)$$

$\phi(t_0), \cdots, \phi(t_{nl})$ is the ***discrete Fourier transform*** (DFT) of $p(0), \cdots, p(nl)$, and thus, the DFT of $P(S = 0), \cdots, P(S = nl)$ is $[\phi(t_0)]^n, \cdots, [\phi(t_{nl})]^n$. Hence, given $\phi(t_0), \cdots, \phi(t_{nl})$, we can compute the probability distribution of S using the inverse DFT:

$$P(S = x) = \frac{1}{nl+1} \sum_{j=0}^{nl} [\phi(t_j)]^n \exp\left(2\pi\iota \frac{jx}{nl+1}\right), x = 0, \cdots, nl$$

Naive computation of $P(S = x)$ using DFT requires $O(n^3 l^2)$ multiplications; but with divide-and-conquer algorithm, we can reduce the number of multiplications by a factor of n.

In R, if `x` is a vector of length n we can compute its DFT with `fft(x)` and the inverse DFT with `fft(tx, inv=T) / length(x)`:

```
probs = # The vector for P(X=x)
dft = fft(probs)
dft.s = dtf^n # S=X1+...+Xn
idft.s = fft(dft.s, inv=T) / length(probs)
Re(idft.s) # Real component of idft.s, or P(S=x)
```

Note. fft is the fast Fourier transform, which is an efficient algorithm for computing the DFT when the length of the sequence is a product of small primes.

2.4 Application: Image Processing

Question: We observe an image denoted by $x(i,j).i = 1, \cdots, m, j = 1, \cdots, n$, where (i,j) denotes a pixel location. We want:

1. Denoising: Think of $\{x(i,j)\}$ as a image corrupted by noise

$$x(i,j) = \underbrace{s(i,j)}_{\text{True}} + \underbrace{\varepsilon(i,j)}_{\text{Noise}}$$

2. Compression: Approximate $x(i,j)$ by $x^*(i,j)$ where

$$x^*(i,j) = \sum_{k=1}^{p} \beta_k \phi_k(i,j)$$

where $p \ll m \times n$ and $\phi_1, \cdots, \phi_p$ are known functions.

2.4.1 Transformation

Define X to be the $m \times n$ matrix whose elements are $x(i,j)$. Define orthogonal matrices H_1 $(m \times m)$ and H_2 $(n \times n)$ and define $\widehat{X} = H_1 X H_2$, which has the same dimensions as X. Since for orthogonal matrix H, $H^{-1} = H^T$ and so $X = H_1^T \widehat{X} H_2^T$. Assume the noisy image model $X = S + E$, if H_1 and H_2 are chosen appropriately,

$$\widehat{X} = \underbrace{H_1 S H_2}_{\text{Sparse}} + \underbrace{H_1 E H_2}_{\approx 0}$$

Therefore,

1. Denoising: Given $\widehat{X}$, find a transformation $\widehat{X} \mapsto T(\widehat{X})$ and define the denoised image

$$X_{\text{dn}} = H_1^T T(\widehat{X}) H_2^T$$

where we assume the smallest elements of $\widehat{X}$ are due to noise and set these equal to 0

$$T(\widehat{X})(i,j) = 0, |\widehat{X}(i,j)| \leqslant \text{Threshold}$$

2. Compression: The same idea is used for compression: for some T,

$$X_{\text{c}} = H_1^T T(\widehat{X}) H_2^T$$

Note. T is usually defined more deterministically. The form of T depends on the amount of compression and the type of image.

2.4.2 Hadamard Matrices and Walsh-Hadamard Transform

Definition 2.7 (Hadamard Matrix)**.** A Hadamard matrix is an $n \times n$ matrix whose elements are all ± 1 with orthogonal rows s.t. $HH^T = nI$.

Note 1. $H^{-1} = \dfrac{H^T}{n}$.

Note 2. Hadamard matrices only exist if $n = 1, n = 2$, or n is a multiple of 4.

Note 3. We focus on the case where $n = 2^k$ since it is simple to construct and we can write the Hadamard matrix as a product of sparse matrices. We start with the trivial 1×1 Hadamard matrix $H_1 = 1$, and then define $H_2, H_4, H_8, \cdots$ recursively:

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$H_{2^k} = \begin{pmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{pmatrix}$$

for $k = 2, 3, \cdots$.

Note 4. H_2 is symmetric and so H_{2^k} is symmetric and thus $H_{2^k}^{-1} = \dfrac{H_{2^k}}{2^k}$.

Definition 2.8 (Kronecker Product)**.** Given arbitrary matrices A and B, the Kronecker product $A \otimes B$ is

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{pmatrix}$$

for an $m \times n$ matrix A.

Property 2.1. Assume below that any matrix sums, products or inverses are well-defined.

1. $A \otimes (B + C) = (A \otimes B) + (A \otimes C)$.
2. $(B + C) \otimes A = (B \otimes A) + (C \otimes A)$.
3. $A \otimes (B \otimes C) = (A \otimes B) \otimes C$.
4. $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$.
5. $(A \otimes B)^T = A^T \otimes B^T$.
6. $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$.

Note. For Hadamard matrices, $H_{2^k} = H_2 \otimes H_{2^{k-1}}$. We rewrite it as $H_{2^k} = (H_2 I_2) \otimes (I_{2^{k-1}} H_{2^{k-1}})$ and using the property, we have

$$H_{2^k} = (H_2 \otimes I_{2^{k-1}})(I_2 \otimes H_{2^{k-1}})$$

Repeating the process with $H_{2^{k-1}}, H_{2^{k-2}}, \cdots$, we get

$$H_{2^k} = \underbrace{(H_2 \otimes I_{2^{k-1}})(I_2 \otimes H_2 \otimes I_{2^{k-2}})(I_4 \otimes H_2 \otimes I_{2^{k-3}}) \cdots (I_{2^{k-1}} \otimes H_2)}_{k=\log_2(n) \text{ terms}}$$

Definition 2.9 (Walsh-Hadamard Transform)**.** Given an $n \times n$ Hadamard matrix H and a vector $\mathbf{x}$ of length n, we define its Walsh-Hadamard transform by $\widehat{\mathbf{x}} = H\mathbf{x}$.

Note 1. Given the W-H transform, we can recover $\mathbf{x}$

$$\mathbf{x} = \frac{1}{n} H^T \widehat{\mathbf{x}}$$

Note 2. If $n = 2^k$, since $H = H^T$, then

$$\mathbf{x} = \frac{1}{n} H \widehat{\mathbf{x}}$$

2.5 Application: Denoising

Question: Suppose we observe $\mathbf{x} = (x_1, \cdots, x_n)^T$ where we assume that

$$\mathbf{x} = \mathbf{s} + \mathbf{e} = \text{Signal} + \text{Noise}$$

We want to recover or estimate the signal $\mathbf{s}$.

2.5.1 Assumption

Assume $\mathbf{s}$ is structured so that its W-H transform $\widehat{\mathbf{s}} = H\mathbf{s}$ contains mostly 0s

$$\widehat{\mathbf{x}} = H\mathbf{x} = \underbrace{H\mathbf{s}}_{\text{Sparse}} + \underbrace{H\mathbf{e}}_{\text{Relatively small}}$$

2.5.2 Thresholding

We shrink smaller components of $\widehat{\mathbf{x}}$ towards 0, and then estimate $\mathbf{s}$ by the inverse W-H transform of the thresholded $\widehat{\mathbf{x}}$. Thresholded W-H transform $\widehat{\mathbf{x}}_s$ is an estimate of the W-H transform of $\mathbf{s}$, and thus we can estimate $\mathbf{s}$ by the inverse W-H transform

$$\widetilde{\mathbf{s}} = \frac{1}{n} H^T \widehat{\mathbf{x}}_s$$

Define thresholds $\lambda_1, \cdots, \lambda_n \geqslant 0$. The ***hard thresholding*** is to modify $\widehat{\mathbf{x}}$ as follows:

$$\widehat{\mathbf{x}}_s = \begin{pmatrix} \widehat{x}_1 I(|\widehat{x}_1| \geqslant \lambda_1) \\ \vdots \\ \widehat{x}_n I(|\widehat{x}_n| \geqslant \lambda_n) \end{pmatrix}$$

The ***soft thresholding*** is to modify $\widehat{\mathbf{x}}$ as follows:

$$\widehat{\mathbf{x}}_s = \begin{pmatrix} \operatorname{sgn}(\widehat{x}_1)(|\widehat{x}_1| - \lambda_1)_+ \\ \vdots \\ \operatorname{sgn}(\widehat{x}_n)(|\widehat{x}_n| - \lambda_n)_+ \end{pmatrix}$$

where $\operatorname{sgn}(y)$ is the sign of y, and y_+ equals y if $y > 0$ and 0 if $y \leqslant 0$.

Typically we set $\lambda_1 = 0$, and use knowledge of the problem to decide $\lambda_2, \cdots, \lambda_n$; or take $\lambda_2 = \cdots = \lambda_n$ and choose the common value based on tools such as half normal plots.

2.5.3 The Fast W-H Transform

A Hadamard matrix H consists of ± 1 so computation of $H\mathbf{x}$ involves only additions and subtractions, but naive computation involves $n(n-1) = O(n^2)$ additions and subtractions, which is less than ideal if n is very large. We can write H as a product of sparse matrices to reduce complexity.

Example 2.2 ($n = 2^3 = 8$)**.** The 8×8 Hadamard matrix is

$$H_8 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

Naive computation of $H_8\mathbf{x}$ needs 56 additions and subtractions. But if $H_8 = A^3$ where

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix}$$

Computation of $AAA\mathbf{x}$ needs $3 \times 8 = 24$ additions and subtractions.

2.5.4 R code for FWHT

The function `fwht` below computes the W-H transform of data in a vector `x`.

```
fwht = function(x) {
  h=1
  len = length(x)
  while (h < len) {
    for (i in seq(1, len, by=h*2)) {
      for (j in seq(i, i+h-1)) {
        a = x[j]
        b = x[j+h]
        x[j] = a + b
        x[j+h] = a - b
      }
    }
  h = 2 * h
  }
  x
}
```

We can compute the inverse W-H transform using `fwht` by dividing the output by the length of the vector.

2.6 Fast Fourier Transform (FFT)

Definition 2.10 (Discrete Fourier Transform)**.** Suppose we have data $x_0, \cdots, x_{n-1}$, and define $\widehat{x}_0, \cdots, \widehat{x}_{n-1}$ by

$$\widehat{x}_j = \sum_{t=0}^{n-1} \exp\left(-2\pi\iota\frac{j}{n}t\right) x_t$$

where $\iota = \sqrt{-1}$.

Property 2.2 (Inverse DFT)**.** Given DFT, recover the original sequence by

$$x_t = \frac{1}{n}\sum_{j=0}^{n-1} \exp\left(2\pi\iota\frac{j}{n}t\right) \widehat{x}_j$$

Proof. For complex numbers z,

$$\sum_{j=0}^{n-1} z^j = \begin{cases} n, & z = 1 \\ \dfrac{1-z^n}{1-z}, & \text{otherwise} \end{cases}$$

Thus if $z = \exp\left(\dfrac{2\pi\iota t}{n}\right)$ for an integer t. we have

$$\sum_{j=0}^{n-1} z^j = \sum_{j=0}^{n-1} \exp\left(2\pi\iota\frac{t}{n}j\right) = \frac{1-\exp(2\pi\iota t)}{1-\exp(2\pi\iota t/n)} = 0$$

since $\exp(2\pi\iota t) = \cos(2\pi t) + \iota\sin(2\pi t) = 1$. Hence,

$$\begin{aligned}\frac{1}{n}\sum_{j=0}^{n-1}\exp\left(2\pi\iota\frac{j}{n}t\right)\widehat{x}_j &= \frac{1}{n}\sum_{j=0}^{n-1}\sum_{s=0}^{n-1}\exp\left(2\pi\iota\frac{t-s}{n}j\right)x_s \\ &= \frac{1}{n}\sum_{s=0}^{n-1}x_s\sum_{j=0}^{n-1}\exp\left(2\pi\iota\frac{t-s}{n}j\right) \\ &= x_t\end{aligned}$$

since

$$\sum_{j=0}^{n-1}\exp\left(2\pi\iota\frac{t-s}{n}j\right) = \begin{cases} n, & s = t \\ 0, & s \neq t \end{cases}$$

□

Definition 2.11 (Matrix Formulation of DFT)**.** Define $\mathbf{x} = (x_0, \cdots, x_{n-1})^T$ and $\widehat{\mathbf{x}} = (\widehat{x}_0, \cdots, \widehat{x}_{n-1})^T$. Then

$$\widehat{\mathbf{x}} = F\mathbf{x}$$

where F is an $n \times n$ matrix whose jth row and kth column is

$$f_{jk} = \exp\left(-2\pi\iota\frac{(j-1)(k-1)}{n}\right)$$

The elements of F^{-1} are

$$\overline{f}_{jk} = \frac{1}{n}\exp\left(2\pi\iota\frac{(j-1)(k-1)}{n}\right)$$

Note 1. Using the matrix form directly, we need $O(n^2)$ additions and multiplications to compute the DFT (and its inverse).

Note 2. We can write F as a product of sparse matrices, but unlike the W-H transform, factorization of the DFT matrix is more complicated.

2.6.1 FFT Derivation

Assume n is a product of prime numbers $n_1, \cdots, n_k : n = n_1 \times \cdots \times n_k$.

2.6.1.1 Case I: Even Number and Product of Small Prime Numbers

Assume n is even, then

$$\begin{aligned}\widehat{x}_j &= \sum_{t=0}^{n/2-1} \exp\left(-2\pi\iota\frac{j}{n}2t\right)x_{2t} + \sum_{t=0}^{n/2-1} \exp\left(-2\pi\iota\frac{j}{n}(2t+1)\right)x_{2t+1} \\ &= \underbrace{\sum_{t=0}^{n/2-1} \exp\left(-2\pi\iota\frac{j}{n/2}t\right)x_{2t}}_{\text{DFT of } x_0,x_2,\cdots} + \exp\left(-2\pi\iota\frac{j}{n}\right)\underbrace{\sum_{t=0}^{n/2-1} \exp\left(-2\pi\iota\frac{j}{n/2}t\right)x_{2t+1}}_{\text{DFT of } x_1,x_3,\cdots}\end{aligned}$$

Hence, the DFT of $x_0, \cdots, x_{n-1}$ is a linear combination of the DFT of the even and odd indices. Our rearrangement into DFT of odd and even indices can be written in matrix form as

$$\widehat{\mathbf{x}} = \underbrace{\begin{pmatrix} I & \Omega \\ I & -\Omega \end{pmatrix}}_{\text{Sparse}} \underbrace{\begin{pmatrix} F_{n/2} & 0 \\ 0 & F_{n/2} \end{pmatrix}}_{\text{Sparser}} P\mathbf{x}$$

where Ω is a diagonal matrix (sparse) and P is a permutation matrix (sparse), i.e., if n is even, we can write F as a product of two sparse matrices and a matrix that is sparser than F ($n^2/2$ 0s).

If $n/2$ is divisible by a prime number n', we can perform a similar decomposition of $F_{n/2}$ and F is now the product of sparser matrices. When $n_1, \cdots, n_k$ are small then we need $O(n\ln(n))$ additions and multiplications.

2.6.1.2 Case II: Prime Number with Zero-Padding

Definition 2.12 (Zero Padding)**.** Add 0s to the end of the sequence so that the length of the ***zero padded*** sequence is a product of small prime numbers:

$$x_0, \cdots, x_{n-1}, \underbrace{0, \cdots, 0}_{m}$$

with $n + m = n_1 \times \cdots \times n_k$ where $n_1, \cdots, n_k$ are small primes.

Note 1. The function `nextn` is useful for zero-padding.

Note 2. Adding 0s to a sequence changes the nature of the sequence - creating a large discontinuity, which is reflected in the DFT.

2.6.2 Analysis of DFT Approach

For the application in computation of probability distributions with DFT approach, we take $m \geqslant nl = 1$ where m is a product of small prime numbers, and follow the steps:

1. Define $\widehat{p}_i(0), \cdots, \widehat{p}_i(m-1)$ to be the DFT of $p_i(0), \cdots, p_i(m-1)$ for $i = 1, \cdots, n$.
2. Define
$$\widehat{p}_s(k) = \prod_{i=1}^{n} \widehat{p}_i(k), k = 0, \cdots, m-1$$
3. Inverse DFT: $P(S=0), \cdots, P(S=m-1)$ is the inverse DFT of $\widehat{p}_s(0), \cdots, \widehat{p}_s(m-1)$.

The number of multiplications at each step is:

1. DFT: $n \times O(m\ln(m)) = O(nm\ln(m))$.
2. Product of DFTs: $O(nm)$.

3. Inverse DFT: $O(m \ln(m))$.

The total number of multiplications is $O(nm \ln(m))$ and thus if $m \approx nl$, the number of multiplications is $O(n^2 l \ln(nl))$ versus $O(n^3 l^2)$ for the brute force algorithm.

3 Generation of Random Variates

3.1 Generation of Random Numbers

Example 3.1 (Importance Sampling)**.** Suppose we want to estimate

$$I = \int \cdots \int g(\mathbf{x}) \mathrm{d}\mathbf{x}$$

for some integrand $g : \mathbb{R}^p \to \mathbb{R}$. If f is a probability density function on $\mathbb{R}^p$, then

$$I = \int \cdots \int g(\mathbf{x}) \mathrm{d}\mathbf{x} = \int \cdots \int \frac{g(\mathbf{x})}{f(\mathbf{x})} f(\mathbf{x}) \mathrm{d}\mathbf{x} = \mathbb{E}_f \left[\frac{g(\mathbf{X})}{f(\mathbf{X})} \right]$$

where $\mathbf{X}$ has a density f. We can use the law of large numbers to estimate the expected value provided $\mathrm{Var}_f \left[\frac{g(\mathbf{X})}{f(\mathbf{X})} \right] < \infty$. Take $\mathbf{X}_1, \cdots, \mathbf{X}_n$ independent from f, LLN gives

$$\widehat{I} = \frac{1}{n} \sum_{i=1}^{n} \frac{g(\mathbf{X}_i)}{f(\mathbf{X}_i)} \approx \int \cdots \int g(\mathbf{x}) \mathrm{d}\mathbf{x}$$

Note. We choose f satisfying precision and expediency:

1. Precision: Minimize the variance of $\widehat{I}$.
2. Expediency: Be able to sample from f.

Example 3.2 (Monte Carlo Estimation of π)**.** If X and Y are independent Unif$(-1, 1)$ r.v.s., then

$$P(X^2 + Y^2 \leqslant 1) = \frac{\pi}{4}$$

We generate independent pairs and have

$$\widehat{\pi} = \frac{4}{n} \sum_{i=1}^{n} I(X_i^2 + Y_i^2 \leqslant 1)$$

3.2 Generation of Unif$(0, 1)$

To generate pseudo-random $U_1, U_2, \cdots$, we generate integers $V_1, V_2, \cdots$ from a uniform distribution on $\{1, \cdots, N\}$ and define $U_i = \dfrac{V_i}{N+1}$ for $i = 1, 2, \cdots$. Note that $U_1, U_2, \cdots$ are uniform on the set $\{1/(N+1), \cdots, N/(N+1)\}$. If N is large enough, $U_1, U_2, \cdots$ are independent Unif$(0, 1)$ r.v.s.:

$$\sup_{0 \leqslant x \leqslant 1} |P(U_i \leqslant x) - x| \leqslant \frac{1}{N}$$

3.2.1 Linear Congruential RNG

Define $V_1, V_2, \cdots$ via the recursion:

$$V_{k+1} = (aV_k + b) \mod m$$

for some integers a, b, and m.

Note 1. The initial value V_0 is the ***seed*** of the RNG.

Note 2. $V_1, V_2, \cdots$ take values in the set $\{0, \cdots, m-1\}$.

Note 3. If $b = 0$ then we have a ***multiplicative congruential*** RNG.

Note 4. We have $V_{k+p} = V_k$ for some $p \leqslant m$, and p is the ***period*** of the RNG.

Property 3.1. If $b = 0$, then the maximum possible period is $m-1$. Furthermore, if m is prime, and

$$a^{(m-1)/q} \mod m \neq 1$$

for every prime factor q of $m-1$ then the RNG has period $m-1$.

Example 3.3. Take $m = 5$ and $m-1 = 4$ has a single prime factor 2. We need $a^2 \mod 5 \neq 1$ so we can take $a = 3$ (for example).

Example 3.4. Let m to be the largest possible prime number $m = 2^{31} - 1$. We can take $a = 16807$ or 48271, or 397204094.

3.2.2 Combining Unif$(0, 1)$ RNGs

Combination increases the period of the RNG.

Example 3.5 (Wichmann-Hill RNG)**.** Combine three multiplicative congruential RNGs:

$$\begin{aligned} V_{k+1}^{(1)} &= 171 V_k^{(1)} \mod 30269 \\ V_{k+1}^{(2)} &= 172 V_k^{(2)} \mod 30307 \\ V_{k+1}^{(3)} &= 170 V_k^{(3)} \mod 30323 \end{aligned}$$

where the periods are short ($\approx 3 \times 10^4$). Then

$$U_k = \left(\frac{V_k^{(1)}}{30269} + \frac{V_k^{(2)}}{30307} + \frac{V_k^{(3)}}{30323} \right) \mod 1$$

where the period is

$$p = \frac{30268 \times 30306 \times 30322}{4} = 6.9536 \times 10^{12}$$

3.2.3 Shift Register Method

We use the binary representation of Unif$(0, 1)$. Suppose $Z_1, Z_2, \cdots$ are independent binary r.v.s. with

$$P(Z_k = 0) = P(Z_k = 1) = \frac{1}{2}$$

then

$$U = \sum_{k=1}^{\infty} \frac{Z_k}{2^k} \sim \text{Unif}(0, 1)$$

In practice, we define U as a finite sum

$$U = \sum_{k=1}^{r} \frac{Z_k}{2^k}$$

where r is the number of bits.

We generate $\{Z_k\}$ via ***exclusive-or*** operations for binary variables x and y. We construct $\{Z_k\}$ as follows:

$$Z_k = Z_{k-p} \oplus Z_{k-p+q}, 1 < q < p$$

and

$$U_n = \sum_{k=1}^{r} \frac{Z_{n-s(k)}}{2^k}$$

for some shifts $\{s(k)\}$.

Recall. If Z_1 and Z_2 are independent, and $Z_3 = Z_1 \oplus Z_2$, then Z_3 is independent of Z_1 and Z_2.

Note 1. For the shifts, we need $s(k) - s(k-1) \gg p$.

Note 2. Initialization of shift register RNGs is much complicated since Z_k is a function of Z_{k-p} and Z_{k-p+q} and U_n depends on r values of $\{Z_k\}$.

Note 3. We need a $p \times r$ matrix of binary seeds.

Example 3.6 (Lewis-Payne RNG)**.** $p = 98, q = 27$, and $s(k) = 100p(k-1)$ s.t. $s(k)-s(k-1) = 100p$. The period is $2^{98} - 1$.

Example 3.7 (Mersenne Twister)**.** The period is $2^{19937} - 1$.

3.3 Testing Unif$(0, 1)$ RNGs

We need to check:

1. Uniformity on $[0, 1]$: For $0 \leqslant a < b \leqslant 1$,

$$\frac{1}{n}\sum_{i=1}^{n} I(a \leqslant U_i \leqslant b) \approx b - a$$

2. Uniformity of k-tuples on $[0, 1]^k$: For $A \subset [0, 1]^k$,

$$\binom{n}{k}^{-1} \sum_{(i_1,\cdots,i_k)} I[(U_{i_1}, \cdots, U_{i_k}) \in A] \approx \text{Volume}(A)$$

3. Independence: U_i independent of $U_{i+1}, U_{i+2}, \cdots$.

3.4 RNGs in R

The function `RNGkind` that allows a user to specify the RNG used to generate Unif$(0, 1)$ r.v.s. and the method used to generate normal r.v.s..

3.5 Methods for Continuous Distribution

3.5.1 Inverse Method

Suppose F is a univariate distribution and we want to generate $X \sim F$.

Definition 3.1. For a general univariate distribution function F, we define

$$F^{-1}(t) = \inf\{x : F(x) \geqslant t\}, 0 < t < 1$$

Property 3.2. If F is a univariate distribution function with inverse F^{-1} and $U \sim$ Unif$(0, 1)$, then

$$X = F^{-1}(U) \sim F$$

Proof. We need to show $P(F^{-1}(U) \leqslant x) = F(x)$ or equivalently $[F^{-1}(U) \leqslant x] = [U \leqslant F(x)]$. By definition of F^{-1}, $[U \leqslant F(x)]$ implies $[F^{-1}(U) \leqslant x]$. If $F^{-1}(U) \leqslant x$ then $F(x + \varepsilon) \geqslant U, \forall \varepsilon > 0$. F is right continuous so $[F^{-1}(U) \leqslant x]$ implies $[U \leqslant f(x)]$. □

Example 3.8 (Exponential Distribution)**.** $F(x) = 1 - \exp(-\lambda x)$ for $x \geqslant 0, \lambda > 0$. Solving $F(F^{-1}(t)) = t$ for $F^{-1}(t)$, we have

$$F^{-1}(t) = -\frac{\ln(1-t)}{\lambda}$$

Thus $X = -\dfrac{\ln(1-U)}{\lambda}$ has an exponential distribution. Since $1 - U \sim \text{Unif}(0, 1)$ so we define $X = -\dfrac{-\ln(U)}{\lambda}$.

Example 3.9 (Logistic Distribution)**.** $F(x) = \dfrac{\exp(x)}{1 + \exp(x)}$. Solving $F(F^{-1}(t)) = t$, we have

$$F^{-1}(t) = \ln\left(\frac{t}{1-t}\right)$$

which is called logit function. Thus $X = \ln\left(\dfrac{U}{1-U}\right)$ has a Logistic distribution.

Example 3.10 (Approximation of Euler's Constant)**.** The Euler's constant is

$$\begin{aligned}
\gamma &= \lim_{m\to\infty}\left[\sum_{k=1}^{m}\frac{1}{k} - \ln(m)\right] \\
&= \int_1^\infty \left(\frac{1}{\lfloor x \rfloor} - \frac{1}{x}\right) \mathrm{d}x \\
&= \int_1^\infty x^2 \left(\frac{1}{\lfloor x \rfloor} - \frac{1}{x}\right) x^{-2} \mathrm{d}x
\end{aligned}$$

where $f(x) = x^{-2}$ is a density function on $[1, \infty)$. If we can sample $X_1, \cdots, X_n$ from $f(x)$, we can estimate γ by

$$\widehat{\gamma} = \frac{1}{n}\sum_{i=1}^{n} X_i^2 \left(\frac{1}{\lfloor X_i \rfloor} - \frac{1}{X_i}\right)$$

The distribution function is $F(x) = 1 - x^{-1}$ whose inverse is $F^{-1}(t) = (1-t)^{-1}$. We can use inverse method to sample from $f(x)$.

```
n = 1000000
u = runif(n)
x = 1 / (1 - u)
gammahat = mean(x^2 * (1 / floor(x) - 1 / x))
```

3.5.2 Rejection Sampling

Assume F is continuous with density function f and $F^{-1}(t)$ is not easily computable. Suppose we want to sample X from a density f. We define a proposal density g s.t. $f(x) \leqslant Mg(x)$ for all x and some $M < \infty$. We sample Y from g and $U \sim \text{Unif}(0, 1)$ where Y and U are independent, and define $T = \dfrac{f(Y)}{Mg(Y)}$. If $U \leqslant T$, then set $X = Y$; if $U > T$, then reject and repeat until acceptance. The algorithm works: Given independent $Y \sim g$ and $U \sim \text{Unif}(0, 1)$,

$$\begin{aligned}
P(X \leqslant x) &= P\left(Y \leqslant x \middle| U \leqslant \frac{f(Y)}{Mg(Y)}\right) \\
&= \frac{P(Y \leqslant x, U \leqslant f(Y)M^{-1}g^{-1}(Y))}{P(U \leqslant f(Y)M^{-1}g^{-1}(Y))}
\end{aligned}$$

Since $Y \perp U$, the joint density of (Y, U) is

$$h(y,u) = \begin{cases} g(y), & 0 \leqslant u \leqslant 1 \\ 0, & \text{otherwise} \end{cases}$$

Therefore,

$$\begin{aligned} P(\text{Accept})P\left(U \leqslant \frac{f(Y)}{Mg(Y)}\right) &= \int_{-\infty}^{\infty} \int_{0}^{f(y)M^{-1}g^{-1}(y)} g(y)\mathrm{d}u\mathrm{d}y \\ &= \frac{1}{M}\int_{-\infty}^{\infty} f(y)\mathrm{d}y \\ &= \frac{1}{M} \end{aligned}$$

and

$$\begin{aligned} P\left(Y \leqslant x, U \leqslant \frac{f(Y)}{Mg(Y)}\right) &= \int_{-\infty}^{x} \int_{0}^{f(y)M^{-1}g^{-1}(y)} g(y)\mathrm{d}u\mathrm{d}y \\ &= \frac{1}{M}\int_{-\infty}^{x} f(y)\mathrm{d}y \\ &= \frac{P(X \leqslant x)}{M} \end{aligned}$$

Note 1. The probability of acceptance of a given proposal is $\frac{1}{M}$.
Note 2. If f and g are close then M will be close to 1.
Note 3. We can evaluate M by maximizing $\frac{f(x)}{g(x)}$ but we do not need to find the smallest possible M with $f(x) \leqslant Mg(x)$ since rejection sampling will work with a sub-optimal M with a lower probability of acceptance.
Note 4. f and g can be joint density functions or probability mass functions.

Example 3.11 (Half-Normal Distribution with Exponential Proposal)**.** Suppose we want to sample X from a half-normal distribution whose density is

$$f(x) = \frac{2}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right), x \geqslant 0$$

i.e., if $X \sim \mathcal{N}(0,1)$, then $|X| \sim f$. Since X takes values on $[0, \infty)$, a natural proposal distribution is exponential

$$g(y) = \exp(-y), y \geqslant 0$$

since the tails of the exponential are heavier than those of the normal distribution so M should be finite.

To fine M, we need to maximize $\frac{f(x)}{g(x)}$ over $x \geqslant 0$, i.e.,

$$\max \ln[f(x)] - \ln[g(x)]$$

After calculation, we find $\frac{f(x)}{g(x)}$ is maximized at $x = 1$ and

$$M = \frac{f(1)}{g(1)} = 1.315489$$

and the probability of acceptance of a given proposal is

$$\frac{1}{M} = 0.76$$

The code to generate half-normal r.v.s. is:

```
x = NULL
count = 0
total = 0 # Number of proposals generated
while (count < 100) {
  reject = T
  while (reject) {
    y = rexp(1)
    u = runif(1)
    total = total + 1
    if (u <= 2*dnorm(y)/(1.315489*dexp(y))) {
      x = c(x, y)
      count = count + 1
      reject = F
    }
  }
}
```

Example 3.12 (Cauchy Distribution)**.** Suppose we want to sample X from a Cauchy distribution whose density is

$$f(x) = \frac{1}{\pi(1+x^2)}$$

The distribution function is

$$F(x) = \frac{1}{2} + \frac{1}{\pi}\tan^{-1}(x)$$

and

$$F^{-1}(t) = \tan\left[\pi\left(t - \frac{1}{2}\right)\right], 0 < t < 1$$

We can generate r.v.s. from a Cauchy distribution using the inverse method bug floating point evaluation of $\tan(x)$ is not always straightforward.

We can write $f(x)$ as a mixture of two densities

$$f(x) = \frac{1}{2}f_1(x) + \frac{1}{2}f_2(x)$$

where

$$f_1(x) = \frac{2}{\pi(1+x^2)}, |x| \leqslant 1$$
$$f_2(x) = \frac{2}{\pi(1+x^2)}, |x| > 1$$

We know that if $X \sim f_1$, then $X^{-1} \sim f_2$. Therefore, we generate Z from f_1 and $U \sim \text{Unif}(0,1)$ with $Z \perp U$. If $U > \frac{1}{2}, X = Z$; if $U < \frac{1}{2}, X = \frac{1}{Z}$. Hence, we can use rejection sampling to sample from

f_1. Taking g to be a uniform distribution on $[-1.1]$ is a reasonable choice, and $\dfrac{f_1(x)}{g(x)}$ is maximized at $x = 0$. Thus

$$M = \frac{f_1(0)}{g(0)} = \frac{4}{\pi} = 1.273$$

and

$$P(\text{Accept}) = \frac{\pi}{4} = 0.785$$

3.6 Sampling from Mixture Densities

Suppose we want to sample from a density $f(x)$ which can be written as a mixture of k components:

$$f(x) = \lambda_1 f_1(x) + \cdots + \lambda_k f_k(x)$$

where $f_1(x), \cdots, f_k(x)$ are densities, $\lambda_1 + \cdots + \lambda_k = 1$. We sample a discrete r.v. J from a discrete distribution with $P(J = j) = \lambda_j$ (and we can do with a single Unif$(0,1)$ r.v.). Given $J = j$, sample X from $f_j(x)$. The algorithm works best if k is small or if $\lambda_1 = \cdots = \lambda_k = \dfrac{1}{k}$: $J = \lceil kU \rceil$.

3.6.1 Application: Walker's Alias Method

Suppose we want to sample X from a discrete distribution

$$f(x_j) = P(X = x_j) = p_j, j = 1, \cdots, k$$

where $p_1 + \cdots + p_k = 1$. We can write f as a mixture of k components each with weight k^{-1} :

$$f(x) = P(X = x) = \frac{1}{k} f_1(x) + \cdots + \frac{1}{k} f_k(x), x = x_1, \cdots, x_k$$

$f_1, \cdots, f_k$ are discrete distribution putting mass at two points:

$$f_j(x) = \begin{cases} \tau_j, & x = x_j \\ 1 - \tau_j, & x = a_j \end{cases}$$

where $a_j \in \{x_1, \cdots, x_k\}$ is called an alias.

Given $\tau_1, \cdots, \tau_k$ and $a_1, \cdots, a_k$, we sample X from f as follows: Generate $U_1 \sim$ Unif$(0,1)$ and set $J = \lceil kU_1 \rceil$; generate $U_2 \sim$ Unif$(0,1)$ and define $X = x_J$ if $U_2 \leqslant \tau_J$ and $X = a_J$ if $U_2 > t_J$.

Note 1. We require a separate algorithm to construct $\tau_1, \cdots, \tau_k$ and $a_1, \cdots, a_k$.

Note 2. Walker's alias method is used by the R function `sample` when the option `replace=T` is given.

Example 3.13 (Binomial Distribution)**.** Take $X \sim$ Binom$(3, 0.4)$:

$$f(x) = P(X = x) = \binom{3}{x} 0.4^x 0.6^{3-x}, x = 0, 1, 2, 3$$

where $f(0) = 0.216, f(1) = 0.432, f(2) = 0.288, f(3) = 0.064$. We need to write

$$f(x) = \frac{1}{4} \sum_{i=0}^{3} f_i(x)$$

with $f_i(x) = \tau_i$ is $x = i$ and $f_i(x) = 1 - \tau_i$ if $x = a_i$ where $a_i \in \{0, 1, 2, 3\}$. We let

$$\begin{aligned} \tau_0 &= 0.272 \quad a_0 = 1 \\ \tau_1 &= 1 \quad \text{No alias} \\ \tau_2 &= 0.408 \quad a_2 = 0 \\ \tau_3 &= 0.256 \quad a_3 = 2 \end{aligned}$$

3.7 Generation of Normal Random Variables

3.7.1 Inverse Method

Define the $\mathcal{N}(0, 1)$ distribution function

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left(-\frac{t^2}{2}\right) \mathrm{d}t$$

$\Phi(x)$ is strictly increasing so we can define its inverse by $\Phi(\Phi^{-1}(t)) = t$ for $0 \leqslant t \leqslant 1$. Thus given $U \sim \text{Unif}(0, 1), X = \Phi^{-1}(U) \sim \mathcal{N}(0, 1)$.

Note 1. It is the default method in R.

Note 2. Though $\Phi^{-1}(t)$ is not a nice function, it is very well approximated.

3.7.2 Box-Muller Method

If X_1 and X_2 are independent $\mathcal{N}(0, 1)$, then their joint density is

$$f(x_1, x_2) = \frac{1}{2\pi} \exp\left(-\frac{x_1^2 + x_2^2}{2}\right)$$

Convert to polar coordinates: $X_1 = R\cos(\Theta)$ and $X_2 = R\sin(\Theta)$, and (R, Θ) has joint density

$$g(r, \theta) = \frac{1}{2\pi} \exp\left(-\frac{r^2}{2}\right) r, r > 0, 0 \leqslant \theta < 2\pi$$

where $g(r, \theta) = g_1(r)g_2(\theta)$ and so $R \perp \Theta$. $\Theta \sim \text{Unif}(0, 2\pi)$ and $R = \sqrt{V}$ where V is exponential with mean 2. We generate R from $g_1(r)$ and Θ from $g_2(\theta)$, and

$$X_1 = R\cos(\Theta), X_2 = R\sin(\Theta)$$

If U_1 and U_2 are independent Unif$(0, 1)$, then we can define

$$\Theta = 2\pi U_1, R = \sqrt{-2\ln(U_2)}$$

3.7.3 Kinderman-Ramage Method

Consider half-normal distribution

$$f(x) = \frac{2}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$$

and we write $f(x)$ as a mixture of two distributions

$$\begin{aligned} f(x) &= \lambda_1 f_1(x) + \lambda_2 f_2(x) \\ &= 0.884 \times \underbrace{\text{Triangular density}}_{0.90 - 0.41x} + 0.116 f_2(x) \end{aligned}$$

Note 1. It is easy to generate from the triangular density $f_1(x) : U_1$ and U_2 are independent Unif$(-1, 1)$, then $V = \dfrac{2.216|U_1 + U_2|}{2}$ has density $f_1(x)$.

Note 2. It is not easy to generate from $f_2(x)$.

3.7.4 Monty Python Method

We generate independent r.v.s. U_1 and U_2 s.t. (U_1, U_2) have a uniform distribution on

$$\mathcal{B} = [0, \sqrt{2\pi}] \times \left[0, \frac{1}{\sqrt{2\pi}}\right]$$

and divide $\mathcal{B}$ into 4 regions - depending on which region (U_1, U_2), we can define a r.v. X with a half-normal distribution.

Define

$$f(x) = \frac{2}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$$

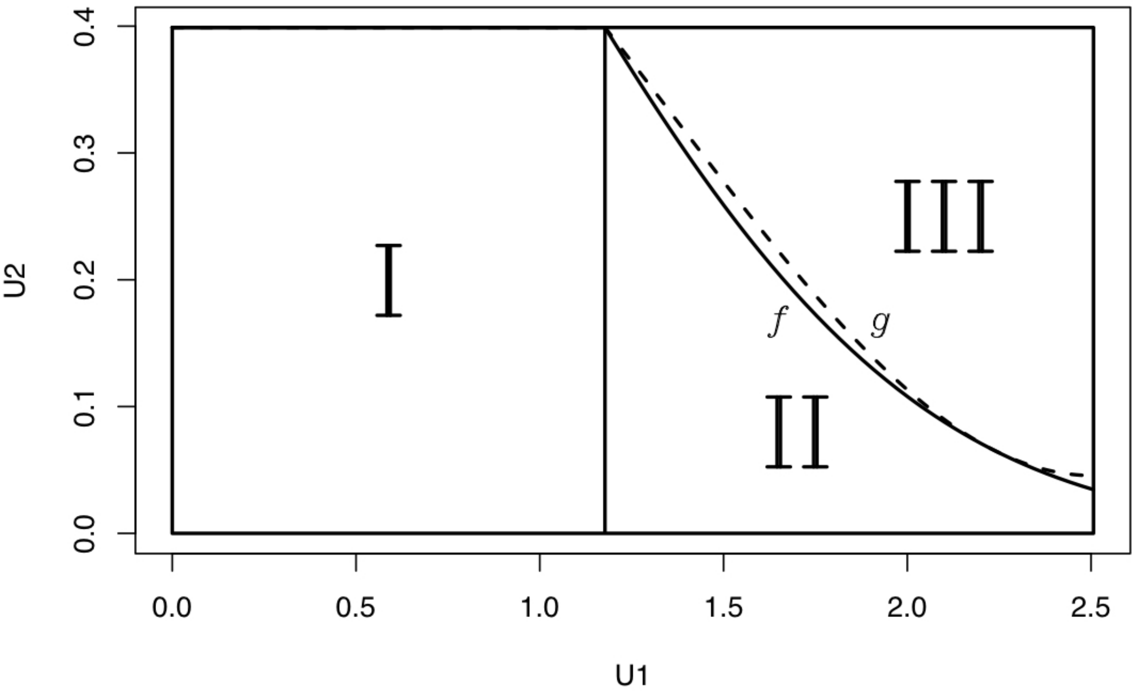
Define $g(x)$ to be $f(x)$ rotated and rescaled into $\mathcal{B}$ for x s.t.

$$f(x) > \frac{1}{\sqrt{2\pi}} = \frac{1}{b} \text{ or } x < \sqrt{\ln(4)} = a$$

i.e.,

$$g(x) = \frac{1}{b} - \frac{a}{b-a}\left[f\left(\frac{a(b-x)}{b-a}\right) - \frac{1}{b}\right]$$

for $\sqrt{\ln(4)} = a \leqslant x \leqslant b = \sqrt{2\pi}$. We can refine regions I, II, and III in terms of $f(x)$ and $g(x)$.



We generate (U_1, U_2) on $\mathcal{B} : U_1 \sim \text{Unif}(0, \sqrt{2\pi})$ and $U_2 \sim \text{Unif}(0, 1/\sqrt{2\pi})$. If $(U_1, U_2) \in \text{I}$, then $X = U_1$; if $(U_1, U_2) \in \text{II}$, then $X = U_1$; if $(U_1, U_2) \in \text{III}$, then $X = \frac{a(b-U_1)}{b-a}$; otherwise, we need to generate X from the tail $(x > \sqrt{2\pi})$ of the half-normal distribution (by rejection sampling with a shifted exponential proposal).

3.7.5 Sum of Uniforms

We sum k independent Unif$(0,1)$ r.v.s. $U_1, \cdots, U_k$ and define

$$X = \frac{U_1 + \cdots + U_k - k/2}{\sqrt{k/12}}$$

where the normalization guarantees $\mathbb{E}[X] = 0$ and $\text{Var}[X] = 1$.

Note. $k = 12$ works well.

3.8 Markov Chain Monte Carlo

3.8.1 Construction of Reversible Markov Chain

We first assume that $f(\mathbf{x})q(\mathbf{x},\mathbf{y}) > f(\mathbf{y})q(\mathbf{y},\mathbf{x})$. Define $q^*(\mathbf{x},\mathbf{y}) = \alpha(\mathbf{x},\mathbf{y})q(\mathbf{x},\mathbf{y})$ s.t.

$$f(\mathbf{x})q^*(\mathbf{x},\mathbf{y}) = f(\mathbf{y})q^*(\mathbf{y},\mathbf{x})$$

The solution is

$$\begin{aligned}\alpha(\mathbf{x},\mathbf{y}) &= \frac{f(\mathbf{y})q(\mathbf{y},\mathbf{x})}{f(\mathbf{x})q(\mathbf{x},\mathbf{y})}\\ \alpha(\mathbf{y},\mathbf{x}) &= 1\end{aligned}$$

We can do the similar thing if $f(\mathbf{x})q(\mathbf{x},\mathbf{y}) < f(\mathbf{y})q(\mathbf{y},\mathbf{x})$. In general,

$$\alpha(\mathbf{x},\mathbf{y}) = \min\left\{\frac{f(\mathbf{y})q(\mathbf{y},\mathbf{x})}{f(\mathbf{x})q(\mathbf{x},\mathbf{y})}, 1\right\}$$

Note. $q^*(\mathbf{x},\mathbf{y})$ may not be a transition density (unless $\alpha(\mathbf{x},\mathbf{y}) = 1$ for all $\mathbf{x},\mathbf{y}$). Given $\mathbf{X}_{i-1} = \mathbf{x}$, we can fix by allowing $\mathbf{X}_i = \mathbf{x}$ w.p.

$$\alpha(\mathbf{x},\mathbf{x}) = 1 - \int\cdots\int q^*(\mathbf{x},\mathbf{y})\mathrm{d}\mathbf{y}$$

which ideally should be small.

3.8.2 Metropolis-Hastings Algorithm

Suppose we want to generate $\mathbf{X}_i$ from $f(\mathbf{x})$ and we have a proposal transition density $q(\mathbf{x},\mathbf{y})$. Given $\mathbf{X}_{i-1} = \mathbf{x}$, we generate $\mathbf{Y}$ from $q(\mathbf{x},\mathbf{y})$ (density in $\mathbf{y}$ for each $\mathbf{x}$) and $U \sim$ Unif$(0,1)$ independent of $\mathbf{Y}$. If $U \leqslant \alpha(\mathbf{X}_{i-1},\mathbf{Y})$, then $\mathbf{X}_i = \mathbf{Y}$; if $U > \alpha(\mathbf{X}_{i-1},\mathbf{Y})$, then $\mathbf{X}_i = \mathbf{X}_{i-1}$.

We want to sample $\mathbf{X}_i$ s.t.

$$\frac{1}{n}\sum_{i=1}^n h(\mathbf{X}_i) \text{ converges to } \int\cdots\int h(\mathbf{x})f(\mathbf{x})\mathrm{d}\mathbf{x}$$

as fast as possible for any function h, and the convergence speed is determined largely by

$$\text{Var}\left[\frac{1}{n}\sum_{i=1}^n h(\mathbf{X}_i)\right] \approx \frac{1}{n}\left\{\text{Var}[h(\mathbf{X}_i)] + 2\sum_{s=1}^{\infty}\text{Cov}[h(\mathbf{X}_i), h(\mathbf{X}_{i+s})]\right\}$$

The choice of $q(\mathbf{x},\mathbf{y})$ is important – it determines $\alpha(\mathbf{x},\mathbf{y})$, i.e., how often $\mathbf{X}_{i-1} = \mathbf{X}_i$, and how quickly $\mathbf{X}_i$ move around the space – we want to make the autocovariance terms small.

Example 3.14. Suppose we want to generate X_i from

$$P(X_i = x) = \binom{2}{x} 0.3^2 0.7^{2-x}, x = 0, 1, 2$$

Using simple transition matrix

$$Q = \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{pmatrix}$$

s.t. $q(x, y) = \dfrac{1}{3}$ for $x, y = 0, 1, 2$. Our acceptance probability is

$$\alpha(x, y) = \min\left\{\frac{f(y)q(y, x)}{f(x)q(x, y)}, 1\right\} = \min\left\{\frac{\binom{2}{y}}{\binom{2}{x}} 0.3^{y-x} 0.7^{x-y}, 1\right\}$$

3.8.2.1 Application to Bayesian Inference

$\alpha(\mathbf{x}, \mathbf{y})$ depends on f only via the ratio $\dfrac{f(\mathbf{y})}{f(\mathbf{x})}$ and we only need to know $f(\mathbf{x})$ up to a multiplicative constant. Hence, we do not need to know the constant to sample from the posterior density.

3.8.2.2 Random Walk (Metropolis) Sampler

$q(\mathbf{x}, \mathbf{y}) = g(\mathbf{y} - \mathbf{x})$ for some density g :

$$\alpha(\mathbf{x}, \mathbf{y}) = \min\left\{\frac{f(\mathbf{y})g(\mathbf{x} - \mathbf{y})}{f(\mathbf{x})g(\mathbf{y} - \mathbf{x})}, 1\right\}$$

Given $\mathbf{X}_{i-1} = \mathbf{x}$, we would generate the proposal $\mathbf{Y}$ by

$$\mathbf{Y} = \mathbf{x} + \mathbf{Z}$$

where the density of $\mathbf{Z}$ is g.

Note. If g is symmetric around 0, then

$$\alpha(\mathbf{x}, \mathbf{y}) = \min\left\{\frac{f(\mathbf{y})}{f(\mathbf{x})}, 1\right\}$$

3.8.2.3 Independence (Hastings) Sampler

$q(\mathbf{x}, \mathbf{y}) = g(\mathbf{y})$: For each i, the distribution of the proposal $\mathbf{Y}$ is independent of $\mathbf{X}_{i-1}$

$$\alpha(\mathbf{x}, \mathbf{y}) = \min\left\{\frac{f(\mathbf{y})g(\mathbf{x})}{f(\mathbf{x})g(\mathbf{y})}, 1\right\}$$

Note. The independence sampler somewhat resembles rejection sampling (but without the rejection).

Example 3.15 (Poisson Distribution)**.** Suppose we want to sample X_i from a Poisson distribution with mean $\lambda > 0$. We use independence sampler with a geometric proposal

$$g(x) = (1 - \theta)\theta^x, x = 0, 1, \cdots$$

whose expected value is $\dfrac{\theta}{1-\theta}$. We choose θ s.t. $\dfrac{\theta}{1-\theta} = \lambda$, i.e., $\theta = \dfrac{\lambda}{1+\lambda}$. We can sample from a geometric distribution by $\lfloor V \rfloor$ where V as an exponential distribution with mean $-\dfrac{1}{\ln(\theta)}$.

The code for $\lambda = 5$ is:

```
lambda = 5
x = 5
samp = NULL
theta = lambda / (1 + lambda)
for (i in 1:10000) {
  v = -rexp(1) / log(theta)
  y = floor(v)
  u = runif(1)
  if (u <= dpois(y, lambda) * theta^(x-y) / dpois(x, lambda)) x = y
  samp = c(samp, x)
}
eprob = NULL
for (i in 0: 10) eprob = c(eprob, sum(samp==i) / 10000)
```

3.8.3 Practical Issues of MCMC

MCMC is often very sensitive to initial conditions. We can discard the first m iterations of the MCMC algorithm.

Note. This is important when sampling high dimensional random vectors.

It is useful to treat the output of an MCMC algorithm as a time series, and we can look at time series plots to see when the output has achieved stationarity, and autocorrelation $\widehat{\rho}(1), \cdots$.

The effective sample size is

$$n_{\text{eff}} = \left[1 + 2\sum_{s=1}^{\infty} \rho(s)\right]^{-1} n$$

4 Numerical Linear Algebra

4.1 Solving Linear Equations

Theorem 4.1 (Sherman-Morrison-Woodbury Formula/Woodbury Matrix Identity)**.**

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$$

Note. If A and C are diagonal matrices, then computation of $(A + UCV)^{-1}$ is easy.

Example 4.1. Let $\mathbf{u}$ and $\mathbf{v}$ be vectors of length n and so $\mathbf{u}\mathbf{v}^T$ has rank 1. We can now use the Woodbury matrix identity setting $A = I, C = 1, U = \mathbf{u}, V = \mathbf{v}^T$

$$\begin{aligned}(I + \mathbf{u}\mathbf{v}^T)^{-1} &= I - I\mathbf{u}(1 + \mathbf{v}^T I\mathbf{u})^{-1}\mathbf{v}^T I \\ &= I - \frac{1}{1 + \mathbf{v}^T\mathbf{u}}\mathbf{u}\mathbf{v}^T\end{aligned}$$

and thus

$$A^{-1}\mathbf{b} = \mathbf{b} - \frac{\mathbf{v}^T\mathbf{b}}{1 + \mathbf{v}^T\mathbf{u}}\mathbf{u}$$

Example 4.2. Define

$$A = \begin{pmatrix} 1 & 1-\varepsilon \\ 1+\varepsilon & 1 \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & -\varepsilon \\ \varepsilon & 0 \end{pmatrix}}_{B_\varepsilon} + \underbrace{\begin{pmatrix} 1 \\ 1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}^T}_{\mathbf{v}\mathbf{v}^T}$$

where $B_\varepsilon^{-1} = B_{1\varepsilon}$ (both off-diagonal matrices). We can apply the Woodbury identity to evaluate $A^{-1}\mathbf{b}$:

$$\begin{aligned}A^{-1}\mathbf{b} &= B_\varepsilon^{-1}\mathbf{b} - \frac{1}{1 + \mathbf{v}^T B_\varepsilon^{-1}\mathbf{v}} B_\varepsilon^{-1}\mathbf{v}\mathbf{v}^T B_\varepsilon^{-1}\mathbf{b} \\ &= B_\varepsilon^{-1}\mathbf{b} - \frac{\mathbf{v}^T B_\varepsilon^{-1}\mathbf{b}}{1 + \mathbf{v}^T B_\varepsilon^{-1}\mathbf{v}} B_\varepsilon^{-1}\mathbf{v}\end{aligned}$$

The R code is:

```
eps = 1.e-7
B = matrix(c(0, -eps, eps, 0), ncol=2, byrow=T)
b = c(1, 1)
v = c(1, 1)
solve(B, b) - sum(v*solve(B, b)) * solve(B, v) / (1 + sum(v*solve(B, v)))
```

Definition 4.1 (Pivoting)**.** Pivoting means exchanging rows. Let P be a permutation matrix where each row and column has exactly one 1 and $n - 1$ 0s, then $P\mathbf{b}$ rearranges elements of $\mathbf{b}$ while PA rearranges rows of A.

Gaussian elimination with partial pivoting is essentially the approach used by R function `solve`. For some permutation matrix P, we find lower and upper triangular matrices L and U s.t.

$$PA = LU$$

Then

$$\begin{aligned}PA\mathbf{x} &= P\mathbf{b} \\ L\underbrace{U\mathbf{x}}_{\mathbf{y}} &= P\mathbf{b}\end{aligned}$$

We solve $L\mathbf{y} = P\mathbf{b}$ for $\mathbf{y}$ and then solve $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$.

4.2 Matrix Factorizations

4.2.1 Cholesky Factorization

If A is a symmetric ($A = A^T$) and positive definite ($\mathbf{x}^T A\mathbf{x} > 0$ for $\mathbf{x} \neq \mathbf{0}$) $n \times n$ matrix, then we can write

$$A = LL^T$$

where L is lower triangular.

4.2.1.1 Computation of L

Define A_k to be the upper left $k \times k$ sub-matrix of A where $A_1 = a_{1,1}$ and $A_n = A$. A_k is symmetric positive definite so $A_k = L_k L_k^T$ where L_k is lower triangular and L_k is a sub-matrix of L_{k+1}.

Define $\mathbf{v}_{k-1} = (l_{k,1}, \cdots, l_{k,k-1})^T$ and $\mathbf{a}_{k-1} = (a_{1,k}, \cdots, a_{k-1,k})^T$, then

$$\begin{aligned} A_k &= \begin{pmatrix} A_{k-1} & \mathbf{a}_{k-1} \\ \mathbf{a}_{k-1}^T & a_{k,k} \end{pmatrix} \\ &= \begin{pmatrix} A_{k-1} & L_{k-1}\mathbf{v}_{k-1} \\ \mathbf{v}_{k-1}^T L_{k-1}^T & l_{k,1}^2 + \cdots + l_{k,k}^2 \end{pmatrix} \end{aligned}$$

Thus we have

$$L_{k-1}\mathbf{v}_{k-1} = \mathbf{a}_{k-1} \text{ (Lower triangular system)}$$

and

$$l_{k,k} = \sqrt{a_{k,k} - (l_{k,1}^2 + \cdots + l_{k,k-1}^2)}$$

Then we can successively compute $L_1 = \sqrt{a_{1,1}}, L_2, \cdots, L_n = L$:

$$L_k = \begin{pmatrix} & L_{k-1} & & 0 \\ l_{k,1} & \cdots & l_{k,k-1} & l_{k,k} \end{pmatrix}$$

Example 4.3. Let

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

Then

$$L_1 = l_{1,1} = \sqrt{a_{1,1}} = \sqrt{2}$$

and

$$L_2 = L = \begin{pmatrix} \sqrt{2} & 0 \\ l_{2,1} & l_{2,2} \end{pmatrix}$$

We have

$$\sqrt{2} l_{2,1} = a_{1,2} = 1 \Rightarrow l_{2,1} = \frac{1}{\sqrt{2}}$$

and

$$l_{2,2} = \sqrt{a_{2,2} - l_{2,1}^2} = \sqrt{\frac{3}{2}}$$

Thus

$$L = \begin{pmatrix} \sqrt{2} & 0 \\ \sqrt{1/2} & \sqrt{3/2} \end{pmatrix}$$

4.2.1.2 Application: Generating Multivariate Normal Random Vectors

Suppose we want to generate a random vector $\mathbf{X}$ from a p-variate normal distribution with mean vector $\mathbf{0}$ and $p \times p$ covariance matrix C where we assume C is positive definite.

We generate $Y_1, \cdots Y_p$ independent $\mathcal{N}(0,1)$ r.v.s. and define $\mathbf{Y} = (Y_1, \cdots, Y_p)^T$, and compute L in the Cholesky factorization of C. We define $\mathbf{X} = L\mathbf{Y}$ and $\mathbf{X} \sim \mathcal{N}_p(\mathbf{0}, LL^T = C)$.

Note. In R, we use function `chol` that returns L^T.

4.3 Iterative Matrix Method

Suppose we solve $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$ where A is an $n \times n$ matrix with n very large. If A is not too complicated then we can solve iteratively, i.e., find a sequence $\{\mathbf{x}_k\}$ s.t. $\mathbf{x}_k$ converges to the solution. We write $A = A_1 + A_2$ where A_1 is nice (e.g., diagonal, lower or upper triangular) and define $\mathbf{x}_1, \mathbf{x}_2, \cdots$ s.t.

$$A_1\mathbf{x}_{k+1} = \mathbf{b} - A_2\mathbf{x}_k \text{ or } \mathbf{x}_{k+1} = A_1^{-1}\mathbf{b} - A_1^{-1}A_2\mathbf{x}_k$$

4.3.1 Jacobi and Gauss-Seidel Algorithm

Suppose we want to solve

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

For each $i = 1, \cdots, n$, we have

$$b_i = \sum_{j=1}^{n} a_{ij}x_j = a_{ii}x_i + \sum_{j \neq i} a_{ij}x_j$$

so that

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij}x_j\right)$$

Then,

1. Jacobi algorithm:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij}x_j^{(k)}\right)$$

2. Gauss-Seidel algorithm:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}x_j^{(k+1)} - \sum_{j>i} a_{ij}x_j^{(k)}\right)$$

Note. The Gauss-Seidel algorithm computes $\mathbf{x}_{k+1}$ in place, which is more efficient for memory; while for the Jacobi algorithm, we need to store both $\mathbf{x}_{k+1}$ and $\mathbf{x}_k$.

Let $A = L + D + U$:

$$A = \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ a_{21} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{n,n-1} & 0 \end{pmatrix} + \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix} + \begin{pmatrix} 0 & a_{12} & \cdots & a_{1,n-1} & a_{nn} \\ \vdots & \cdots & \ddots & \cdots & \vdots \\ 0 & 0 & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}$$

We can write both the Gauss-Seidel and Jacobi iterations in terms of L, D and U :

1. Jacobi: $\mathbf{x}_{k+1} = D^{-1}[\mathbf{b} - (L+U)\mathbf{x}_k]$.
2. Gauss-Seidel: $(L+D)\mathbf{x}_{k+1} = \mathbf{b} - U\mathbf{x}_k$ or $\mathbf{x}_{k+1} = (L+D)^{-1}(\mathbf{b} - U\mathbf{x}_k)$.

4.3.1.1 Convergence of Gauss-Seidel and Jacobi Algorithm

Suppose $\mathbf{x}_k \to \mathbf{x}^*$ as $k \to \infty$, then

1. Jacobi: $\mathbf{x}^* = D^{-1}[\mathbf{b} - (L+U)\mathbf{x}^*]$ or $(L+D+U)\mathbf{x}^* = \mathbf{b}$.
2. Gauss-Seidel: $\mathbf{x}^* = (L+D)^{-1}(\mathbf{b} - U\mathbf{x}^*)$ or $(L+D+U)\mathbf{x}^* = \mathbf{b}$.

4.3.1.2 Comment

1. When n is large and A is relatively sparse, then iterative method can be much more efficient than direct method.
2. Iterative method is easy to program.
3. We can extend the basic idea behind the Gauss-Seidel algorithm to other problems, such as back-fitting, coordinate descent, etc.
4. We can improve the algorithm with successive over-relaxation (S0OR) method.

4.3.1.3 Application: Quadratic Minimization

Suppose we want to minimize the quadratic function

$$g(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b} + c$$

where A is symmetric positive definite, i.e., $A^T = A$ and $\mathbf{x}^T A\mathbf{x} > 0$ for all $\mathbf{x} \neq \mathbf{0}$.

We can write out $g(\mathbf{x})$ explicitly:

$$g(\mathbf{x}) = g(x_1, \cdots, x_n) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_i x_j - \sum_{i=1}^{n} x_i b_i + c$$

Now fix $\{x_i : i \neq k\}$ and minimize $g(\mathbf{x})$ w.r.t. x_k :

$$\frac{\partial}{\partial x_k} g(\mathbf{x}) = \sum_{i=1}^{n} a_{ik}x_i - b_k = a_{kk}x_k + \sum_{i\neq k} a_{ik}x_i - b_k$$

Setting the partial derivative to 0, we have

$$x_k = \frac{1}{a_{kk}}\left(b_k - \sum_{i\neq k} a_{ik}x_i\right)$$

which is simply a Gauss-Seidel iteration.

4.4 Solving Least Squares Problems

4.4.1 Additive Regression Models

Definition 4.2 (Additive Regression Model). Given data $\{(x_{i1}, \cdots, x_{ip}, y) : i = 1, \cdots, n\}$, we assume that

$$y_i = \beta_0 + f_1(x_{i1}) + \cdots + f_p(x_{ip}) + \varepsilon_i, i = 1, \cdots, n$$

where $\{x_{ij} : i = 1, \cdots, n; j = 1, \cdots, p\}$ are predictor variables, $f_1, \cdots, f_p$ are unknown smooth functions, and $\{\varepsilon_i\}$ are r.v.s. with mean 0 and finite variance.

Note 1. For identifiability reasons, we assume that

$$\sum_{i=1}^{n} f_j(x_{ij}) = 0, j = 1, \cdots, p$$

Note 2. Suppose we have a single predictor, we observe $(x_1, y_1), \cdots, (x_n, y_n)$ and the model is $y_i = g(x_i) + \varepsilon_i, i = 1, \cdots, n$ where g is smooth function. We estimate $g(x)$ by a weighted average of $\{y_i : |x_i - x| \leqslant h\}$ where h is a tuning parameter (bandwidth) that controls the smoothness of the estimate $\widehat{g}(x)$ (***non-parametric estimation***).

Note 3. Vector from:

$$\mathbf{y} = \beta_0 \mathbf{1} + \mathbf{f}_1 + \cdots + \mathbf{f}_p + \varepsilon$$

Definition 4.3 (Smoothing Matrix)**.** Define vectors

$$\widehat{\mathbf{g}} = \begin{pmatrix} \widehat{g}(x_i) \\ \vdots \\ \widehat{g}(x_n) \end{pmatrix} \text{ and } \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

We have

$$\widehat{\mathbf{g}} = S\mathbf{y}$$

where the $n \times n$ matrix S is a smoothing matrix.

Note 1. For simplicity, we assume S is symmetric.

Note 2. S typically depends on some tuning parameters (e.g., a bandwidth parameter).

Note 3. The eigenvalues $\lambda_1, \cdots, \lambda_n$ of a smoothing matrix S satisfy:

1. $-1 < \lambda_1, \cdots, \lambda_n \leqslant 1$.
2. At least one eigenvalue is 1.
3. Special case: Projection matrix (eigenvalues 0 and 1).

Note 4. The space spanned by eigenvectors of S with eigenvalue 1 typically includes simple functions (e..g, linear functions and possibly low order polynomials) $S(a\mathbf{1} + b\mathbf{x}) = a\mathbf{1} + b\mathbf{x}$.

Definition 4.4. For a given smoothing matrix S, we define the equivalent degrees of freedom or equivalent number of parameters of S as

$$\text{eqdf}(S) = \text{trace}(S) = \text{Measure of model complexity}$$

Example 4.4 (Penalized Least Squares)**.** Define $\widehat{\mathbf{g}}$ to minimize

$$\underbrace{\sum_{i=1}^{n} (y_i - g(x_i))^2}_{\|\mathbf{y}-\mathbf{g}\|^2 = (\mathbf{y}-\mathbf{g})^T(\mathbf{y}-\mathbf{g})} + \lambda \mathbf{g}^T A \mathbf{g}$$

where A is non-negative definite. Then

$$\widehat{\mathbf{g}} = \underbrace{(I + \lambda A)^{-1}}_{S} \mathbf{y}$$

4.4.2 The Backfitting Algorithm

Define smoothing matrices $S_0, \cdots, S_k$ where

$$S_0 = \begin{pmatrix} 1/n & \cdots & 1/n \\ \vdots & \ddots & \vdots \\ 1/n & \cdots & 1/n \end{pmatrix}$$

and $S_k\mathbf{1} = \mathbf{0}$ for $k = 1, \cdots, p$. Given current estimates $\widehat{\beta}_0, \widehat{\mathbf{f}}_1, \cdots, \widehat{\mathbf{f}}_p$, we update $\widehat{\mathbf{f}}_k$ as

$$\widehat{\mathbf{f}}_k \leftarrow S_k\left(\mathbf{y} - \widehat{\beta}_0\mathbf{1} - \sum_{j\neq k}\widehat{\mathbf{f}}_j\right)$$

Then $\widehat{\beta}_0$ is updated as

$$\widehat{\beta}_0\mathbf{1} \leftarrow S_0\left(\mathbf{y} - \sum_{j=1}^{p}\widehat{\mathbf{f}}_j\right)$$

Suppose the backfitting estimate converges, then at convergence, we would have

$$\widehat{\mathbf{f}}_k = S_k\left(\mathbf{y} - \widehat{\beta}_0\mathbf{1} - \sum_{j\neq k}\widehat{\mathbf{f}}_j\right), k = 1, \cdots, p$$

and

$$\widehat{\beta}_0\mathbf{1} = S_0\left(\mathbf{y} - \sum_{j=1}^{p}\widehat{\mathbf{f}}_j\right)$$

We can write in matrix form:

$$\begin{pmatrix} I & S_1 & S_1 & \cdots & S_1 & S_1 \\ S_2 & I & S_2 & \cdots & S_2 & S_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ S_p & S_p & S_p & \cdots & I & S_p \\ S_0 & S_0 & S_0 & \cdots & S_0 & I \end{pmatrix}\begin{pmatrix} \widehat{\mathbf{f}}_1 \\ \widehat{\mathbf{f}}_2 \\ \vdots \\ \widehat{\mathbf{f}}_p \\ \widehat{\beta}_0\mathbf{1} \end{pmatrix} = \begin{pmatrix} S_1\mathbf{y} \\ S_2\mathbf{y} \\ \vdots \\ S_p\mathbf{y} \\ S_0\mathbf{y} \end{pmatrix}$$

4.4.3 Application: Measure of Dependence

Suppose X and Y are r.v.s. with some joint distribution. The measure of linear dependence or association is

$$\rho = \frac{\text{Cov}(X, Y)}{\sqrt{\text{Var}[X]\text{Var}[Y]}}$$

Given bivariate data $\{(x_i, y_i) : i = 1, \cdots, n\}$ from distribution and we can estimate ρ by

$$\widehat{\rho} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}}$$

Recall. $\rho = 0$ does not imply $X \perp Y$, and two r.v.s. can be highly dependent and have $\rho = 0$.

Definition 4.5 (Maximal Correlation)**.** Functions ψ and ϕ to maximize the correlation between $\psi(X)$ and $\phi(Y)$.

Note. The computation for ψ and ϕ are difficult. Assume X and Y have joint PDF $f(x, y)$ with respective marginal density functions $f_X(x)$ and $f_Y(y)$. To compute the maximal correlation, we need to maximize $\mathbb{E}[\psi(X)\phi(Y)]$ over all functions ψ and ϕ satisfying $\mathbb{E}[\psi(X)] = \mathbb{E}[\phi(Y)] = 0$ and $\mathbb{E}[\psi^2(X)] = \mathbb{E}[\phi^2(Y)] = 1$, then ψ and ϕ must satisfy the conditions

$$\mathbb{E}[\psi(X)|Y = y] = \int_{-\infty}^{\infty} \psi(x)\frac{f(x,y)}{f_Y(y)}\mathrm{d}x = \lambda\phi(y)$$

and

$$\mathbb{E}[\phi(Y)|X = x] = \int_{-\infty}^{\infty} \phi(y)\frac{f(x,y)}{f_X(x)}\mathrm{d}y = \lambda\psi(x)$$

where $\lambda \in [0, 1]$ is the maximal correlation.

Example 4.5. Suppose $X \sim \text{Unif}(-1, 1)$ and $Y = X^2$, then $\text{Cov}(X, Y) = \mathbb{E}[XY] = \mathbb{E}[X^3] = 0$. If we take $\psi(X) = X^2$ and $\phi(Y) = Y = X^2$, then the correlation between $\psi(X)$ and $\phi(Y)$ is 1.

4.4.3.1 Computing λ : The Alternating Conditional Expectation (ACE) Algorithm

Given data $(x_1, y_1), \cdots, (x_n, y_n)$ we can estimate ψ and ϕ by iterative smoothing:

1. Estimate $\psi(x)$ by smoothing $\{\widehat{\phi}(y_i)\}$ as a function of $\{x_i\}$.
2. Estimate $\phi(y)$ by smoothing $\{\widehat{\psi}(x_i)\}$ as a function of $\{y_i\}$.
3. Iterate the process until convergence.

The R function for the ACE algorithm is:

```
ace = function(x, y, niter=5, span=0.75) {
  x1 = scale(x)
  y1 = scale(y)
  for (i in 1:niter) {
    r = loess(y1~x, span=span)
    x1 = scale(r$fitted)
    r = loess(x1~y, span=span)
    y1= scale(r$fitted)
    }
  corr =cor(x1, y1)
  r = list(x=x, y=y, xhat=x1, yhat=y1, cor=corr)
  }
```

Note 1. `loess` is the locally weighted quadratic smoother.

Note 2. `span=0.75` parameter in `loess` means we use approximately 75% of the data.

Note 3. The estimates of ψ and ϕ are contained in the components `$xhat` and `$yhat`, respectively. The maximal correlation is in `$cor`.

4.4.4 Least Squares

The general least squares problem is to minimize

$$\|\mathbf{y} - X\beta\|^2$$

where X is an $n \times r$ matrix with $n > r$ and $\| \cdot \|$ is the L_2 norm. The problem is called and overdetermined system of equations.

4.4.4.1 Linear Regression Model

Suppose the linear regression model is

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^T \beta + \varepsilon_i, i = 1, \cdots, n.$$

We can write the model in matrix form

$$\mathbf{y} = X\beta + \varepsilon$$

where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, X = \begin{pmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix} \text{ and } \varepsilon = \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

The least squares problem is to find $\widehat{\beta}$ to minimize

$$\sum_{i=1}^{n} (y_i - \mathbf{x}_i^T \beta)^2 = \|\mathbf{y} - X\beta\|^2.$$

If $\{\varepsilon_i\}$ are independent $\mathcal{N}(0, \sigma^2)$ then $\widehat{\beta}$ is the MLE of β. If we differentiable the objective function w.r.t. β and set the partial derivatives to 0, we get the ***normal equations*** for $\widehat{\beta}$

$$(X^T X)\widehat{\beta} = X^T \mathbf{y}$$

If $X^T X$ is invertible then

$$\widehat{\beta} = (X^T X)^{-1} X^T \mathbf{y}$$

where $X^T X$ is invertible if $\text{rank}(X) = p + 1$.

Note 1. One simple algorithm (Cholesky): If $X^T X$ is positive definite and $\text{rank}(X) = p+1$, then Cholesky factorization gives $X^X = LL^T$ where L is lower triangular, and we can solve $LL^T \widehat{\beta} = X^T \mathbf{y}$.

Note 2. The Cholesky algorithm relies on being able to compute $X^T X$ and $X^T \mathbf{y}$ with minimal round-off error. Computation of $X^T X$ and $X^T \mathbf{y}$ is moderately expensive: $O(np^2)$ floating point operations for $X^T X$ and $O(np)$ floating point operations for $X^T \mathbf{y}$. If the columns of X are collinear, the condition number of $X^T X$ is large, then round-off error in computation of $X^T X$ is magnified and the Cholesky factorization does not work.

4.4.4.2 Least Square Estimation with the QR Decomposition

Suppose the columns of X are orthogonal vectors $\mathbf{q}_0, \cdots, \mathbf{q}_p$:

$$\mathbf{q}_j^T \mathbf{q}_k = 0, j \neq k$$

then $X^T X = D$, a diagonal matrix with diagonal elements $\mathbf{q}_0^T \mathbf{q}_0, \cdots, \mathbf{q}_p^T \mathbf{q}_p$ and

$$\widehat{\beta} = D^{-1} X^T \mathbf{y}$$

We write $X = QR$ where the columns of Q are orthonormal vectors (norm is 1) and R is upper triangular. We want to minimize

$$\|\mathbf{y} - X\beta\|^2 = \|\mathbf{y} - QR\beta\|^2$$

Define $\alpha = R\beta$ and minimize

$$\|\mathbf{y} - Q\alpha\|^2$$

w.r.t. α. The normal equations for $\widehat{\alpha}$ are

$$\widehat{\alpha} = (Q^T Q)^{-1} Q^T \mathbf{y} = Q^T \mathbf{y}$$

since Q has orthonormal columns. We can then compute $\widehat{\beta}$ by solving the upper triangular system

$$R\widehat{\beta} = \widehat{\alpha}$$

Write

$$X = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix} = (\mathbf{v}_0 \quad \cdots \quad \mathbf{v}_p)$$

We can use the Gram-Schmidt algorithm to orthogonalize the vectors $\mathbf{v}_0, \cdots, \mathbf{v}_p$ to get Q. Note that $\mathbf{v}_0$ depends only on the first column of $Q, \mathbf{q}_0$ and for $j = 1, \cdots, p, \mathbf{v}_j$ depends on $\mathbf{q}_0, \cdots, \mathbf{q}_j$. Thus $X = QR$ where R is upper triangular.

Note 1. The Gram-Schmidt algorithm is numerically unstable: Due to round-off error, $\mathbf{q}_1, \cdots, \mathbf{q}_r$ are not exactly orthogonal, i.e., $Q^Q \neq I$. The lack of orthogonality is worse if the condition number of $X^T X$ is large (collinear columns).

Note 2. Alternative algorithms for computing Q and R that are more numerically stable:

1. Modified Gram-Schmidt (adjusts the vectors to improve orthogonality).
2. Householder reflections (best method for achieving orthogonality).

Example 4.6 (Leverage Score)**.** In regression, the fitted values $\hat{\mathbf{y}} = X\widehat{\beta}$ can be written as

$$\hat{\mathbf{y}} = X(X^T X)^{-1} X^T \mathbf{y} = H\mathbf{y}$$

where matrix H is the hat matrix. The diagonal element h_{ii} of H is the leverage score that measure the potential influence of observation i. Using the QR decomposition $X = QR$, we have

$$\begin{aligned} H &= QR(R^T Q^T QR)^{-1} R^T Q^T \\ &= QR(R^T R)^{-1} R^T Q^T \text{ (Since } Q^T Q = I) \\ &= QRR^{-1}(R^T)^{-1} R^T Q^T \\ &= QQ^T \end{aligned}$$

If $\mathbf{v}_1, \cdots, \mathbf{v}_n^T$ are the rows of Q then $h_{ii} = \mathbf{v}_i^T \mathbf{v}_i$.

4.5 Randomized Numerical Linear Algebra

If we want to infer the properties of a matrix A using random sampling, we can sample random vectors $\mathbf{V}_1, \cdots, \mathbf{V}_m$ from some distribution, evaluate $\mathbf{X}_1 = A\mathbf{V}_1, \cdots, \mathbf{X}_m = A\mathbf{V}_m$, and use the empirical distribution of $\mathbf{X}_1, \cdots, \mathbf{X}_m$ to infer properties of A. The random vectors $\mathbf{V}_1, \cdots, \mathbf{V}_m$ are probing vectors.

4.5.1 Hutchinson's Method

Theorem 4.2. Suppose that $\mathbf{V}$ is a random vector with $\mathbb{E}[\mathbf{V}\mathbf{V}^T] = I$, then

$$\mathbb{E}[\mathbf{V}^T A \mathbf{V}] = \text{trace}(A)$$

Proof. Define $Z = \mathbf{V}\mathbf{V}^T$ whose elements are r.v.s. $\{Z_{ij}\}$ with $\mathbb{E}[Z_{ij}] = 1$ if $i = j$, and 0 otherwise. Then

$$\begin{aligned}\mathbb{E}[\mathbf{V}^T A\mathbf{V}] = \mathbb{E}[\text{trace}(AZ)] = \mathbb{E}\left[\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}Z_{ji}\right] \\ = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}\mathbb{E}[Z_{ji}] \\ = \sum_{i=1}^{n} a_{ii} = \text{trace}(A)\end{aligned}$$

□

Note 1. We can now use the law of large numbers to approximate $\mathbb{E}[\mathbf{V}^T A\mathbf{V}] = \text{trace}(A)$. If $\mathbf{V}_1, \cdots, \mathbf{V}_m$ are independent random vectors with $\mathbb{E}[\mathbf{V}_i\mathbf{V}_i^T] = I$, then

$$\widehat{\text{trace}}(A) = \frac{1}{m}\sum_{i=1}^{m}\mathbf{V}_i^T A\mathbf{V}_i \approx \text{trace}(A)$$

for large m.

Note 2. The variance of $\widehat{\text{trace}}(A)$ is

$$\text{Var}[\widehat{\text{trace}}(A)] = \frac{1}{n}\text{Var}[\mathbf{V}^T A\mathbf{V}]$$

and we can find the distribution of $\mathbf{V}$ to minimize $\text{Var}[\mathbf{V}^T A\mathbf{V}]$ over distributions where $\mathbb{E}[\mathbf{V}\mathbf{V}^T] = I$.

Theorem 4.3. For any $A, \text{Var}[\mathbf{V}^T A\mathbf{V}]$ is minimized for $\mathbf{V} = (V_1, \cdots, V_n)^T$ where $V_1, \cdots, V_n$ are independent with

$$P(V_i = 1) = P(V_i = -1) = \frac{1}{2}$$

which is the Rademacher distribution.

4.5.1.1 Application: Equivalent Degrees of Freedom for Loess

Suppose the model

$$\mathbf{y} = \mathbf{g} + \varepsilon$$

We estimate $\mathbf{g}$ by $\widehat{\mathbf{g}} = S\mathbf{y}$ for some smoothing matrix S that is not explicitly defined. Recall that $\text{eqdf}(S) = \text{trace}(S)$. Using Hutchinson's method, we have

$$\widehat{\text{eqdf}}(S) = \frac{1}{M}\sum_{i=1}^{m}\mathbf{V}_i^T S\mathbf{V}_i$$

The code computing Hutchinson estimate in R is:

```
tracedf = function(x, span=0.7, m=100) {
  traces = NULL
  n = length(x)
  for (i in 1:m) {
    v = ifelse(runif(n)>0.5, 1, -1)
    r = loess(v~x, span=span)
    traces = c(traces, sum(v*r$fitted))
    }
enp = mean(traces)
std.err = sd(traces) / sqrt(m)
r = list(enp=enp, std.err=std.err)
return(r)
}
```

4.5.1.2 Function of Symmetric Matrix and Trace

Suppose that A is symmetric ($A^T = A$), then

$$A = \Gamma\Lambda\Gamma^T$$

where Γ is a diagonal matrix whose elements are the eigenvalues $\lambda_1, \cdots, \lambda_n$ of A and Λ is an orthogonal matrix whose columns are the eigenvectors of A.

If $f(x)$ is a function with $f(\lambda_1), \cdots, f(\lambda_n)$ well-defined, then we can define

$$f(A) = \Gamma f(\Lambda)\Gamma^T$$

where $f(\Lambda)$ is a diagonal matrix matrix with diagonal elements $f(\lambda_1), \cdots, f(\lambda_n)$.

Note. $\text{trace}[f(A)] = \sum_{i=1}^{n} f(\lambda_i)$.

If f is a polynomial

$$f(A) = \alpha_0 I + \alpha_1 A + \cdots + \alpha_p A^p$$

then approximating trace$[f(A)]$ using Hutchinson's method is straightforward

$$\widehat{\text{trace}}[f(A)] = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=0}^{p} \mathbf{V}_i^T A^k \mathbf{V}_i$$

where $A^k\mathbf{V}_i = A(A^{k-1}\mathbf{V}_i)$.

If f is not a polynomial, then we may be able to approximate it by a polynomial, such as Taylor series, i.e., for some A_0 we have

$$f(A) = f(A_0) + f'(A_0)(A - A_0) + \frac{1}{2}f''(A_0)(A - A_0)^2 + \cdots$$

4.5.1.3 Estimating $\det(A)$

Suppose that $A = I - B$ where the eigenvalues of B lie in $(-1, 1)$, then eigenvalues of A lie in $(0, 2)$. We have

$$\begin{aligned}
\ln(\det(A)) &= \sum_{i=1}^{n} \ln(1 - \lambda_i) \\
&= -\sum_{i=1}^{n}\sum_{k=1}^{\infty} \frac{\lambda_i^k}{k} \\
&= -\sum_{k=1}^{\infty}\left(\frac{1}{k}\sum_{i=1}^{n}\lambda_i^k\right) \\
&= -\sum_{k=1}^{\infty} \frac{\text{trace}(B^k)}{k}
\end{aligned}$$

To apply Hutchinson's method, we need to find r s.t.

$$\sum_{k=r+1}^{\infty} \frac{\operatorname{trace}(B^k)}{k} \approx 0$$

Then we can estimate $\det(A)$ by

$$\widehat{\det}(A) = \exp\left(- \sum_{k=1}^{r} \sum_{i=1}^{m} \frac{\mathbf{V}_i^T B^k \mathbf{V}_i}{km} \right)$$

for probing vectors $\mathbf{V}_1, \cdots, \mathbf{V}_m$ with $\mathbb{E}[\mathbf{V}_i \mathbf{V}_i^T] = I$.

We can extend the method to a general symmetric positive definite matrix A :

1. Write $A = D(I - B)$ where D is diagonal and eigenvalues of B lie in $(-1, 1)$. We have $B = I - D^{-1}A$.

2. We have $\det(A) = \det(D)\det(I - B)$. Note that we need to know the eigenvalues of A to define D.

5 Optimization

We want to minimize or maximize some objective function $g(\mathbf{x})$ for $\mathbf{x} \in \mathcal{C}$. If $\mathcal{C}$ is an open set, i.e., every point in $\mathcal{C}$ is an interior point, we use unconstrained optimization. If $\mathcal{C}$ is not an open set, i.e., some points in $\mathcal{C}$ lie on the boundary of $\mathcal{C}$, we use constrained optimization.

5.1 Unconstrained Optimization

Consider minimizing $g(\mathbf{x})$ over an open set $\mathcal{O}$, or equivalently maximizing $-g(\mathbf{x})$ over $\mathcal{O}$. If g is differentiable over $\mathcal{O}$, then $\mathbf{x}^*$ minimizing g must satisfy

$$\nabla g(\mathbf{x}^*) = \begin{pmatrix} \frac{\partial}{\partial x_1} g(\mathbf{x}^*) \\ \vdots \\ \frac{\partial}{\partial x_p} g(\mathbf{x}^*) \end{pmatrix} = \mathbf{0}$$

Note. We may have multiple solutions and we check the second partial derivatives to determine if we have a local maximum or local minimum.

5.2 Convex Function

Convex minimization is ubiquitous in statistics and machine learning.

Definition 5.1 (Convex)**.** A function $g(\mathbf{x})$ is convex if

$$g(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) \leqslant \lambda g(\mathbf{x}) + (1-\lambda)g(\mathbf{y})$$

for any $\lambda \in [0,1]$. If the inequality holds strictly for $\lambda \in (0,1)$, then g is strictly convex.

Note 1. If $p = 1$ and $g''(x) \geqslant 0$ for all x, then g is convex. For $p > 1$, if the $p \times p$ matrix of second partial derivatives is non-negative definite for all $\mathbf{x}$, then g is convex.

Note 2. Convex functions are not necessarily differentiable at all $\mathbf{x}$. E.g., $g(x) = |x|$.

Note 3. If $g(\mathbf{x})$ is a strictly convex and differentiable function, then $\mathbf{x}^*$ satisfying $\nabla g(\mathbf{x}^*) = \mathbf{0}$ is the unique minimizing value of g, which follows from the fact that for strictly convex function,

$$g(\mathbf{x}_0) + [\nabla g(\mathbf{x}_0)]^T(\mathbf{x} - \mathbf{x}_0) < g(\mathbf{x})$$

for all $\mathbf{x} \neq \mathbf{x}_0$.

Property 5.1. If $g_1(\mathbf{x})$ and $g_2(\mathbf{x})$ are convex, then $g(\mathbf{x}) = g_1(\mathbf{x}) + g_2(\mathbf{x})$ is also convex.

Property 5.2. If g_0 is convex, then $g(\mathbf{x}) = g_0(A\mathbf{x} + \mathbf{b})$ is also convex for any matrix A and $\mathbf{b}$.

5.2.1 Sub-Gradient

Definition 5.2 (Sub-Gradient)**.** Suppose $g(\mathbf{x})$ is a convex function, the sub-gradient $\partial g(\mathbf{x})$ is

$$\partial g(\mathbf{x}) = \{\mathbf{v} : g(\mathbf{y}) \geqslant g(\mathbf{x}) + \mathbf{v}^T(\mathbf{y} - \mathbf{x}) \text{ for all } \mathbf{y}\}$$

Note 1. In general, $\partial g(\mathbf{x})$ is set-valued (a closed convex set).

Note 2. If g is defined on the real line, then

$$\partial g(x) = \{v : g(y) \geqslant g(x) + v(y - x) \text{ for all } y\}$$

i.e., $\phi_x(y) = g(x) + v(y - x)$ is a line with slope v s.t. $\phi_x(x) = g(x)$ and $\phi_x(y) \leqslant g(y)$ for all y.

Property 5.3. If g is differentiable at $\mathbf{x}$, then $\partial g(\mathbf{x}) = \nabla g(\mathbf{x})$. Note that $\partial g(\mathbf{x})$ is a single point in $\mathbb{R}^p$.

Property 5.4. If $g = g_1 + g_2$, then

$$\partial g(\mathbf{x}) = \{\mathbf{v}_1 + \mathbf{v}_2 : \mathbf{v}_1 \in \partial g_1(\mathbf{x}), \mathbf{v}_2 \in \partial g_2(\mathbf{x})\}$$

Property 5.5. If $g(\mathbf{x}) = g_0(A\mathbf{x} + \mathbf{b})$ where g_0 is convex, then

$$\partial g(\mathbf{x}) = A^T \partial g_0(A\mathbf{x} + \mathbf{b}) = \{A^T\mathbf{v} : \mathbf{v} \in \partial g_0(A\mathbf{x} + \mathbf{b})\}$$

Example 5.1. $g(x) = x^2$. We have $g'(1) = \partial g(1) = 2$.

Example 5.2. $g(x) = x^2 + 2|x - 1|$. We have $\partial g(1) = [0, 4]$.

Example 5.3. $g(x) = |x|$. We have

$$\partial g(x) = \begin{cases} -1, & x < 0 \\ 1, & x > 0 \\ [-1, 1], & x = 0 \end{cases}$$

Example 5.4. $g(x) = g_0(x) + |x|$ where g_0 is differentiable. We have

$$\partial g(x) = \begin{cases} g_0'(x) - 1, & x < 0 \\ g_0'(x) + 1, & x > 0 \\ [g_0'(0) - 1, g_0'(0) + 1], & x = 0 \end{cases}$$

Example 5.5. $g(x) = \max\{0, 1 - x\}$ is not differentiable at $x = 1$ with $\partial g(1) = [-1, 0]$.

Example 5.6. $g(x, y) = \max\{x, y\}$. We have

$$\partial g(x, y) = \begin{cases} (1 \quad 0)^T, & x > y \\ (0 \quad 1)^T, & x < y \\ \{(\tau \quad 1 - \tau)^T : 0 \leqslant \tau \leqslant 1\}, & x = y \end{cases}$$

5.2.2 Sub-Gradient and Convex Optimization

If $g(\mathbf{x})$ is a convex function with sub-gradient $\partial g(\mathbf{x})$, then $\mathbf{x}^*$ minimizes g iff

$$\mathbf{0} \in \partial g(\mathbf{x}^*)$$

If g is strictly convex, then $\mathbf{x}^*$ satisfying $\mathbf{0} \in \partial g(\mathbf{x}^*)$ is unique.

5.2.3 Application: L_1 Regression

Suppose we want to minimize

$$g(\beta) = \sum_{i=1}^n |y_i - \mathbf{x}_i^T \beta|$$

w.r.t. β. We evaluate the sub-gradient of each summand of g:

$$\partial |y_i - \mathbf{x}_i^T \beta| = \begin{cases} -\operatorname{sgn}(y_i - \mathbf{x}_i^T \beta)\mathbf{x}_i, & y_i - \mathbf{x}_i^T \beta \neq 0 \\ \{\tau \mathbf{x}_i : -1 \leqslant \tau \leqslant 1\}, & y_i - \mathbf{x}_i^T \beta = 0 \end{cases}$$

The L_1 estimate $\widehat{\beta}$ satisfies

$$\sum_{y_i \neq \mathbf{x}_i^T \beta} \operatorname{sgn}(y_i - \mathbf{x}_i^T \widehat{\beta})\mathbf{x}_i = \sum_{y_i = \mathbf{x}_i^T \beta} \tau_i \mathbf{x}_i$$

where $-1 \leqslant \tau_i \leqslant 1$.

5.3 Constrained Optimization

We want to find $\mathbf{x}^*$ to minimize $g(\mathbf{x})$ for $\mathbf{x} \in \mathcal{C}$ subject to some conditions. One general approach to solve constrained optimization problems is to make the problem an unconstrained minimization problem.

Two basic approaches:

1. Introduce Lagrange multipliers and slack variables: Minimize

$$g(\mathbf{x}) + \sum_{j=1}^{k} \lambda_j (g_j(\mathbf{x}) - z_j^2)$$

w.r.t. $\mathbf{x}$, Lagrange multipliers $\lambda_1, \cdots, \lambda_k$, and slack variables $z_1, \cdots, z_k$.

2. Approximate the constrained problem by a sequence of unconstrained problems: Find $\mathbf{x}_\lambda^*$ to minimize

$$g(\mathbf{x}) + \text{Penalty}_\lambda(\mathbf{x})$$

so that $\mathbf{x}_\lambda^* \to \mathbf{x}^*$ as $\lambda \to 0$.

Example 5.7 (L_1 Estimation in Regression)**.** For a given β, write

$$y_i - \mathbf{x}_i^T \beta = e_i^+ - e_i^-$$

where $e_i^+, e_i^- \geqslant 0$. e_i^+ and e_i^- are the positive and negative parts of $y_i - \mathbf{x}_i^T\beta$ and thus $|y_i - \mathbf{x}_i^T\beta| = e_i^+ + e_i^-$. Then

$$\sum_{i=1}^{n} |y_i - \mathbf{x}_i^T\beta| = \sum_{i=1}^{n} (e_i^+ + e_i^-)$$

Thus we can formulate L_1 estimation as a constrained minimization problem

$$\min \sum_{i=1}^{n} (e_i^+ + e_i^-) \text{ s.t. } y_i - \mathbf{x}_i^T\beta = e_i^+ - e_i^-$$

where $e_i^+, e_i^- \geqslant 0$ for $i = 1, \cdots, n$.

5.3.1 Active and Inactive Constraints

Suppose $g(\mathbf{x})$ is minimized subject to the constraints $g_j(\mathbf{x}) \geqslant 0$ for $j = 1, \cdots, k$ at $\mathbf{x}^*$. If $g_j(\mathbf{x}^*) = 0$, then the constraint $g_j(\mathbf{x})$ is active. If $g_j(\mathbf{x}^*) > 0$, then the constraint $g_j(\mathbf{x}) \geqslant 0$ is inactive. If a constraint is inactive, then it could be omitted without affecting the solution, and equality constraints are always active.

In the Lagrange multiplier-slack variable formulation, we have: (1) $z_j = 0$ if the constraint $g_j(\mathbf{x}) \geqslant 0$ is active, and (2) $z_j > 0$ and $\lambda_j = 0$ if the constraint $g_j(\mathbf{x}) \geqslant 0$ is inactive.

5.3.2 Interior Point Algorithm

We want to find $\mathbf{x}^*$ to minimize $g(\mathbf{x})$ subject to $g_1(\mathbf{x}) \geqslant 0, \cdots, g_k(\mathbf{x}) \geqslant 0$. Define the constraint set

$$\mathcal{C} = \{\mathbf{x} : g_1(\mathbf{x}) \geqslant 0, \cdots, g_k(\mathbf{x}) \geqslant 0\}$$

For interior point method, we assume the interior of $\mathcal{C}$ to be an open set, which rules out equality constraints. We minimize

$$g(\mathbf{x}) + \sum_{j=1}^{k} \phi_r(g_j(\mathbf{x}))$$

for some sequence of functions $\{\phi_r\}$. We need to choose $\{\phi_r\}$ so that for each $r > 0$, the penalized objective function is minimized at $\mathbf{x}_r^* \in \text{int}(\mathcal{C})$, and $\mathbf{x}_r^* \to \mathbf{x}^*$ as $r \downarrow 0$ where $g(\mathbf{x}^*) = \min\{g(\mathbf{x}) : \mathbf{x} \in \mathcal{C}\}$. Besides, $\phi_r(t)$ should be differentiable for $t > 0$, and ideally, $\phi_r(t)$ should be independent of the scaling of the constraints.

We can choose logarithmic barrier function

$$\phi_r(t) = -r\ln(t)$$

Since $\ln(at) = \ln(t) + \ln(a)$ so we do not need to worry about the scaling of the constraints, i.e., if we replace $g_j(\mathbf{x}) \geqslant 0$ by $ag_j(\mathbf{x}) \geqslant 0$ for some $a > 0$, we only change the penalized objective by a constant.

Now we minimize $g(\mathbf{x})$ over $\mathbf{x} \in \mathcal{C}$ where the interior of $\mathcal{C}$ is open:

1. Define a sequence $\{r_n\}$ where $r_n \to 0$ as $n \to \infty$.
2. Define $\mathbf{x}_n^*$ to minimize

$$g(\mathbf{x}) - r_n \sum_{j=1}^{k} \ln(g_j(\mathbf{x}))$$

over $\mathbf{x} \in \text{int}(\mathcal{C})$. The sequence $\{r_n\}$ is called a schedule. The choice of $\{r_n\}$ is important and we cannot take r_n to 0 too quickly.

Example 5.8. We minimize $g(x, y) = x + y$ subject to $y - x^2 \geqslant 0$ and $x \geqslant 0$. Hence,

$$\mathcal{C} = \{(x, y) : y \geqslant x^2, x \geqslant 0\}$$

Note that $g(x, y)$ is minimized at $x^* = y^* = 0$, i.e., (x^*, y^*) lies on the boundary of $\mathcal{C}$. We can use interior point method: For $r > 0$, minimize

$$\mathcal{G}_r(x, y) = x + y - r\ln(y - x^2) - r\ln(x)$$

over the interior of $\mathcal{C}$. Take partial derivatives:

$$\begin{aligned}\frac{\partial}{\partial x}\mathcal{G}_r(x, y) &= 1 + \frac{2xr}{y - x^2} - \frac{r}{x} \\ \frac{\partial}{\partial y}\mathcal{G}_r(x, y) &= 1 - \frac{r}{y - x^2}\end{aligned}$$

Setting the partial derivatives to 0 and we get

$$\begin{aligned}x_r^* &= \frac{-1 + \sqrt{1 + 8r}}{4} \\ y_r^* &= r + \frac{(-1 + \sqrt{1 + 8r})^2}{16}\end{aligned}$$

where (x_r^*, y_r^*) lie in the interior of $\mathcal{C}$. As $r \downarrow 0$, we have $x_r^* \to 0$ and $y_r^* \to 0$.

5.3.2.1 Convergence of the Interior Point Algorithm

Convergence of the interior point algorithm follows from the properties of the barrier function for r close to 0 and t close to 0: (a) for fixed $r > 0, -r\ln(t) \to \infty$ as $t \downarrow 0$, and (b) for fixed $t > 0, -r\ln(t) \to 0$ as $r \downarrow 0$.

Property (a) pushes the minimizer of the penalized objective away from the boundary of $\mathcal{C}$. Property (b) implies that if $\mathbf{x} \in \text{int}(\mathcal{C})$, then

$$g(\mathbf{x}) - r\sum_{j=1}^{k} \ln(g_j(\mathbf{x})) \approx g(\mathbf{x})$$

if r is close to 0, and thus if g is a continuous function then for small r, $g(\mathbf{x}_r^*) \approx g(\mathbf{x}^*)$, which suggests $\mathbf{x}_r^* \approx \mathbf{x}^*$.

5.3.2.2 Choice of r_0 and $\{r_n\}$

The success of the interior point algorithm relies on being able to minimize the penalized functions efficiently, and thus it may be tempting to set r_0 close to 0 and/or have r_n converging to 0 quickly.

If r_0 is too small, then it may be difficult to minimize the penalized objective function. If we take r_n to 0 too quickly, then we may run into problems.

In practice, we often use $\mathbf{x}_n^*$ as an initial estimate for the penalized objective function with $r = r_{n+1}$. Typically, $r_{n+1} = \gamma r_n$ for some $\gamma \in (0, 1)$. The choice of γ involves a trade-off: If γ is close to 1, then $\mathbf{x}_n^*$ serves as a very good initial estimate, but r_n converges to 0 very slowly.

5.3.2.3 Quantile Regression Estimation

For a given $\tau \in (0, 1)$, we define the ***regression quantile estimate*** $\widehat{\beta}$ to minimize

$$\sum_{i=1}^{n} \rho_\tau(y_i - \mathbf{x}_i^T\beta)$$

where $\rho_\tau(x) = \begin{cases} (\tau - 1)x, & x \leqslant 0 \\ \tau x, & x > 0 \end{cases}$.

An example in R:

```
library(quantreg)
x = rnomr(100)
y = 4 + 3*x + rnorm(100)
r = rq(y~x, tau=1/4)
```

Note $\widehat{\beta}$ also minimizes

$$\sum_{i=1}^{n} \frac{1}{\tau}\rho_\tau(y_i - \mathbf{x}_i^T\beta)$$

For $\tau > 0$, we have

$$\frac{1}{\tau}\rho_\tau(x) = \begin{cases} (\tau - 1)x/\tau, & x \leqslant 0 \\ x, & x > 0 \end{cases}$$

and taking $\tau \downarrow 0$, we have

$$\frac{1}{\tau}\rho_\tau(x) \to \rho_0(x) = \begin{cases} \infty, & x < 0 \\ x, & x \geqslant 0 \end{cases}$$

If we minimize

$$\sum_{i=1}^{n} \rho_0(y_i - \mathbf{x}_i^T\beta)$$

then the objective function equals ∞ if any residual $y_i - \mathbf{x}_i^T\beta < 0$. Thus our minimization problem becomes (***extreme quantile regression***)

$$\min \sum_{i=1}^{n}(y_i - \mathbf{x}_i^T\beta) \text{ s.t. } y_i \geqslant \mathbf{x}_i^T\beta, i = 1, \cdots, n$$

We can use interior point method: For $r > 0$, the penalized objective function is

$$\mathcal{G}_r(\beta) = -\sum_{i=1}^{n}\mathbf{x}_i^T\beta - r\sum_{i=1}^{n}\ln(y_i - \mathbf{x}_i^T\beta)$$

The gradient of $\mathcal{G}_r(\beta)$ is

$$\nabla\mathcal{G}_r(\beta) = \sum_{i=1}^{n}\left(\frac{r}{y_i - \mathbf{x}_i^T\beta} - 1\right)\mathbf{x}_i$$

The Hessian matrix of $\mathcal{G}_r(\beta)$ is

$$\nabla^2\mathcal{G}_r(\beta) = \sum_{i=1}^{n}\frac{r}{(y_i - \mathbf{x}_i^T\beta)^2}x_ix_i^T$$

For each r, we need to solve the equation

$$\nabla\mathcal{G}_r(\widehat{\beta}) = \mathbf{0}$$

for $\widehat{\beta}$, which can be done using N-R, IRLS, or gradient descent algorithm.

An alternative is to define $\widehat{\beta}$ to (***constrained least squares***)

$$\min \sum_{i=1}^{n}(y_i - \mathbf{x}_i^T\beta)^2 \text{ s.t. } y_i \geqslant \mathbf{x}_i^T\beta, i = 1, \cdots, n$$

which penalizes large values of $y_i - \mathbf{x}_i^T\beta$ more heavily than the extreme regression quantile procedure. In practice, the difference between the two methods is usually very small.

5.3.3 Exterior Point Algorithm

If int($\mathcal{C}$) is not open, we can use a variation of the interior point method, or relax the constraints.

Suppose we want to minimize $g(\mathbf{x})$ s.t. $\mathbf{x} \in \mathcal{C}$. Define

$$\overline{g}(\mathbf{x}) = \begin{cases} g(\mathbf{x}), & x \in \mathcal{C} \\ \infty, & \mathbf{x} \notin \mathcal{C} \end{cases}$$

Alternatively, we can write

$$\overline{g}(\mathbf{x}) = g(\mathbf{x}) + I_{\mathcal{C}}(\mathbf{x})$$

where

$$I_{\mathcal{C}}(\mathbf{x}) = \begin{cases} 0, & \mathbf{x} \in \mathcal{C} \\ \infty, & \mathbf{x} \notin \mathcal{C} \end{cases}$$

and $I_{\mathcal{C}}$ is called the ***characteristic function*** of $\mathcal{C}$.

We first approximate $I_{\mathcal{C}}(\mathbf{x})$ by a sequence of functions $I_r(\mathbf{x})$ s.t. $I_r(\mathbf{x}) = 0$ for $\mathbf{x} \in \mathcal{C}$ and for $\mathbf{x} \notin \mathcal{C}, I_r(\mathbf{x}) \to \infty$ as $r \downarrow 0$. Given $I_r(\mathbf{x})$, we can define $\mathbf{x}_r^*$ to minimize

$$\overline{g}_r(\mathbf{x}) = g(\mathbf{x}) + I_r(\mathbf{x})$$

where $\mathbf{x}_r^*$ may lie outside $\mathcal{C}$. If $\mathcal{C} = \{\mathbf{x} : g_1(\mathbf{x}) \geqslant 0, \cdots, g_k(\mathbf{x}) \geqslant 0\}$, then we can define

$$I_r(\mathbf{x}) = \frac{1}{r}\sum_{j=1}^{k}\{\min[0, g_j(\mathbf{x})]\}^2$$

Note that $I_r(\mathbf{x}) = 0$ if $\mathbf{x} \in \mathcal{C} : \overline{g}_r(\mathbf{x}) = g(\mathbf{x})$ for $\mathbf{x} \in \mathcal{C}$; if $g_j(\mathbf{x}) = -\varepsilon$, then $I_r(\mathbf{x}) \geqslant \dfrac{\varepsilon^2}{r} \to \infty$ as $r \downarrow 0$; if $\mathbf{x}^* \in \operatorname{int}(\mathcal{C})$, then $\mathbf{x}^*$ minimizes $\overline{g}_r(\mathbf{x})$ for all $r > 0$.

Note that $g_j(\mathbf{x}) = 0$ is equivalent to $g_j(\mathbf{x}) \geqslant 0$ and $-g_j(\mathbf{x}) \geqslant 0$. If $g_j(\mathbf{x}) > 0$, then $\min[0, g_j(\mathbf{x})] = 0$ and $\min[0, -g_j(\mathbf{x})] = -g_j(\mathbf{x})$. If $g_j(\mathbf{x}) < 0$, then $\min[0, g_j(\mathbf{x})] = g_j(\mathbf{x})$ and $\min[0, -g_j(\mathbf{x})] = 0$. Therefore,

$$\{\min[0, g_j(\mathbf{x})]\}^2 + \{\min[0, -g_j(\mathbf{x})]\}^2 = [g_j(\mathbf{x})]^2$$

5.3.3.1 Constrained Quadratic Minimization

Suppose we want to minimize $\mathbf{x}^T A\mathbf{x}$ s.t. $C\mathbf{x} = \mathbf{b}$ where A is symmetric positive definite. Assume $\mathcal{C} = \{\mathbf{x} : C\mathbf{x} = \mathbf{b}\}$ and $\operatorname{rank}(C) < \operatorname{rank}(A)$. In this case, all the constraints are equality constraints so

$$I_r(\mathbf{x}) = \frac{1}{r}(C\mathbf{x} - \mathbf{b})^T(C\mathbf{x} - \mathbf{b})$$

and

$$\overline{g}_r(\mathbf{x}) = \mathbf{x}^T A\mathbf{x} + \frac{1}{r}(C\mathbf{x} - \mathbf{b})^T(C\mathbf{x} - \mathbf{b})$$

The gradient of $\overline{g}_r(\mathbf{x})$ is

$$\nabla\overline{g}_r(\mathbf{x}) = 2A\mathbf{x} + \frac{2}{r}(C^T C\mathbf{x} - C^T\mathbf{b})$$

and $\mathbf{x}_r^*$ satisfies

$$(rA + C^T C)\mathbf{x}_r^* = C^T\mathbf{b}$$

Since A is symmetric positive definite, $rA + C^T C$ is symmetric positive definite for all $r > 0$. However, for small $r, rA + C^T C$ may be ill-conditioned.

If A^{-1} is easily computable, then we can use the Woodbury matrix identity to evaluate $(rA + C^T C)^{-1}$:

$$(rA + C^T C)^{-1} = \frac{1}{r}A^{-1} - \frac{1}{r^2}A^{-1}C^T\left(I + \frac{CA^{-1}C^T}{r}\right)^{-1}CA^{-1}$$

If $n = \dim(\mathbf{x})$ is large, then the Gauss-Seidel algorithm is useful for computing $\mathbf{x}_r^*$.

Example 5.9. Suppose we want to minimize

$$\sum_{i=1}^{100} x_i^2 = \mathbf{x}^T\mathbf{x} \text{ s.t. } x_1 + \cdots + x_{100} = \mathbf{x}^T\mathbf{1} = 100$$

The exterior point method gives $\mathbf{x}_r^*$ minimizes

$$\overline{g}_r(\mathbf{x}) = \mathbf{x}^T\mathbf{x} + \frac{1}{r}(\mathbf{x}^T\mathbf{1} - 100)^2$$

Since

$$\nabla \overline{g}_r(\mathbf{x}) = 2\mathbf{x} + \frac{2}{r}(\mathbf{x}^T\mathbf{1} - 100)\mathbf{1}$$

then $\mathbf{x}_r^*$ satisfies

$$(rI + \mathbf{1}\mathbf{1}^T)\mathbf{x}_r^* = 100\mathbf{1}$$

with $\mathbf{x}_r^* = a(r)\mathbf{1}$.

5.3.3.2 Pros and Cons of Exterior Point Algorithm

1. Pros:

   (1) Relaxation allows for greater flexibility.

   (2) $\mathbf{x}_r^*$ is relatively easy to compute for each $r > 0$.

   (3) We do not need to check that $\mathbf{x} \in \mathcal{C}$.

2. Cons:

   (1) $\mathbf{x}_r^*$ may violate at least one of the constraints.

   (2) Implementation of exterior point methods may be difficult - we need to think more carefully about the scaling of constraints.

5.3.4 Combining Interior and Exterior Point Algorithm

Suppose we want to minimize $g(\mathbf{x})$ subject to mixed constraints

$$\begin{aligned} g_1(\mathbf{x}) &\geqslant 0 \\ &\vdots \\ g_l(\mathbf{x}) &\geqslant 0 \\ g_{l+1}(\mathbf{x}) &= 0 \\ &\vdots \\ g_k(\mathbf{x}) &= 0 \end{aligned}$$

We can combine interior and exterior point method to minimize

$$g(\mathbf{x}) - r\sum_{j=1}^{l} \ln(g_j(\mathbf{x})) + \frac{1}{r}\sum_{j=l+1}^{k} [g_j(\mathbf{x})]^2$$

5.4 Fixed Point Algorithm

Start with a single variable $h : \mathbb{R} \to \mathbb{R}$ and we want to find x^* s.t. $h(x^*) = 0$. We consider iterative methods of the form

$$x_k = \phi(x_{k-1}), k = 1, 2, \cdots$$

for some function ϕ. If the algorithm converges ($x_k \to x^*$) then $x^* = \phi(x^*)$, where x^* is called a ***fixed point*** of ϕ.

We re-express $h(x^*) = 0$: If $a(x) \neq 0$ for all x, then

$$h(x^*) = 0 \Rightarrow \frac{h(x^*)}{a(x^*)} = 0 \Rightarrow \frac{h(x^*)}{a(x^*)} + x^* = x^*$$

which suggests that we can define $\phi(x) = x + \dfrac{h(x)}{a(x)}$ and so

$$x_k = x_{k-1} + \frac{h(x_{k-1})}{a(x_{k-1})}$$

5.4.1 Analysis of Fixed Point Algorithm

Suppose $x_k = \phi(x_{k-1})$ and $x_k \to x^*$ where x^* is a fixed point of ϕ. Assume $|x_{k-1} - x^*| = \delta$. The Taylor series expansion of $|x_k - x^*|$ is

$$\begin{aligned}|\phi(x_{k-1}) - \phi(x^*)| &= |\phi'(\xi)(x_{k-1} - x^*)| \\ &= |\phi'(\xi)||x_{k-1} - x^*|\end{aligned}$$

and so $|x_k - x^*| < \delta$ if $|\phi'(\xi)| < 1$, which suggests that the convergence of the fixed point iteration depends on $|\phi'(x)|$ for $x \in [a, b]$.

5.4.2 Fixed Point Theorem

Theorem 5.1 (Fixed Point Theorem)**.** Suppose $\phi(x)$ has a fixed point $x^* \in [a, b]$, i.e., $\phi(x^*) = x^*$ and $|\phi'(x)| \leqslant \lambda < 1$ for $x \in [a, b]$. If $x_0 \in (a, b)$ and $x_k = \phi(x_{k-1})$ for $k \geqslant 1$, then $x_k \to x^*$ as $k \to \infty$.

Proof. From the mean value theorem

$$x_1 - x^* = \phi(x_0) - \phi(x^*) = \phi'(\xi)(x_0 - x^*)$$

where ξ lies between x_0 and x^*. Thus $\xi \in [a, b]$ and so $|\phi'(\xi)| \leqslant \lambda < 1$. Iterating the process, we have

$$|x_k - x^*| \leqslant \lambda^k \underbrace{|x_0 - x^*|}_{\leqslant b-a} \to 0$$

□

Definition 5.3 (Linear Convergence Rate)**.** A fixed point algorithm satisfying the conditions of fixed point theorem has

$$|x_k - x^*| \leqslant \lambda |x_{k-1} - x^*|, \lambda < 1$$

which is called a linear convergence rate.

Note. We have a faster convergence rate if $|\phi'(x)|$ is closer to 0.

5.4.3 Multivariate Fixed Point Theorem

Suppose we want to solve $\mathbf{h}(\mathbf{x}^*) = \mathbf{0}$ for some function $\mathbf{h} : \mathbb{R}^p \to \mathbb{R}^p$. Consider iterative algorithms of the form

$$\mathbf{x}_k = \mathbf{x}_{k-1} + A^{-1}(\mathbf{x}_{k-1})\mathbf{h}(\mathbf{x}_{k-1}) = \phi(\mathbf{x}_{k-1})$$

where the matrix $A(\mathbf{x})$ is invertible for all $\mathbf{x}$.

We can replace $\phi'(x)$ by the Jacobian matrix of ϕ

$$J(\mathbf{x}) = \begin{pmatrix} [\nabla \phi_1(\mathbf{x})]^T \\ \vdots \\ [\nabla \phi_p(\mathbf{x})]^T \end{pmatrix}$$

where $J(\mathbf{x})$ is a $p \times p$ matrix. We would like to have

$$\|\mathbf{x}_k - \mathbf{x}^*\| < \|\mathbf{x}_{k-1} - \mathbf{x}^*\|$$

for some norm. From the definition of matrix norms, we have

$$\|\mathbf{x}_k - \mathbf{x}^*\| \leqslant \|J(\xi)\| \|\mathbf{x}_{k-1} - \mathbf{x}^*\|$$

Thus if $\|J(\mathbf{x})\| < 1$ for $\mathbf{x}$ in some large enough set, then we will have

$$\|\mathbf{x}_k - \mathbf{x}^*\| < \|\mathbf{x}_{k-1} - \mathbf{x}^*\|$$

and $\mathbf{x}_k \to \mathbf{x}^*$.

Note 1. Iterative algorithms require a leap of faith in their implementation.

Note 2. When p is large, we prefer simple algorithms (such as gradient descent).

5.5 Newton-Raphson Algorithm

Suppose we solve $h(x^*) = 0$ where $h(x)$ is differentiable, and we take $a(x) = -h'(x)$ so that

$$x_k = x_{k-1} - \frac{h(x_{k-1})}{h'(x_{k-1})}$$

We have

$$\phi'(x) = 1 - \frac{[h'(x)]^2 - h(x)h''(x)}{[h'(x)]^2} = \frac{h(x)h''(x)}{[h'(x)]^2}$$

If $h'(x) \neq 0$ for x close to x^*, we have $\phi'(x^*) = 0$ and $|\phi'(x)| \leqslant \varepsilon < 1$ for $|x - x^*| < \delta$. Thus, if x_{k-1} is close to x^*, we have a ***quadratic convergence rate***

$$|x_k - x^*| \leqslant C|x_{k-1} - x^*|^2$$

Note 1. N-R applies only when x_k is close to x^*.

Note 2. δ could be small and the radius of convergence of N-R could be very narrow.

Note 3. We may lose the quadratic convergence rate if $h'(x^*) = 0$.

Example 5.10 (Logistic Location MLE)**.** Suppose data $x_1, \cdots, x_n$ is from a Logistic distribution with unknown location (center) θ :

$$f(x;\theta) = \frac{\exp(x - \theta)}{[1 + \exp(x - \theta)]^2}$$

The log-likelihood function is

$$\ln(\mathcal{L}(\theta)) = \sum_{i=1}^{n} [x_i - \theta - 2\ln(1 + \exp(x_i - \theta))]$$

Differentiating w.r.t. θ, the MLE $\widehat{\theta}$ satisfies

$$S(\widehat{\theta}) = \sum_{i=1}^{n} \left[\frac{2\exp(x_i - \widehat{\theta})}{1 + \exp(x_i - \widehat{\theta})} - 1 \right] = 0$$

where $S(\widehat{\theta})$ is called the ***score function***. Using N-R:

$$\widehat{\theta}_k = \widehat{\theta}_{k-1} + \frac{S(\widehat{\theta}_{k-1})}{H(\widehat{\theta}_{k-1})}$$

where

$$H(\theta) = 2\sum_{i=1}^{n} \frac{\exp(x_i - \theta)}{[1 + \exp(x_i - \theta)]^2}$$

Note 1. The Logistic distribution is symmetric around 0, and thus a natural choice of initial estimate $\widehat{\theta}_0$ is a measure of the center (sample mean or sample median) of the distribution.

Note 2. If we assume that $x_1, \cdots, x_n$ come from a Logistic distribution, then we can use $H(\widehat{\theta})$, the ***observed Fisher information*** to estimate the standard error of $\widehat{\theta}$

$$\widehat{\mathrm{SE}}(\widehat{\theta}) = [H(\widehat{\theta})]^{-1/2}$$

5.5.1 Variations on Newton-Raphson

5.5.1.1 Partial N-R Steps

For some $\alpha > 0$,

$$x_k = x_{k-1} - \alpha \frac{h(x_{k-1})}{h'(x_{k-1})} = \phi_\alpha(x_{k-1})$$

with

$$\phi'_\alpha(x) = 1 - \alpha + \alpha \phi'_1(x)$$

Note 1. By adjusting α, we may be able to increase the radius of convergence.

Note 2. Typically, $\alpha(0, 1)$, but $\alpha > 1$ is possible.

5.5.1.2 Secant Method

Replace $h'(x_{k-1})$ in N-R by the secant approximation

$$\widetilde{h}'(x_{k-1}) = \frac{h(x_{k-1}) - h(x_{k-2})}{x_{k-1} - x_{k-2}}$$

which has a slightly slower convergence rate

$$|x_k - x^*| \leqslant C|x_{k-1} - x^*|^{1.618}$$

5.5.2 Newton-Raphson and Reweighted Least Squares

We have

$$S(\theta) = \sum_{i=1}^{n} S_i(\theta) = \sum_{i=1}^{n} \frac{\partial}{\partial \theta} \ln(f(x_i; \theta))$$

and

$$H(\theta) = \sum_{i=1}^{n} H_i(\theta) = -\sum_{i=1}^{n} \frac{\partial^2}{\partial \theta^2} \ln(f(x_i; \theta))$$

The N-R iterations are

$$\widehat{\theta}_k = \widehat{\theta}_{k-1} + \frac{S(\widehat{\theta}_{k-1})}{H(\widehat{\theta}_{k-1})}$$

We have

$$\widehat{\theta}_k = \sum_{i=1}^{n} w_i(\widehat{\theta}_{k-1}) \left(\widehat{\theta}_{k-1} + \frac{S_i(\widehat{\theta}_{k-1})}{H_i(\widehat{\theta}_{k-1})} \right)$$

where

$$w_i(\theta) = \frac{H_i(\theta)}{H_1(\theta) + \cdots + H_n(\theta)}$$

Note that $w_1(\theta) + \cdots + w_n(\theta) = 1$ for all θ. We can think of N-R update at step k as a weighted average of pseudo-data

$$\widehat{\theta}_{k-1} + \frac{S_1(\widehat{\theta}_{k-1})}{J_1(\widehat{\theta}_{k-1})}, \cdots, \widehat{\theta}_{k-1} + \frac{S_n(\widehat{\theta}_{k-1})}{H_n(\widehat{\theta}_{k-1})}$$

Thus at convergence, $\widehat{\theta}$ is the weighted average of

$$\widehat{\theta} + \frac{S_1(\widehat{\theta})}{H_1(\widehat{\theta})}, \cdots, \widehat{\theta} + \frac{S_n(\widehat{\theta})}{H_n(\widehat{\theta})}$$

which can be used to obtain an estimate of the standard error of $\widehat{\theta}$ (***sandwich estimate***):

$$\widehat{\mathrm{SE}}(\widehat{\theta}) = \left[\frac{S_1^2(\widehat{\theta}) + \cdots + S_n^2(\widehat{\theta})}{[H_1(\widehat{\theta}) + \cdots + H_n(\widehat{\theta})]^2}\right]^{1/2}$$

5.5.3 Multivariate N-R Algorithm

We take

$$A(\mathbf{x}) = -\begin{pmatrix} [\nabla h_1(\mathbf{x})]^T \\ \vdots \\ [\nabla h_p(\mathbf{x})]^T \end{pmatrix}$$

5.5.4 Application: Maximum Likelihood Estimation

Suppose data $x_1, \cdots, x_n$ with log-likelihood function $\ln(\mathcal{L}(\theta)) = \ln(f(x_1, \cdots, x_n; \theta))$ defined for $\theta \in \Theta$ (parameter space). Assume that the MLE $\widehat{\theta}$ satisfies

$$\nabla \ln(\mathcal{L}(\widehat{\theta})) = \mathbf{S}(\widehat{\theta}) = \mathbf{0}$$

$H(\theta)$ is the matrix of negative second derivatives:

$$h_{ij}(\theta) = -\frac{\partial^2}{\partial\theta_i \partial\theta_j} \ln(\mathcal{L}(\theta))$$

The N-R iteration is

$$\widehat{\theta}_k = \widehat{\theta}_{k-1} + [H(\widehat{\theta}_{k-1})]^{-1}\mathbf{S}(\widehat{\theta}_{k-1})$$

We can estimate the variance-covariance matrix of $\widehat{\theta}$ by $[H(\widehat{\theta})]^{-1}$.

Example 5.11 (Weibull Distribution)**.** Suppose $x_1, \cdots, x_n$ are independent observations from a Weibull distribution (which is often used to model lifetime data or failure data). The density function is

$$f(x; \alpha, \sigma) = \left(\frac{\alpha}{\sigma}\right)\left(\frac{x}{\sigma}\right)^{\alpha-1} \exp\left[-\left(\frac{x}{\sigma}\right)^{\alpha}\right], x > 0$$

where $\alpha > 0$ and $\sigma > 0$ are unknown parameters: α is a shape parameter and σ is a scale parameter. Note that

$$\mathbb{E}[X] = \int_0^\infty x f(x; \alpha, \sigma) \mathrm{d}x = \sigma\Gamma\left(1 + \frac{1}{\alpha}\right)$$

The ***hazard function*** is given by

$$h(x) = \frac{f(x; \alpha, \sigma)}{1 - F(x; \alpha, \sigma)} = \frac{\alpha}{\sigma^\alpha} x^{\alpha-1}$$

The MLEs of α and σ satisfy the likelihood equations

$$\sum_{i=1}^{n}\left[\frac{1}{\widehat{\alpha}}+\ln\left(\frac{x_i}{\widehat{\sigma}}\right)-\ln\left(\frac{x_i}{\widehat{\sigma}}\right)\left(\frac{x_i}{\widehat{\sigma}}\right)^{\widehat{\alpha}}\right]=0$$

and

$$\sum_{i=1}^{n}\left[\frac{\widehat{\alpha}}{\widehat{\sigma}}\left(\frac{x_i}{\widehat{\sigma}}\right)^{\widehat{\alpha}}-\frac{\widehat{\alpha}}{\widehat{\sigma}}\right]=0$$

To compute the MLEs, we can use the N-R algorithm to solve the likelihood equations.

We need to choose proper initial values for α and σ in the N-R algorithm since it can be critical in guaranteeing that the N-R algorithm converges. We consider two basic criteria:

1. $\widehat{\alpha}_0$ and $\widehat{\sigma}_0$ should be easy to compute.
2. $\widehat{\alpha}_0$ and $\widehat{\sigma}_0$ should be good estimates of α and σ.

Criterion 2 is much more important, and we can use method of moments estimates: We find functions g_1 and g_2 s.t.

$$\mathbb{E}[g_1(X)]=\psi_1(\alpha,\sigma)$$
$$\mathbb{E}[g_2(X)]=\psi_2(\alpha,\sigma)$$

We estimate $\mathbb{E}[g_1(X)]$ and $\mathbb{E}[g_2(X)]$ by sample means and define $\widehat{\alpha}_0$ and $\widehat{\sigma}_0$ as follows:

$$\frac{1}{n}\sum_{i=1}^{n}g_1(x_i)=\psi_1(\widehat{\alpha}_0,\widehat{\sigma}_0)$$
$$\frac{1}{n}\sum_{i=1}^{n}g_2(x_i)=\psi_2(\widehat{\alpha}_0,\widehat{\sigma}_0)$$

If $\mathrm{Var}[g_1(X)]$ and $\mathrm{Var}[g_2(X)]$ are both finite, then $\widehat{\alpha}_0$ and $\widehat{\sigma}_0$ are good estimates. Note that if we can estimate α by $\widehat{\alpha}_0$, then we can define

$$\widehat{\sigma}_0=\frac{\overline{x}}{\Gamma(1+1/\widehat{\alpha}_0)}$$

We can use a Weibull plot to define $\widehat{\alpha}_0$: Note that

$$1-F(x;\alpha,\sigma)=\exp\left[-\left(\frac{x}{\sigma}\right)^{\alpha}\right]$$

and so

$$\ln(-\ln(1-F(x)))=\alpha\ln(x)-\alpha\ln(\sigma)\Rightarrow\ln(x)=\frac{1}{\alpha}\ln(-\ln(1-F(x)))+\ln(\sigma)$$

i.e., $\ln(x)$ is linear function of $\ln(-\ln(1-F(x)))$. The Weibull plot is constructed as follows:

1. Order the data from smallest to largest: $x_{(1)}\leqslant\cdots\leqslant x_{(n)}$.
2. Plot $\ln(x_{(i)})$ versus $t_i=\ln\left(-\ln\left(1-\dfrac{i}{n+1}\right)\right)$ for $i=1,\cdots,n$.

For Weibull data, the points should lie close to a straight line with slope $\dfrac{1}{\alpha}$ and intercept $\ln(\sigma)$. We can use the Weibull plot to obtain $\widehat{\alpha}_0$ and possibly $\widehat{\sigma}_0$ using least squares to estimate the slope and intercept. The R code for Weibull plot is:

```
weibullplot = function(x, line=T){
  x = sort(x)
  n = length(x)
  s = 1 - c(1:n)/(n+1)
  plot(log(-log(s)), log(x), pch=20)
  r = lm(log(x)~log(-log(s)))
  alpha = 1/r$coef[2]
  sigma = exp(r$coef[1])
  if (line){
    abline(r, col="red", lwd=3)
    title(main=paste("alpha = ", round(alpha, 2), "sigma = ", round(sigma, 2)),
       cex=0.5)
    }
  else{
    a = list(alpha=alpha, sigma=sigma)
    a
    }
  }
```

5.6 Fisher Scoring

We replace the matrix $H(\theta)$ in the N-R algorithm by its expected value $\mathcal{H}(\theta) = \mathbb{E}_\theta[H(\theta)]$:

$$\widehat{\theta}_k = \widehat{\theta}_{k-1} + [\mathcal{H}(\widehat{\theta}_{k-1})]^{-1}\mathbf{S}(\widehat{\theta}_{k-1})$$

Fisher scoring tends to converge slower than N-R and may be more sensitive to initial values. In exponential family models where the joint density has the general form

$$f(\mathbf{x};\theta) = \exp[\theta^T\mathbf{T}(\mathbf{x}) - c(\theta) + d(\mathbf{x}))]$$

Fisher scoring and N-R are essentially the same.

Example 5.12 (Weibull Distribution)**.** The form of $H(\alpha, \sigma)$ is rather complicated: It depends on $y_i = \dfrac{x_i}{\sigma}$ for $i = 1, \cdots, n$ via the following sums:

$$\sum_{i=1}^n \ln(y_i) y_i^\alpha, \sum_{i=1}^n y_i^\alpha, \sum_{i=1}^n [\ln(y_i)]^2 y_i^\alpha$$

We can compute $\mathcal{H}(\alpha, \sigma)$ by taking expected values:

$$\mathcal{H}(\alpha, \sigma) = n \begin{pmatrix} \dfrac{6+\pi^2-12\gamma+6\gamma^2}{6\alpha^2} & \dfrac{\gamma-1}{\sigma} \\ \dfrac{\gamma-1}{\sigma} & \dfrac{\alpha^2}{\sigma^2} \end{pmatrix}$$

where $\gamma = 0.5772\cdots$ is Euler's constant.

5.6.1 Quasi-Likelihood Estimation

Suppose the data is $\{(\mathbf{x}_i, y_i) : i = 1, \cdots, n\}$. For a generalized linear model, we specify the variance function $V(\mu_i)$ and the link function $\phi(\mu_i) = \mathbf{x}_i^T \beta$. If the variance function corresponds to a particular distribution, then we can estimate β by maximum likelihood estimation; otherwise, we define a ***quasi-likelihood*** function and maximize it.

We define the function $\mathcal{Q}(y; \mu)$ by the differential equation

$$\frac{\partial}{\partial \mu} \mathcal{Q}(y; \mu) = \frac{y - \mu}{V(\mu)}$$

Given $\{(\mathbf{x}_i, y_i)\}$ with functions ϕ and V, we maximize

$$\sum_{i=1}^{n} \mathcal{Q}(y_i; \mu_i) = \sum_{i=1}^{n} \mathcal{Q}(y_i; \phi^{-1}(\mathbf{x}_i^T \beta))$$

w.r.t. β. Note that the gradient of $\mathcal{Q}(y_i; \mu_i)$ w.r.t. β is

$$\nabla \mathcal{Q}(y_i; \phi^{-1}(\mathbf{x}_i^T \beta)) = \underbrace{\left[\frac{\partial}{\partial \mu_i} \mathcal{Q}(y_i; \mu_i)\right]}_{(y_i - \mu_i)/V(\mu_i)} \underbrace{[\nabla \phi^{-1}(\mathbf{x}_i^T \beta)]}_{[\phi'(\phi^{-1}(\mathbf{x}_i^T \beta))]^{-1} \mathbf{x}_i}$$

The maximum quasi-likelihood estimate $\widehat{\beta}$ satisfies

$$\mathbf{S}(\widehat{\beta}) = \sum_{i=1}^{n} \left[\frac{y_i - \mu_i(\widehat{\beta})}{V(\mu_i(\widehat{\beta}))\phi'(\mu_i(\widehat{\beta}))}\right] \mathbf{x}_i = \mathbf{0}$$

where $\mu_i(\beta) = \phi^{-1}(\mathbf{x}_i^T \beta)$. We can solve these equations using N-R or Fisher scoring, and the Fisher scoring algorithm is simple:

$$\widehat{\beta}_k = \widehat{\beta}_{k-1} + [\mathcal{H}(\widehat{\beta}_{k-1})]^{-1} \mathbf{S}(\widehat{\beta}_{k-1})$$

where

$$\mathcal{H}(\beta) = \sum_{i=1}^{n} \frac{\mathbf{x}_i \mathbf{x}_i^T}{[\phi'(\mu_i(\beta))]^2 V(\mu_i(\beta))}$$

5.6.2 Fisher Scoring and Iteratively Reweighted Least Squares

We want to find a stable implementation of the Fisher scoring algorithm. Define

$$X = \begin{pmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix}$$

and $W(\beta)$ to be a diagonal matrix with diagonal elements

$$w_i(\beta) = \frac{1}{[\phi'(\mu_i(\beta))]^2 V(\mu_i(\beta))}$$

for $i = 1, \cdots, n$. Then

$$\mathcal{H}(\beta) = X^T W(\beta) X$$

We can define $\widehat{\beta}_k$ as a weighted least squares estimate minimizing

$$\sum_{i=1}^{n} w_i(\widehat{\beta}_{k-1})(z_i(\widehat{\beta}_{k-1}) - \mathbf{x}_i^T\beta)^2$$

w.r.t. β Note that

$$\begin{aligned}\widehat{\beta}_k &= \widehat{\beta}_{k-1} + [\mathcal{H}(\widehat{\beta}_{k-1})]^{-1}\mathbf{S}(\widehat{\beta}_{k-1}) \\ &= (X^TW(\widehat{\beta}_{k-1})X)^{-1}[X^TW(\widehat{\beta}_{k-1})X\widehat{\beta}_{k-1} + \mathbf{S}(\widehat{\beta}_{k-1})] \\ &= (X^TW(\widehat{\beta}_{k-1})X)^{-1}X^TW(\widehat{\beta}_{k-1})\mathbf{z}(\widehat{\beta}_{k-1})\end{aligned}$$

where

$$z_i(\beta) = \mathbf{x}_i^T\beta + \phi'(\mu_i(\beta))(y_i - \mu_i(\beta))$$

We call $\{z_i(\beta)\}$ as the ***adjusted dependent variable***.

Example 5.13 (Logistic Regression)**.** Suppose the model is $\{Y_i\}$ binary with $\mathbb{E}[Y_i|\mathbf{x}_i] = \mu_i, \mathrm{Var}[Y_i|\mathbf{x}_i] = \mu_i(1-\mu_i) = V(\mu_i)$, and logit link

$$\ln\left(\frac{\mu_i}{1-\mu_i}\right) = \mathbf{x}_i^T\beta$$

so that

$$\mu_i(\beta) = \frac{\exp(\mathbf{x}_i^T\beta)}{1+\exp(\mathbf{x}_i^T\beta)}$$

We have

$$w_i(\beta) = \mu_i(\beta)(1-\mu_i(\beta))$$

and

$$z_i(\beta) = \mathbf{x}_i^T\beta + \frac{y_i - \mu_i(\beta)}{\mu_i(\beta)(1-\mu_i(\beta))}$$

5.6.2.1 Application: Poisson Regression

Suppose the model is $Y_i \sim$ Poisson with mean μ_i, with $V(\mu) = \mu$ and log link $\phi(\mu_i) = \ln(\mu_i) = \mathbf{x}_i^T\beta$ Then $\mu_i = \mu_i(\beta) = \exp(\mathbf{x}_i^T\beta)$. For IRLS algorithm, we define

$$w_i(\beta) = \frac{1}{[\phi'(\mu_i)]^2V(\mu_i)} = \mu_i(\beta)$$

and

$$z_i(\beta) = \mathbf{x}_i^T\beta + \frac{y_i - \mu_i(\beta)}{\mu_i(\beta)}$$

5.7 Application: M-Estimation

Least squares estimation is very sensitive to outliers, and the M-estimation is to minimize

$$\sum_{i=1}^{n} \rho(y_i - \mathbf{x}_i^T\beta)$$

where $\dfrac{\rho(x)}{x^2} \to 0$ as $|x| \to \infty$. E.g., $\rho(x) = |x|^r$ for $1 \leqslant r < 2$.

Suppose we define $\widehat{\beta}$ to minimize the objective function of M-estimation, where $\rho(t)$ is a twice differentiable function s.t. $\rho(0) = 0$ and $\rho(t)$ increases as $|t|$ increases. We define $\psi(t) = \rho'(t)$ and $\psi'(t) = \rho''(t)$. M-estimates are often defined in terms of $\psi(t)$. $\widehat{\beta}$ satisfies the condition

$$\sum_{i=1}^{n} \psi(y_i - \mathbf{x}_i^T \beta)\mathbf{x}_i = \mathbf{0}$$

Example 5.14 (Huber Estimate)**.** Define $\rho(t)$ so that $\rho(t) = Ct^2$ for $|t|$ close to 0 and $\rho(t) \approx C|t|$ for larger $|t|$. Define $\psi(t) = \psi_c(t)$ as follows:

$$\psi_c(t) = \begin{cases} -c, & t < -c \\ t, & |t| \leqslant c \\ c, & t > c \end{cases}$$

where c is a tuning parameter. Note that $\psi_c'(t)$ is not defined at $t = \pm c$. The corresponding $\rho_c(t)$ is defined as follows:

$$\rho_c(t) = \begin{cases} \dfrac{t^2}{2}, & |t| < c \\ c|t| - \dfrac{c^2}{2}, & |t| > c \end{cases}$$

5.7.1 N-R for M-Estimation

The M-estimate $\widehat{\beta}$ satisfies the equation

$$\mathbf{S}(\widehat{\beta}) = \sum_{i=1}^{n} \psi(y_i - \mathbf{x}_i^T \beta)\mathbf{x}_i = \mathbf{0}$$

Since we assume that ψ' exists, we can use the N-R algorithm to compute $\widehat{\beta}$:

$$\widehat{\beta}_k = \widehat{\beta}_{k-1} + [H(\widehat{\beta}_{k-1})]^{-1}\mathbf{S}(\widehat{\beta}_{k-1})$$

where

$$H(\beta) = \sum_{i=1}^{n} \psi'(y_i - \mathbf{x}_i^T \beta)\mathbf{x}_i\mathbf{x}_i^T$$

5.7.2 IRLS for M-Estimation

We write

$$H(\beta) = X^T W(\beta) X$$

where $W(\beta)$ is a diagonal matrix with diagonals

$$w_i(\beta) = \psi'(y_i - \mathbf{x}_i^T \beta)$$

for $i = 1, \cdots, n$. Then we have

$$\widehat{\beta}_k = (X^T W(\widehat{\beta}_{k-1})X)^{-1} X^T W(\widehat{\beta}_{k-1})\mathbf{z}(\widehat{\beta}_k)$$

where

$$z_i(\beta) = \mathbf{x}_i^T \beta + \frac{\psi(y_i - \mathbf{x}_i^T \beta)}{\psi'(y_i - \mathbf{x}_i^T \beta)}$$

for $i = 1, \cdots, n$.

5.7.3 Fisher Scoring Modification

We can replace the matrix $W(\beta)$ and the vector $\mathbf{z}(\beta)$ by $W(\beta) = \tau(\beta)I$ where

$$\tau(\beta) = \frac{1}{n}\sum_{i=1}^{n}\psi'(y_i - \mathbf{x}_i^T\beta)$$

and

$$\mathbf{z}(\beta) = \mathbf{x}_i^T\beta + \frac{\psi(y_i - \mathbf{x}_i^T\beta)}{\tau(\beta)}$$

Example 5.15 (Huber Estimate)**.** $\widehat{\beta}$ satisfies

$$\sum_{i=1}^{n}\psi_c(y_i - \mathbf{x}_i^T\beta)\mathbf{x}_i = \mathbf{0}$$

where

$$\psi_c(t) = \begin{cases} -c, & t < -c \\ t, & |t| \leqslant c \\ c, & t > c \end{cases}$$

Then $\psi_c'(t) = 1$ for $|t| < c$ and $\psi_c'(t) = 0$ for $|t| > c$. For IRLS, we have $w_i(\beta) = 0$ if $|y_i - \mathbf{x}_i^T\beta| > c$, and $w_i(\beta) = 1$ and $z_i(\beta) = y_i$ if $|y_i - \mathbf{x}_i^T\beta| \leqslant c$. The R code is:

```
huber = function(x, y, c, niter=10){
   # Ues LS as initial estimate
   r = lm(y~x)
   if (missing(c)) c = median(abs(r$resid)) * 1.35
   for (i in 1:niter){
      resid = r$resid
      fitted = r$fitted
      wt = ifelse(abs(resid)>c, 0, 1)
      z = y
      r = lm(y~x, weights=wt)
      }
   beta = r$coef
   beta
   }
```

5.8 L_∞ Estimation and Lawson's Algorithm

The L_∞ estimation is to minimize

$$\max_{1\leqslant i\leqslant n}|y_i - \mathbf{x}_i^T\beta| = \|\mathbf{y} - X\beta\|_\infty$$

w.r.t. β. L_∞ estimation is useful if we know $y_i \approx \mathbf{x}_i^T\beta$ for some β and the approximation error is very small and bounded. For example, minimize median$|y_i - \mathbf{x}_i^T\beta|$, which is the least median of squares (LMS) estimate, is an L_∞ estimate on a subset of the data.

The L_∞ is essentially determined by $p+1$ points where p is the length of the vector β. If these points are $(\mathbf{x}_{i_1}, y_{i_1}), \cdots, (\mathbf{x}_{i_{p+1}}, y_{i_{p+1}})$ then

$$\max_{1\leqslant i\leqslant n}|y_i - \mathbf{x}_i^T\widehat{\beta}| = |y_{i_j} - \mathbf{x}_{i_j}^T\widehat{\beta}|$$

for $j = 1, \cdots, p+1$. We successively downweight observations with small absolute residuals until only $p+1$ points have positive weights. The Lawson's algorithm is:

1. Define $\widehat{\beta}_0$ to be the LS estimate and weights $w_1^{(0)} = \cdots = w_n^{(0)} = \dfrac{1}{n}$.
2. For $k = 1, 2, \cdots$:

(1) Define weights $\{w_i^{(k)}\}$ to be

$$w_i^{(k)} = \frac{w_i^{(k-1)}|y_i - \mathbf{x}_i^T \widehat{\beta}_{k-1}|}{\sum_{j=1}^n w_j^{(k-1)}|y_j - \mathbf{x}_j^T \widehat{\beta}_{k-1}|}$$

for $i = 1, \cdots, n$.

(2) Define $\widehat{\beta}_k$ to minimize

$$\sum_{i=1}^n w_i^{(k)}(y_i - \mathbf{x}_i^T \beta)^2$$

We need to exclude points where $w_i^{(k)} = 0$ for some k, and at convergence we will have

$$w_i^* = \frac{w_i^*|y_i - \mathbf{x}_i^T \widehat{\beta}|}{\sum_{j=1}^n w_j^*|y_j - \mathbf{x}_i^T \widehat{\beta}|}$$

for some non-negative weights $w_q^*, \cdots, w_n^*$.

5.9 Gradient Descent Algorithm

Suppose we want to minimize $g(\mathbf{x})$ over some open set $\mathcal{O}$ and $\mathbf{x}^*$ satisfies

$$\nabla g(\mathbf{x}^*) = \mathbf{0}$$

For small $\varepsilon > 0$, we have

$$g(\mathbf{x} + \varepsilon\mathbf{u}) \approx g(\mathbf{x}) + \varepsilon\mathbf{u}^T \nabla g(\mathbf{x})$$

where the right hand side above decreases most rapidly over unit vectors $\mathbf{u}$ taking

$$\mathbf{u} = -\frac{\nabla g(\mathbf{x})}{\|\nabla g(\mathbf{x})\|}$$

which suggests the gradient descent algorithm

$$\mathbf{x}_k = \mathbf{x}_{k-1} - \varepsilon \nabla g(\mathbf{x}_{k-1}) = \phi(\mathbf{x}_{k-1})$$

where the parameter $\varepsilon > 0$ is called the ***learning rate*** in machine learning, and ε may depend on k, i.e., $\varepsilon = \varepsilon_k$.

5.9.1 Convergence of Gradient Descent

The convergence of gradient descent depends on the eigenvalues of

$$\phi(\mathbf{x}) = \mathbf{x} - \varepsilon \nabla g(\mathbf{x})$$

with $\mathbf{x}_k = \phi(\mathbf{x}_{k-1})$. We require the eigenvalues of the Jacobian matrix of ϕ, $J_\phi(\mathbf{x})$ to lie in $(-1, 1)$. If $\lambda_1(\mathbf{x}) \geqslant \cdots \geqslant \lambda_p(\mathbf{x}) > 0$ are the eigenvalues of $J_{(\nabla g)}(\mathbf{x})$ then the eigenvalues of $J_\phi(\mathbf{x})$ are $1 - \varepsilon\lambda_j(\mathbf{x})$ for $i = 1, \cdots, p$.

The maximum size of ε depends on the largest eigenvalue of $J_{(\nabla g)}(\mathbf{x})$. If the largest eigenvalue is large then ε must be small for convergence.

The convergence rate depends on the smallest eigenvalue $J_{(\nabla g)}$. If the smallest eigenvalue is very small then we may have slow convergence.

Example 5.16 (Quadratic Function)**.** Suppose we want o minimize

$$g(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b}$$

where A is symmetric positive definite with eigenvalues $\lambda_1 \geqslant \cdots \geqslant \lambda_p > 0$, and $\nabla g(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$. The gradient descent iteration is

$$\mathbf{x}_k = \mathbf{x}_{k-1} - \varepsilon(A\mathbf{x}_{k-1} - \mathbf{b}) = (I - \varepsilon A)\mathbf{x}_{k-1} + \varepsilon\mathbf{b}$$

We need to take $0 < \varepsilon < \dfrac{2}{\lambda_1}$ for convergence.

Suppose we take $\varepsilon < \dfrac{1}{\lambda_1}$. All the eigenvalues of $I - \varepsilon A$ lie in $(0, 1)$, then the maximum eigenvalue of $I - \varepsilon A$ is $1 - \varepsilon\lambda_p$ which is bounded below by

$$1 - \frac{\lambda_p}{\lambda_1} = 1 - \frac{1}{\kappa(A)}$$

where $\kappa(A)$ is the condition number of A.

5.9.2 Accelerated Gradient Descent

We can add a second order (momentum) term to $\mathbf{x}_k$ at each step:

$$\mathbf{x}_k \leftarrow \mathbf{x}_k + \underbrace{\mu_k(\mathbf{x}_k - \mathbf{x}_{k-1})}_{\text{Momentum}}$$

for some sequence $\{\mu_k\}$.

5.9.2.1 Heavy Ball Method

For $k = 1, 2, \cdots$, define

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mu_k(\mathbf{x}_k - \mathbf{x}_{k-1}) - \varepsilon\nabla g(\mathbf{x}_k)$$

We may also allow ε to varying with k, i.e., $\varepsilon = \varepsilon_k$.

5.9.2.2 Nesterov's Method

For $k = 1, 2, \cdots$, define

$$\begin{aligned}\mathbf{y}_k &= \mathbf{x}_k + \mu_k(\mathbf{x}_k - \mathbf{x}_{k-1})\\ \mathbf{x}_{k+1} &= \mathbf{y}_k - \varepsilon\nabla g(\mathbf{y}_k)\end{aligned}$$

where $0 < \mu_k < 1$ and $\mu_k \to 1$ as $k \to \infty$. For example, take $\mu_k = \dfrac{k-1}{k+2} = 1 - \dfrac{3}{k+2}$.

We have less flexibility than we have for ordinary gradient descent: We require ε small enough so that the eigenvalues of $I - J_{(\nabla g)}(\mathbf{x})$ lie in $[0, 1)$ rather than $(-1, 1)$.

5.9.2.3 Special Case: Quadratic Objective Function

Suppose

$$g(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b}$$

where A is symmetric positive definite, $\nabla g(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$, and $\mathbf{x}^*$ minimizing g satisfies $A\mathbf{x}^* = \mathbf{b}$. We can write an accelerated gradient descent iteration as

$$\begin{pmatrix} \mathbf{x}_{k+1} \\ \mathbf{x}_k \end{pmatrix} = C_k \begin{pmatrix} \mathbf{x}_k \\ \mathbf{x}_{k-1} \end{pmatrix} + \varepsilon \begin{pmatrix} \mathbf{b} \\ \mathbf{0} \end{pmatrix}$$

where

$$C_k = \begin{pmatrix} (1+\mu_k)(I - \varepsilon A) & -\mu_k(I - \varepsilon A) \\ I & 0 \end{pmatrix}$$

Convergence and speed of convergence depends on the eigenvalues of C_k.

C_k is a $(2p) \times (2p)$ matrix and thus has $2p$ eigenvalues. We can determine the eigenvalues by solving

$$\det(C_k - \alpha I) = 0$$

where

$$\det(C_k - \alpha I) = \det(\alpha^2 I - (\alpha(1+\mu_k) - \mu_k)(I - \varepsilon A))$$

If μ_k is close to 1, then the eigenvalues of C_k are complex-valued and each eigenvalue of $I - \varepsilon A$ contributes a pair of eigenvalues of C_k: If $\theta_1 \geqslant \cdots \geqslant \theta_p \geqslant 0$ are the eigenvalues of $I - \varepsilon A$, then

$$\mu_k^{1/2}\theta_j^{1/2}\exp(\pm i\omega_j), j = 1, \cdots, p$$

are eigenvalues of C_k, where $\omega_1, \cdots, \omega_p$ are defined by

$$\cos(\omega_j) = \left(\frac{\theta_j}{\mu_k}\right)^{1/2}\left(\frac{\mu_k + 1}{2}\right)$$

and the moduli of these eigenvalues are $\mu_k^{1/2}\theta_k^{1/2}$ for $j = 1, \cdots, p$.

Roughly speaking, the convergence of Nesterov's method is determined by the speed at which $\prod_{k=1}^{m}(\mu_k^{1/2}\theta_1^{1/2})$ converges to 0.

5.9.2.4 Comparison

The ordinary gradient descent converges slowly when the condition number of A, $\kappa(A) = \dfrac{\lambda_1}{\lambda_p}$, is large. If $\varepsilon < \dfrac{1}{\lambda_1}$, then

$$\theta_1 > 1 - \frac{1}{\kappa(A)}$$

Take $\mu_k = 1 - \dfrac{3}{k+2}$. If $\mu_k^{1/2}\theta_1^{1/2} < \theta_1$, then $\mu_k^{1/2} < \theta_1^{1/2}$ or $\mu_k < \theta_1$. We will have $\mu_k < \theta_i$ if

$$\mu_k = 1 - \frac{3}{k+2} < 1 - \frac{1}{\kappa(A)}$$

or $k < 3\kappa(A) - 2$, which suggests that when $\kappa(A)$ is large, Nesterov's method is a significant improvement over ordinary gradient descent.

5.10 Robbins-Monro Procedure

5.10.1 Stochastic Approximation

Suppose we want to solve $\mathbf{h}(\mathbf{x}^*) = \mathbf{0}$ for $\mathbf{x}^*$. We can write

$$\mathbf{h}(\mathbf{x}) = \sum_{i=1}^{n} \mathbf{h}_i(\mathbf{x})$$

5.10.2 Robbins-Monro Procedure

Suppose we want to solve

$$\mathbb{E}_F[\mathbf{h}(X;\theta)] = \mathbf{0}$$

for θ. Assume that for each x, $\mathbf{h}(x;\theta)$ is the gradient (w.r.t. θ) of a convex function. The algorithm is

$$\theta_k = \theta_{k-1} - a_k \mathbf{h}(X_k;\theta_{k-1})$$

for some $\{a_k\}$ of positive numbers, and $X_1, \cdots$ are independent r.v.s. with distribution F. The conditions on $\{a_k\}$ are needed for convergence: $a_k \to 0$ as $k \to \infty$ but not too quickly or too slowly, i.e.,

1. $\sum_{k=1}^{\infty} a_k = \infty$.
2. $\sum_{k=1}^{\infty} a_k^2 < \infty$.

5.10.3 Application: Convex Minimization

Suppose we want to minimize $g(\mathbf{x})$ where $g(\mathbf{x}) = \sum_{i=1}^{n} g_i(\mathbf{x})$ and each g_i is convex and differentiable. If $\mathbf{x}^*$ minimizes $g(\mathbf{x})$, then

$$\nabla g(\mathbf{x}^*) = \sum_{i=1}^{n} \nabla g_i(\mathbf{x}^*) = \mathbf{0}$$

We can uas a stochastic approximation approach to approximate $\mathbf{x}^*$. We write the condition for $\mathbf{x}^*$ as an expected value:

$$\mathbf{0} = \frac{1}{n}\sum_{i=1}^{n} \nabla g_i(\mathbf{x}^*) = \mathbb{E}[\nabla g_{\mathcal{I}}(\mathbf{x}^*)]$$

where $P(\mathcal{I} = i) = \dfrac{1}{n}$ for $i = 1, \cdots, n$.

5.11 Stochastic Gradient Descent Algorithm

Suppose we want to minimize convex function g where

$$\nabla g(\mathbf{x}) = \sum_{i=1}^{n} \nabla g_i(\mathbf{x})$$

The algorithm is:

$$\mathbf{x}_k = \mathbf{x}_{k-1} - a_k \nabla g_{\mathcal{I}_k}(\mathbf{x}_{k-1})$$

where $\mathcal{I}_1, \cdots$ are independent r.v.s. with $P(\mathcal{I}_j = i) = \dfrac{1}{n}$ for $i = 1, \cdots, n$, $\sum_{k=1}^{\infty} a_k = \infty$, and $\sum_{k=1}^{\infty} a_k^2 < \infty$. Suppose $a_k = \dfrac{\alpha}{k}$ for some $\alpha > 0$:

1. α too small: Take small steps and convergence may be too slow - significant bias if m is too small, i.e., low variance and high bias.
2. α too large: Take large steps and possibly inflate the variance of the estimates, i.e., low bias and high variance.

Example 5.17 (Huber Estimate in Regression)**.** $\widehat{\beta}$ satisfies

$$\sum_{i=1}^{n} \psi_c(y_i - \mathbf{x}_i^T \beta)\mathbf{x}_i = \mathbf{0}$$

where

$$\psi_c(t) = \begin{cases} -c, & t < -c \\ t, & |t| \leqslant c \\ c, & t > c \end{cases}$$

The stochastic gradient descent iteration is:

$$\widehat{\beta}_k = \widehat{\beta}_{k-1} + a_k \psi(y_{\mathcal{I}_k} - \mathbf{x}_{\mathcal{I}_k}^T \widehat{\beta}_{k-1})$$

where $P(\mathcal{I}_k = i) = \dfrac{1}{n}$ for $i = 1, \cdots, n$. The R code for stochastic gradient descent is:

```
stochapprox = function(x, y, c, alpha=1, nrep=10000){
  x = cbind(1, x)
  p = ncol(x)
  beta = rep(0, p)
  betas = NULL
  n = length(y)
  for (k in 1:nrep){
    i = sample(c(1:n), size=1)
    resi = y[i] - sum(beta*x[i, ])
    psi = ifelse(abs(resi)<c, resi, c*sign(resi))
    beta = beta + alpha * psi * x[i, ] / k
    betas = rbind(betas, beta)
    }
  betas
  }
```

Bias decreases as k increases, and bias persists for $\alpha = 1$ even for large k. Variance decreases as k increases, and is larger for larger values of α.

5.12 Coordinate Descent Algorithm

Convergence for gradient descent is much faster, but the advantage of coordinate descent lies in its simplicity. However, in order to be used effectively, the one variable minimization must be easily computable.

5.12.1 Convergence of Coordinate Descent

Define $\widehat{x}_j^{(k+1)}$ to minimize

$$g(x_1^{(k)}, \cdots, x_{j-1}^{(k)}, x_j, x_{j+1}^{(k)}, \cdots, x_p^{(k)})$$

w.r.t. x_j where the other $p-1$ arguments are fixed. Set $x_i^{(k+1)} = x_i^{(k)}$ for $i \neq j$. We have $g(\mathbf{x}_{k+1}) \leqslant g(\mathbf{x}_k)$ and the sequence $\{g(\mathbf{x}_k)\}$ converges as $k \to \infty$ since it is non-increasing. However, $\{\mathbf{x}_k\}$ does not necessarily converge to $\mathbf{x}^*$ minimizing g (not even for convex functions).

Theorem 5.2. Suppose $g(\mathbf{x})$ is a convex function that can be decomposed as

$$g(\mathbf{x}) = g_{\mathrm{d}}(\mathbf{x}) + g_{\mathrm{nd}}(\mathbf{x})$$

where $g_{\mathrm{d}}(\mathbf{x})$ is differentiable for all $\mathbf{x}$ and $g_{\mathrm{nd}}(\mathbf{x})$ is not differentiable for some $\mathbf{x}$. If $g_{\mathrm{nd}}(\mathbf{x})$ is separable in the sense that

$$g_{\mathrm{nd}}(\mathbf{x}) = \phi_1(x_1) + \cdots + \phi_p(x_p)$$

then the coordinate descent sequence $\{\mathbf{x}_k\}$ converges as $k \to \infty$ to $\mathbf{x}^*$ minimizing $g(\mathbf{x})$.

5.12.2 Application: The LASSO in Linear Regression

Suppose the model is $Y_i = \mathbf{x}_i^T\beta + \varepsilon_i$ for $i = 1, \cdots, n$. We define $\widehat{\beta}(\lambda)$ to minimize

$$\sum_{i=1}^{n}(y_i - \mathbf{x}_i^T\beta)^2 + \lambda \underbrace{\sum_{j=1}^{p}|\beta_j|}_{\|\beta\|_1}$$

where $\lambda > 0$. As we increase λ, more components of $\widehat{\beta}(\lambda)$ are exactly equal to 0. Note that the objective function is convex since it is a sum of convex functions, the non-differentiable component of the objective function is separable and thus coordinate descent is a feasible optimization method.

For the LASSO, coordinate descent involves minimizing functions of the form

$$g(x) = ax^2 - 2bx + c + \lambda|x|$$

so that g is not differentiable at $x = 0$. The sub-gradient of $g(x)$ is

$$\begin{aligned}\partial g(x) &= \frac{\mathrm{d}}{\mathrm{d}x}(ax^2 - 2bx + c) + \lambda\partial|x| \\ &= \begin{cases} 2ax - 2b + \lambda\mathrm{sgn}(x), & x \neq 0 \\ [-\lambda - 2b, \lambda - 2b], & x = 0 \end{cases}\end{aligned}$$

x^* minimizes g iff $0 \in \partial g(x^*)$ and thus $g(x)$ is minimized at $x = 0$ if $-\lambda - 2b \leqslant 0 \leqslant \lambda - 2b$ or equivalently if

$$|b| \leqslant \frac{\lambda}{2}$$

Otherwise for $|b| > \frac{\lambda}{2}$, g is minimized when $g'(x^*) = 0$:

$$2ax^* - 2b + \lambda\mathrm{sgn}(x^*) = 0 \Rightarrow x^* = \frac{1}{a}\left(b - \frac{\lambda}{2}\mathrm{sgn}(x^*)\right)$$

Note that if $b > \dfrac{\lambda}{2}, \text{sgn}(x^*) = 1$; if $b < -\dfrac{\lambda}{2}, \text{sgn}(x^*) = -1$. Thus $\text{sgn}(x^*) = \text{sgn}(b)$. Hence, if $|b| > \dfrac{\lambda}{2}$,

$$x^* = \frac{1}{a}\left(b - \frac{\lambda}{2}\text{sgn}(b)\right)$$

We typically normalize the predictors to have mean 0 and the same variance, which allows us to estimate β_0 by $\overline{y}$ and redefine y_i to be $y_i - \overline{y}$. We fix β_j for $j \neq l$ and minimize the objective function w.r.t. β_l :

$$g_l(\beta_l) = n\beta_l^2 - 2\beta_l\left[\sum_{i=1}^{n} x_{il}\left(y_i - \overline{y} - \sum_{j\neq l}\beta_j x_{ij}\right)\right] + \lambda|\beta_l| + R_l$$

where R_l does not depend on β_l. The minimizer of g_l depends on

$$B_l(\beta_j : j \neq l) = \sum_{i=1}^{n} x_{il}\left(y_i - \overline{y} - \sum_{j\neq l}\beta_j x_{ij}\right)$$

In particular, it depends on whether this term is greater than $\dfrac{\lambda}{2}$, less than $-\dfrac{\lambda}{2}$, or lies in the interval $\left[-\dfrac{\lambda}{2}, \dfrac{\lambda}{2}\right]$. Note that $\dfrac{B_l(\beta_j : j \neq l)}{n}$ is essentially a covariance between predictor l and a residual based on all the other predictors.

Define $B_l(\widehat{\beta}_j : j \neq l)$ be the value of B_l based on current estimates of β_j for $j \neq l$. Then the updated estimate of β_l for coordinate descent is

$$\widehat{\beta}_l \leftarrow 0 \text{ if } |B_l(\widehat{\beta}_j : j \neq l)| < \frac{\lambda}{2}$$

otherwise

$$\widehat{\beta}_l \leftarrow \frac{1}{n}\left(B_l(\widehat{\beta}_j : j \neq l) - \frac{\lambda}{2}\text{sgn}[B_l(\widehat{\beta}_j : j \neq l)]\right)$$

Note that the tuning parameter λ adjusts the amount of shrinkage towards 0, and for λ sufficiently large, all the estimates of $\beta_1, \cdots, \beta_p$ will be exactly 0.

5.12.2.1 Special Case: Orthogonal Designs

Suppose $\sum_{i=1}^{n} x_{ij}x_{il} = 0$ for all $j \neq l$. In this case, the LS estimates of $\beta_1, \cdots, \beta_p$ have a very simple form under normalization

$$\widehat{\beta}_j = \widehat{\beta}_j(0) = \frac{1}{n}\sum_{i=1}^{n} x_{ij}y_i$$

For the LASSO, we have

$$\widehat{\beta}_j(\lambda) = \begin{cases} 0, & |\widehat{\beta}_j| \leqslant \frac{\lambda}{2} \\ \text{sgn}(\widehat{\beta}_j(0))\left(|\widehat{\beta}_j(0)| - \frac{\lambda}{2}\right), & \text{otherwise} \end{cases}$$

5.12.2.2 Comparison to Ridge Regression

Define $\widetilde{\beta}_1(\lambda), \cdots, \widetilde{\beta}_p(\lambda)$ to minimize

$$\sum_{i=1}^{n}(y_i - \overline{y} - \mathbf{x}_i^T\beta)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

and assume orthogonal (i.e., uncorrelated) predictors. Note that ridge regression is typically useful for highly correlated predictors. The ridge regression estimates are defined as follows:

$$\widetilde{\beta}_j(\lambda) = \frac{1}{n+\lambda}\sum_{i=1}^{n} x_{ij}y_i = \frac{n}{n+\lambda}\underbrace{\widetilde{\beta}_j(0)}_{\text{LS}}$$

Hence the ridge regression still shrinks LS estimates (with λ controlling the shrinkage) but we do not have shrinkage to 0 as we have for the LASSO.

5.12.2.3 Alternative Formulation of the LASSO

We can reformulate the LASSO estimate as follows: minimize

$$\sum_{i=1}^{n}(y_i - \overline{y} - \mathbf{x}_i^T\beta)^2 \text{ s.t. } \sum_{j=1}^{p}|\beta_j| \leqslant t$$

for some $t \geqslant 0$. For each t, there is an equivalent λ, and we can take $t \leqslant \sum_{j=1}^{p}|\widehat{\beta}_j(0)|$.

5.12.2.4 The LASSO Plot

Define for each λ,

$$s(\lambda) = \frac{\|\widehat{\beta}(\lambda)\|_1}{\|\widehat{\beta}(0)\|_1}$$

where $s(\lambda) \leqslant 1$. We plot $\widehat{\beta}_j(\lambda)$ versus $s(\lambda)$ for $j = 1, \cdots, p$. The LASSO plot gives an idea of the relative importance of each predictor as the model size decreases to 0.

We need to evaluate $\widehat{\beta}(\lambda)$ for a sufficiently rich set of λ values.

For some large m, we define $\Delta = \dfrac{\lambda_{\max}}{m}$ and set $\lambda_k = k\Delta$ for $k = 0, \cdots, m$. We compute LS estimates: $\lambda = \lambda_0 = 0$. For $k = 1, \cdots, m$, compute $\widehat{\beta}(\lambda_k)$ using $\widehat{\beta}(\lambda_{k-1})$ as initial estimates for coordinate descent.

5.12.3 Important Feature of the LASSO

The LASSO can be used when $p > n$, i.e., there are more predictors than observations. In this case, the LS estimates are non-unique, but for $\lambda > 0$, LASSO estimates are typically unique.

Define $\mathcal{B}$ to be the set of β minimizing

$$\sum_{i=1}^{n}(y_i - \overline{y} - \mathbf{x}_i^T\beta)^2$$

We can use $\widehat{\beta}(\lambda)$ to obtain a unique LS estimate:

$$\lim_{\lambda \downarrow 0} \widehat{\beta}(\lambda) = \{\beta^* \in \mathcal{B} : \|\beta^*\|_1 \leqslant \|\beta\|_1, \forall \beta \in \mathcal{B}\}$$

In other words, $\widehat{\beta}(\lambda)$ converges to $\beta^* \in \mathcal{B}$ having minimum L_1 norm.

5.12.3.1 Elastic Net

We can combine the LASSO with ridge regression, i.e., we minimize

$$\sum_{i=1}^{n} (y_i - \overline{y} - \mathbf{x}_i^T \beta) + \lambda_1 \|\beta\|_1 + \lambda_2 \|\beta\|_2^2$$

for some $\lambda_1, \lambda_2 \geqslant 0$.

Note. The $\|\beta\|_2^2$ penalty makes the level sets of the quadratic part of the objective function more spherical. Potentially, the $\|\beta\|_1$ part of the penalty is more effective, which is useful if the predictors are highly correlated.

5.13 Expectation-Maximization (EM) Algorithm

Suppose we have observed data $x_1, \cdots, x_n$, complete data $y_1, \cdots, y_m$, and complete data log-likelihood

$$\ln \mathcal{L}_c(\theta) = \ln \mathcal{L}_c(\theta; Y_1, \cdots, Y_m)$$

The EM algorithm is:

1. Expectation-step: Given $\widehat{\theta}_k$, define

$$L_k(\theta) = \mathbb{E}_{\widehat{\theta}_k}[\ln L_c(\theta; Y_1, \cdots, Y_m) | x_1, \cdots, x_m]$$

2. Maximization-step: Define $\widehat{\theta}_{k+1}$ to maximize $L_k(\theta)$.
3. The E- and M-steps are iterated until convergence.

If $\ln \mathcal{L}(\theta)$ is multi-modal, then $\widehat{\theta}_k$ will sometimes converge to a local but not global maximizer, which can be avoided by choosing $\widehat{\theta}_0$ appropriately. Besides, if $\widehat{\theta}_k \to \widehat{\theta}$, the convergence can be very slow.

Example 5.18 (Interval Censored Exponential Data)**.** We observe $(l_1, u_1), \cdots, (l_n, u_n)$ where $l_i \leqslant Y_i \leqslant u_i, i = 1, \cdots, n$ where $Y_1, \cdots, Y_n$ are independent exponential r.v.s. with

$$f(y; \lambda) = \lambda \exp(-\lambda y), y \geqslant 0$$

where $\lambda > 0$. The complete data likelihood is

$$\ln \mathcal{L}_c(\lambda) = n \ln(\lambda) - \lambda \sum_{i=1}^{n} Y_i$$

The EM algorithm is:

1. E-step: Given $\widehat{\lambda}_k$, compute

$$L_k(\lambda) = \mathbb{E}_{\widehat{\lambda}_k}\left[n \ln(\lambda) - \lambda \sum_{i=1}^{n} Y_i | l_i \leqslant Y_i \leqslant u_i, i = 1, \cdots, n \right]$$

2. M-step: Find $\widehat{\lambda}_{k+1}$ to maximize $L_k(\lambda)$ for $\lambda > 0$.

For the E-step, note that

$$L_k(\lambda) = n\ln(\lambda) - \lambda \sum_{i=1}^{n} \mathbb{E}_{\widehat{\lambda}_k}[Y_i | l_i \leqslant Y_i \leqslant u_i]$$

The conditional density of Y_i given $l_i \leqslant Y_i \leqslant u_i$ is

$$g_i(y;\lambda) = \frac{\lambda\exp(-\lambda y)}{\exp(-\lambda u_i) - \exp(-\lambda l_i)}, l_i \leqslant y \leqslant u_i$$

and so

$$\begin{aligned}\mathbb{E}_\lambda[Y_i | l_i \leqslant Y_i \leqslant u_i] &= \int_{l_i}^{u_i} y g_i(y;\lambda)\mathrm{d}y \\ &= \frac{1}{\lambda} + \frac{l_i\exp(-\lambda l_i) - u_i\exp(-\lambda u_i)}{\exp(-\lambda l_i) - \exp(-\lambda u_i)} \\ &= \widehat{y}_i(\lambda)\end{aligned}$$

Thus

$$L_k(\lambda) = n\ln(\lambda) - \lambda \sum_{i=1}^{n} \widehat{y}_i(\widehat{\lambda}_k)$$

and the M-step is

$$\widehat{\lambda}_{k+1} = \left(\frac{1}{n}\sum_{i=1}^{n} \widehat{y}_i(\widehat{\lambda}_k)\right)^{-1}$$

5.13.1 Mixture Model and the EM Algorithm

Suppose we observe $x_1, \cdots, x_n$ from a density

$$f(x;\theta,\lambda) = \lambda_1 f_1(x;\theta) + \cdots + \lambda_r f_r(x;\theta)$$

where λ and θ are unknown parameters. We also assume that $\lambda_1, \cdots, \lambda_r$ are non-negative with $\lambda_1 + \cdots + \lambda_r = 1$. The observed data log-likelihood is

$$\ln\mathcal{L}(\theta,\lambda) = \sum_{i=1}^{n} \ln(\lambda_1 f_1(x_1;\theta) + \cdots + \lambda_r f_r(x_i;\theta))$$

which is difficult to maximize in general.

The complete data are

$$\mathbf{Y}_i = (X_i, \Delta_{i1}, \cdots, \Delta_{ir})$$

where $\Delta_{ij} = 0$ or 1 with $\Delta_{i1} + \cdots + \Delta_{ir} = 1$ and $P(\Delta_{i1} = \delta_1, \cdots, \Delta_{ir} = \delta_r) = \lambda_1^{\delta_1} \cdots \lambda_r^{\delta_r}$. The conditional density of X_i given $\Delta_{i1} = \delta_1, \cdots, \Delta_{ir} = \delta_r$ is

$$f_1(x;\theta)^{\delta_1} \cdots f_r(x;\theta)^{\delta_r}$$

The likelihood contribution $\mathbf{y}_i = (x_1, \Delta_{i1}, \cdots, \Delta_{ir})$ is

$$\prod_{j=1}^{r} [\lambda_j f_j(x_i;\theta)]^{\Delta_{ij}}$$

The complete data log-likelihood is

$$\ln \mathcal{L}_c(\theta, \lambda) = \sum_{i=1}^{n} \sum_{j=1}^{r} \Delta_{ij} [\ln(\lambda_j) + \ln f_j(x_i; \theta)]$$

The EM algorithm now boils down to computing

$$\mathbb{E}_{(\theta,\lambda)}[\Delta_{ij} | X_i = x_i]$$

for given (θ, λ).

1. E-step: We can use Bayes theorem

$$\begin{aligned}\mathbb{E}_{(\theta,\lambda)}[\Delta_{ij} | X_i = x_i] &= P_{(\theta,\lambda)}(\Delta_{ij} = 1 | X_i = x_i) \\ &= \frac{P(\Delta_{ij} = 1) P(X_i = x_i | \Delta_{ij} = 1)}{\sum_{l=1}^{r} P(\Delta_{il} = 1) P(X_i = x_i | \Delta_{il} = 1)} \\ &= \frac{\lambda_j f_j(x_i; \theta)}{\sum_{l=1}^{r} \lambda_l f_l(x_i; \theta)} \\ &= \widehat{\delta}_{ij}(\theta, \lambda)\end{aligned}$$

Note that $\sum_{j=1}^{r} \widehat{\delta}_{ij}(\theta, \lambda) = 1$ for all θ and λ. We impute $\Delta_{i1}, \cdots, \Delta_{ir}$ by

$$\widehat{\delta}_{ij}(\widehat{\theta}_k, \widehat{\lambda}_k) = \frac{\widehat{\lambda}_j^{(k)} f_j(x_i; \widehat{\theta}_k)}{\sum_{l=1}^{r} \widehat{\lambda}_l^{(k)} f_l(x_i; \widehat{\theta}_k)}$$

for $j = 1, \cdots, r$.

2. M-step: Define $\widehat{\theta}_{k+1}$ and $\widehat{\lambda}_{k+1}$:

$$\widehat{\lambda}_{k+1} = \frac{1}{n} \sum_{i=1}^{n} \widehat{\delta}_i(\widehat{\theta}_k, \widehat{\lambda}_k)$$

and $\widehat{\theta}_{k+1}$ maximizes

$$\sum_{i=1}^{n} \sum_{j=1}^{r} \widehat{\delta}_{ij}(\widehat{\theta}_k, \widehat{\lambda}_k) \ln f_j(x_i; \theta)$$

w.r.t. θ.

5.13.2 Application: Mixture Density Estimation

Suppose $x_1, \cdots, x_n$ are independent observations from some density $f(x)$ on $[0, 1]$. We approximate $f(x)$ by a mixture of Beta densities

$$f(x) = \lambda_1 f_1(x) + \cdots + \lambda_r f_r(x)$$

where

$$f_j(x) = \frac{r!}{(j-1)!(r-j)!} x^{j-1} (1-x)^{r-j}, 0 \leqslant x \leqslant 1$$

When $\lambda_1 = \cdots = \lambda_r = \dfrac{1}{r}$, then $f(x) = 1$, i.e., Unif$(0, 1)$ density.

5.13.3 Application: Zero-Inflated Distribution

A mixture of a discrete distribution $f_0(x;\theta)$ on the integers $\{0,1,\cdots\}$ and a distribution putting all its mass at 0. $x_1,\cdots,x_n$ are independent observations from the probability mass function

$$f(x;\theta,\lambda)=\begin{cases}\lambda+(1-\lambda)f_0(0;\theta), & x=0\\ (1-\lambda)f_0(x;\theta), & x=1,2,\cdots\end{cases}$$

Define $y=\sum_{i=1}^{n} I(x_i=0)$ which contains information about λ. Information about θ mainly comes from $\{x_i : x_i>0\}$.

Suppose we know Z is the number of observations from the population of 0s, and $Z\sim \text{Binom}(n,\lambda)$. If $Z=z$, then the number of 0s in the population with mass function $f_0(x;\theta)$ is $y-z$. Then we have

$$\mathcal{L}_c(\lambda,\theta)=\binom{n}{z}\lambda^z(1-\lambda)^{n-z}[f_0(0;\theta)]^{y-z}\prod_{x_i>0} f_0(x_i;\theta)$$

For the E-step of the EM algorithm, we need

$$\mathbb{E}_{(\lambda,\theta)}[Z|\mathbf{X}=\mathbf{x}]=\widehat{z}(\lambda,\theta)=\frac{\lambda y}{\lambda+(1-\lambda)f_0(0;\theta)}$$

1. E-step: Substitute $\widehat{\lambda}_k$ and $\widehat{\theta}_k$ into $\widehat{z}(\lambda,\theta)$.
2. M-step: Substitute $\widehat{z}(\widehat{\lambda}_k,\widehat{\theta}_k)$ for z in $\mathcal{L}_c$:

$$\widehat{\lambda}_{k+1}=\frac{\widehat{z}(\widehat{\lambda}_k,\widehat{\theta}_k)}{n}$$

and $\widehat{\theta}_{k+1}$ maximizes

$$L_k(\theta)=[y-\widehat{z}(\widehat{\lambda}_k,\widehat{\theta}_k)]\ln f_0(0;\theta)+\sum_{x_i>0}\ln f_0(x_i;\theta)$$

Example 5.19. Suppose

$$f_0(x;\theta)=\frac{\exp(-\theta)\theta^x}{x!}, x=0,1,\cdots$$

Given $x_1,\cdots,x_n$ from the zero-inflated Poisson distribution, and define

$$y=\sum_{i=1}^{n} I(x_i=0)$$

In this case, we have

$$\widehat{z}(\lambda,\theta)=\frac{\lambda y}{\lambda+(1-\lambda)\exp(-\theta)}$$

The EM updates given $\widehat{\lambda}_k$ and $\widehat{\theta}_k$ are

$$\begin{aligned}\widehat{\lambda}_{k+1}&=\frac{\widehat{z}(\widehat{\lambda}_k,\widehat{\theta}_k)}{n}\\ \widehat{\theta}_{k+1}&=\frac{1}{n-\widehat{z}(\widehat{\lambda}_k,\widehat{\theta}_k)}\sum_{x_i>0}x_i=\frac{1}{n-\widehat{z}(\widehat{\lambda}_k,\widehat{\theta}_k)}\sum_{i=1}^{n}x_i\end{aligned}$$

5.13.4 The MM Algorithm

The EM algorithm is a special case of an MM algorithm.

Minorization-Maximization: Suppose we want to maximize $g(\mathbf{x})$. For some $\mathbf{x}_k$ define $g_k(\mathbf{x})$ s.t. $g_k(\mathbf{x}) \leqslant g(\mathbf{x})$ for all $\mathbf{x}$, i.e., g_k minorizes g, and $g_k(\mathbf{x}_k) = g(\mathbf{x}_k)$.

Majorization-Minimization: Suppose we want to minimize $g(\mathbf{x})$. For some $\mathbf{x}_k$ define $g_k(\mathbf{x})$ s.t. $g_k(\mathbf{x}) \geqslant g(\mathbf{x})$ for all $\mathbf{x}$, i.e., g_k majorizes g, and $g_k(\mathbf{x}_k) = g(\mathbf{x}_k)$.

5.13.4.1 Convergence of the MM Algorithm

Property 5.6. If $\mathbf{x}_{k+1}$ maximizes $g_k(\mathbf{x})$, then $g(\mathbf{x}_{k+1}) \geqslant g(\mathbf{x}_k)$. If $g(\mathbf{x}) > g_k(\mathbf{x})$ for all $\mathbf{x} \neq \mathbf{x}_k$ and $\mathbf{x}_{k+1} \neq \mathbf{x}_k$, then $g(\mathbf{x}_{k+1}) > g(\mathbf{x}_k)$.

Proof. We have

$$\begin{aligned} g(\mathbf{x}_{k+1}) &\geqslant g_k(\mathbf{x}_{k+1}) \text{ (Minorization)} \\ &\geqslant g_k(\mathbf{x}_k) = g(\mathbf{x}_k) \end{aligned}$$

Note that if $g(\mathbf{x}) > g_k(\mathbf{x})$ for all $\mathbf{x} \neq \mathbf{x}_k$ and $\mathbf{x}_{k+1} \neq \mathbf{x}_k$, then we can replace the first line above by $g(\mathbf{x}_{k+1}) > g_k(\mathbf{x}_{k+1})$ and so $g(\mathbf{x}_{k+1}) > g(\mathbf{x}_k)$. □

5.13.4.2 Application: Bradley-Terry Model

Suppose a competition involving n teams, and the data consist of outcomes of games between pairs of teams. Based on these outcomes, we want to rank the teams from strongest to weakest. Implicitly, we assume an unbalanced schedule, i.e., the number of games that each pair of teams plays against each other is not necessarily equal.

Define unknown strength parameters $\tau_1, \cdots, \tau_n > 0$, then

$$\pi_{ij} = P(\text{Team } i \text{ beats team } j) = \frac{\tau_i}{\tau_i + \tau_j}$$

Note that

$$\pi_{ji} = \frac{\tau_j}{\tau_i + \tau_j} = 1 - \pi_{ij}$$

Note that if $\pi_{ij} = \dfrac{\tau_i}{\tau_i + \tau_j}$, then $\pi_{ij} = \dfrac{a\tau_i}{a\tau_i + a\tau_j}$ for any $a > 0$. Thus the strength parameters are not unambiguously defined, and we need to put some constraint on $\{\tau_i\}$ to make the model identifiable. For instance, we compare teams $2, \cdots, n$ to team 1, i.e., $\tau_1 = 1$; the average strength is set to 1. Also note that we can write the model as a logistic regression model

$$\text{logit}(\pi_{ij}) = \ln\left(\frac{\pi_{ij}}{\pi_{ji}}\right) = \ln(\tau_i) - \ln(\tau_j) := \alpha_i - \alpha_j$$

We assume independence between games, and the log-likelihood function is given by

$$\ln \mathcal{L}(\tau_1, \cdots, \tau_n) = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} \ln\left(\frac{\tau_i}{\tau_i + \tau_j}\right)$$

where w_{ij} represents the number of games won by team i over team j (with $w_{ii} = 0$), and we set $\tau_1 = 1$ and maximize the log-likelihood over $\tau_2, \cdots, \tau_n$. Differentiating the log-likelihood function w.r.t. τ_k for $k = 2, \cdots, n$, we get

$$\frac{\partial}{\partial \tau_k} \ln \mathcal{L}(\tau_1 = 1, \tau_2, \cdots, \tau_n) = \sum_{i=1}^{n} w_{ki} \left(\frac{1}{\tau_k} - \frac{1}{\tau_k + \tau_i} \right) - \sum_{i=1}^{n} \frac{w_{ik}}{\tau_k + \tau_i}$$

If we set these partial derivatives to 0, we get

$$\widehat{\tau}_k = \left(\sum_{i=1}^{n} w_{ki} \right) \left(\sum_{i=1}^{n} \frac{w_{ki} + w_{ik}}{\widehat{\tau}_k + \widehat{\tau}_i} \right)^{-1}$$

for $k = 2, \cdots, n$.

Note 1. Define $m_k = \sum_{i=1}^{n} (w_{ki} + w_{ik})$ to be the number of games played by team k.

Note 2. $\frac{1}{m_k} \sum_{i=1}^{n} w_{ki}$ is the proportion of games won by team k.

Note 3. $\left(\frac{1}{m_k} \sum_{i=1}^{n} \frac{w_{ki} + w_{ik}}{\widehat{\tau}_k + \widehat{\tau}_i} \right)^{-1}$ is a measure of the average quality of games played by team k.

Note 4. We rewrite $\widehat{\tau}_k$ as

$$\widehat{\tau}_k = \underbrace{\left(\frac{1}{m_k} \sum_{i=1}^{n} w_{ki} \right)}_{\text{Winning percentage}} \underbrace{\left(\frac{1}{m_k} \sum_{i=1}^{n} \frac{w_{ki} + w_{ik}}{\widehat{\tau}_k + \widehat{\tau}_i} \right)^{-1}}_{\text{Average quality}}$$

Given $\widehat{\tau}_1^{(l)} = 1, \widehat{\tau}_2^{(l)}, \cdots, \widehat{\tau}_n^{(l)}$, define

$$\mathcal{Q}_l(\tau) = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \left[\ln(\tau_i) - \frac{\tau_i + \tau_j}{\tau_i^{(l)} + \tau_j^{(l)}} - \ln(\tau_i^{(l)} + \tau_j^{(l)}) + 1 \right]$$

Note that

$$\mathcal{Q}_l(\widehat{\tau}_1^{(l)}, \cdots, \widehat{\tau}_n^{(l)}) = \ln \mathcal{L}(\widehat{\tau}_1^{(l)}, \cdots, \widehat{\tau}_n^{(l)})$$

$\mathcal{Q}_l(\tau_1, \cdots, \tau_n)$ is maximized over $\tau_2, \cdots, \tau_n$ at

$$\widehat{\tau}_k^{(l+1)} = \left(\sum_{i=1}^{n} w_{ki} \right) \left(\sum_{i=1}^{n} \frac{w_{ki} + w_{ik}}{\widehat{\tau}_k^{(l)} + \widehat{\tau}_i^{(l)}} \right)^{-1}$$

for $k = 2, \cdots, n$.

To prove that $\mathcal{Q}_l$ minorizes $\ln \mathcal{L}$, we use the inequality $-\ln(z) \geqslant 1 - z$ for $z > 0$. Thus

$$\begin{aligned} -\ln(\tau_i + \tau_j) &= -\ln \left(\frac{\tau_i + \tau_j}{\tau_i^{(l)} + \tau_j^{(l)}} \right) - \ln(\tau_i^{(l)} + \tau_j^{(l)}) \\ &\geqslant 1 - \frac{\tau_i + \tau_j}{\tau_i^{(l)} + \tau_j^{(l)}} - \ln(\tau_i^{(l)} + \tau_j^{(l)}) \end{aligned}$$

and therefore

$$\ln \mathcal{L}(\tau_1, \cdots, \tau_n) \geqslant \mathcal{Q}_l(\tau_1, \cdots, \tau_n)$$

5.14 Simulated Annealing

Suppose we want to maximize $g(\mathbf{x})$ over some set $\mathcal{C}$. We can do a random walk around $\mathcal{C}$ where the transition probabilities depend on a temperature parameter. We define the transition probabilities

$$\alpha_n(\mathbf{x}, \mathbf{y}) = \min\left\{\exp\left(\frac{g(\mathbf{y}) - g(\mathbf{x})}{T_n}\right), 1\right\}$$

where $\{T_n\}$ are temperature parameters with $T_n \to 0$ (slowly) as $n \to \infty$. Note that $\alpha_n(\mathbf{x}, \mathbf{y}) = 1$ if $g(\mathbf{y}) \geqslant g(\mathbf{x})$, and if $g(\mathbf{y}) < g(\mathbf{x})$, then $\alpha_n(\mathbf{x}, \mathbf{y}) \to 0$.

Example 5.20. Suppose we want to maximize $g(x) = \begin{cases} [\sin(20x) + \cos(50x)]^2, & 0 \leqslant x \leqslant 1 \\ 0, & \text{otherwise} \end{cases}$ For simulated annealing, we set $T_n = \dfrac{1}{1 + \ln(n)}$. Given x_n, define $y = x_n + z_n$ where $z_n \sim \text{Unif}(-T_n^{1/2}, T_n^{1/2})$.

Example 5.21 (Traveling Salesman Problem)**.** Suppose m cities with a distance $d(i, j)$ from city i to city j. We want to find shortest route/circuit between m cities returning to starting point, i.e., find a circuit $C = \{i_1, \cdots, i_m, i_{m+1} = i_1\}$ to minimize

$$d(C) = \sum_{k=1}^{m} d(i_k, i_{k+1})$$

Given C_n, we generate a proposal C by randomly exchanging 2 cities in C_n, then $C_{n+1} = C$ w.p.

$$\alpha_n(C_n, C) = \min\left\{\exp\left(\frac{d(C_n) - d(C)}{T_n}\right), 1\right\}$$

with $C_{n+1} = C_n$ w.p. $1 - \alpha_n(C_n, C)$. We take $T_n = \dfrac{k}{\ln(1 + n)}$ where k depends on the problem.