

# Statistical Computation

Derek Li

## Contents

<b>1</b>	<b>Review</b>	<b>3</b>
1.1	Complex Number . . . . .	3
1.2	Markov Chain . . . . .	3
1.3	Linear Algebra . . . . .	3
1.3.1	Application: Condition Numbers . . . . .	6
1.3.2	Gram-Schmidt Orthogonalization . . . . .	7
<b>2</b>	<b>Basics</b>	<b>8</b>
2.1	Floating Point . . . . .	8
2.1.1	Floating Point Representation . . . . .	8
2.1.2	Round-Off Error . . . . .	8
2.1.3	Machine Epsilon and Other Constants . . . . .	8
2.1.4	Overflow and Underflow Error . . . . .	8
2.1.5	Catastrophic Cancellation . . . . .	9
2.2	Sparse Matrices . . . . .	9
2.3	Application: Computation of Probability Distributions . . . . .	9
2.3.1	Brute Force Approach . . . . .	9
2.3.2	Probability Generating Function . . . . .	10
2.3.3	Discrete Fourier Transform (DFT) . . . . .	10
2.4	Application: Image Processing . . . . .	11
2.4.1	Transformation . . . . .	11
2.4.2	Hadamard Matrices and Walsh-Hadamard Transform . . . . .	12
2.5	Application: Denoising . . . . .	13
2.5.1	Assumption . . . . .	13
2.5.2	Thresholding . . . . .	13
2.5.3	The Fast W-H Transform . . . . .	13
2.5.4	R code for FWHT . . . . .	14
2.6	Fast Fourier Transform (FFT) . . . . .	14
2.6.1	FFT Derivation . . . . .	15
2.6.1.1	Case I: Even Number and Product of Small Prime Numbers . . . . .	16
2.6.1.2	Case II: Prime Number with Zero-Padding . . . . .	16
2.6.2	Analysis of DFT Approach . . . . .	16
<b>3</b>	<b>Generation of Random Variates</b>	<b>18</b>
3.1	Generation of Random Numbers . . . . .	18
3.2	Generation of Unif(0, 1) . . . . .	18
3.2.1	Linear Congruential RNG . . . . .	18

3.2.2	Combining Unif(0, 1) RNGs . . . . .	19
3.2.3	Shift Register Method . . . . .	19
3.3	Testing Unif(0, 1) RNGs . . . . .	20
3.4	RNGs in R . . . . .	20
3.5	Methods for Continuous Distribution . . . . .	20
3.5.1	Inverse Method . . . . .	20
3.5.2	Rejection Sampling . . . . .	21
3.6	Sampling from Mixture Densities . . . . .	24
3.6.1	Application: Walker's Alias Method . . . . .	24
3.7	Generation of Normal Random Variables . . . . .	25
3.7.1	Inverse Method . . . . .	25
3.7.2	Box-Muller Method . . . . .	25
3.7.3	Kinderman-Ramage Method . . . . .	25
3.7.4	Monty Python Method . . . . .	26
3.7.5	Sum of Uniforms . . . . .	27
3.8	Markov Chain Monte Carlo . . . . .	27
3.8.1	Construction of Reversible Markov Chain . . . . .	27
3.8.2	Metropolis-Hastings Algorithm . . . . .	27
3.8.2.1	Application to Bayesian Inference . . . . .	28
3.8.2.2	Random Walk (Metropolis) Sampler . . . . .	28
3.8.2.3	Independence (Hastings) Sampler . . . . .	28
3.8.3	Practical Issues of MCMC . . . . .	29
<b>4</b>	<b>Numerical Linear Algebra</b>	<b>30</b>
4.1	Solving Linear Equations . . . . .	30
4.2	Matrix Factorizations . . . . .	31
4.2.1	Cholesky Factorization . . . . .	31
4.2.1.1	Computation of $L$ . . . . .	31
4.2.1.2	Application: Generating Multivariate Normal Random Vectors . . . . .	32
4.3	Iterative Matrix Method . . . . .	32
4.3.1	Jacobi and Gauss-Seidel Algorithm . . . . .	32
4.3.1.1	Convergence of Gauss-Seidel and Jacobi Algorithm . . . . .	33
4.3.1.2	Comment . . . . .	33
4.3.1.3	Application: Quadratic Minimization . . . . .	33
4.4	Solving Least Squares Problems . . . . .	33
4.4.1	Additive Regression Models . . . . .	33
4.4.2	The Backfitting Algorithm . . . . .	35
4.4.3	Application: Measure of Dependence . . . . .	35
4.4.3.1	Computing $\lambda$ : The Alternating Conditional Expectation (ACE) Algorithm . . . . .	36
4.4.4	Least Squares . . . . .	36
4.4.4.1	Linear Regression Model . . . . .	37
4.4.4.2	Least Square Estimation with the QR Decomposition . . . . .	37
4.5	Randomized Numerical Linear Algebra . . . . .	38
4.5.1	Hutchinson's Method . . . . .	38
4.5.1.1	Application: Equivalent Degrees of Freedom for Loess . . . . .	39
4.5.1.2	Function of Symmetric Matrix and Trace . . . . .	40
4.5.1.3	Estimating $\det(A)$ . . . . .	40

# 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.,

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

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

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

**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  $a_{11}, \dots, a_{nn}$  are all non-zero.

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

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

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

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

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

1.  $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ .
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).

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

**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** ( $L_\infty$  Norm).

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

**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.  $\|AB\|_p \leq \|A\|_p \|B\|_p$ .
2.  $\|A^k\|_p \leq \|A\|_p^k$ .
3.  $\|I\|_p = \|AA^{-1}\|_p = 1 \Rightarrow \|A^{-1}\|_p \geq \frac{1}{\|A\|_p}$ .
4. For any vector  $\mathbf{x}$ ,  $\|A\mathbf{x}\|_p \leq \|A\|_p \|\mathbf{x}\|_p$ .
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** ( $L_\infty$  Norm for Matrix). Consider vectors  $\mathbf{x}$  whose elements are all  $\pm 1$ , i.e.,  $\|\mathbf{x}\|_\infty = 1$ .

We maximize  $\|A\mathbf{x}\|_\infty$  by taking  $\mathbf{x}$  so that  $\sum_{j=1}^n a_{ij}x_j$  is maximized and

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

**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

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

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

$$\max_{1 \leq k \leq n} |\lambda_k| \leq \|A\|_p, p \geq 1$$

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

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

□

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

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

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

If  $\geq$  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  $r_i = \sum_{j \neq i} |a_{ij}|$  and

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

which is a circle on the complex plane centered at  $a_{ii}$  with radius  $r_i$ . If  $\lambda$  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

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

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

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

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

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

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

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

### 1.3.1 Application: Condition Numbers

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

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

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

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

and thus

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

Then

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

and

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

where  $\kappa_p(A) = \|A\|_p \|A^{-1}\|_p \geq 1$  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  $\varepsilon$ .

### 1.3.2 Gram-Schmidt Orthogonalization

Given vectors  $\mathbf{v}_1, \dots, \mathbf{v}_r$ , suppose we want to find orthonormal vectors  $\mathbf{q}_1, \dots, \mathbf{q}_r$  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  $b_1, \dots, b_r$ .

The Gram-Schmidt algorithm is:

1. Define  $\mathbf{u}_1 = \mathbf{v}_1$ .
2. For  $i = 2, \dots, r$ , 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  $\mathbf{q}_i = \frac{\mathbf{u}_i}{\|\mathbf{u}_i\|}$  for  $i = 1, \dots, r$ .

## 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 ( $\pm 1$ ),  $F$  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  $x$ ,  $\text{fl}(x)$  is

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

**Note.**  $x$  and  $\text{fl}(x)$  need not be the same, since  $\text{fl}(x)$  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

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

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

#### 2.1.3 Machine Epsilon and Other Constants

For a given real number  $x$ , we have

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

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

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

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

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

### 2.1.5 Catastrophic Cancellation

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

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

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

$$\text{fl}(z_1) - \text{fl}(z_2) = \underbrace{z_1 - z_2}_y + \underbrace{z_1 u_1 - z_2 u_2}_{\text{error}}$$

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.** 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  $0, \dots, l$  with

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

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

### 2.3.1 Brute Force Approach

Start with  $n = 2$  and proceed inductively:

$$\begin{aligned} p_2(x) &:= P(X_1 + X_2 = x) = \sum_{y=0}^x P(X_1 = y, X_2 = x - y) \\ p_3(x) &:= P(X_1 + X_2 + X_3 = x) = \sum_{y=0}^x P(X_1 + X_2 = y, X_3 = x - y) \\ &\vdots \end{aligned}$$

$p_k(x)$  requires  $x + 1$  multiplications and to evaluate  $p_k(x)$  for  $x = 0, \dots, kl$ , 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.** If  $X$  is a discrete r.v. taking values  $0, 1, \dots$ , 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  $0, \dots, l$ , then  $P(X = x)$  can be recovered from evaluating  $\phi(t)$  at  $l + 1$  distinct (non-zero) points  $t_0, \dots, t_l$ .

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

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

Thus we can recover  $P(S = x)$  for  $x = 0, \dots, nl$  by evaluating  $[\phi(t)]^n$  at  $t_0, \dots, t_{nl}$ . 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  $t_0, \dots, t_{nl}$  are complex exponentials

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

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

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

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

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

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  $\mathbf{x}$  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 = dft.^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  $x(i, j), i = 1, \dots, m, j = 1, \dots, n$ , 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  $\phi_1, \dots, \phi_p$  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  $\hat{X} = H_1 X H_2$ , which has the same dimensions as  $X$ . Since for orthogonal matrix  $H$ ,  $H^{-1} = H^T$  and so  $X = H_1^T \hat{X} H_2^T$ . Assume the noisy image model  $X = S + E$ , if  $H_1$  and  $H_2$  are chosen appropriately,

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

Therefore,

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

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

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

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

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

$$X_c = H_1^T T(\hat{X}) H_2^T$$

**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.** A **Hadamard matrix** is an  $n \times n$  matrix whose elements are all  $\pm 1$  with orthogonal rows s.t.  $HH^T = nI$ .

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

**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  $1 \times 1$  Hadamard matrix  $H_1 = 1$ , and then define  $H_2, H_4, H_8, \dots$  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  $k = 2, 3, \dots$ .

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

**Definition 2.8.** 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  $H_{2^{k-1}}, H_{2^{k-2}}, \dots$ , 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.** Given an  $n \times n$  Hadamard matrix  $H$  and a vector  $\mathbf{x}$  of length  $n$ , we define its **Walsh-Hadamard transform** by  $\hat{\mathbf{x}} = H\mathbf{x}$ .

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

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

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

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

## 2.5 Application: Denoising

**Question:** Suppose we observe  $\mathbf{x} = (x_1, \dots, x_n)^T$  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  $\hat{\mathbf{s}} = H\mathbf{s}$  contains mostly 0s

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

### 2.5.2 Thresholding

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

$$\tilde{\mathbf{s}} = \frac{1}{n} H^T \hat{\mathbf{x}}_s$$

Define thresholds  $\lambda_1, \dots, \lambda_n \geq 0$ . The **hard thresholding** is to modify  $\hat{\mathbf{x}}$  as follows:

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

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

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

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

Typically we set  $\lambda_1 = 0$ , and use knowledge of the problem to decide  $\lambda_2, \dots, \lambda_n$ ; or take  $\lambda_2 = \dots = \lambda_n$  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  $\pm 1$  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  $8 \times 8$  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  $AAAx$  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  $\mathbf{x}$ .

---

```
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  $x_0, \dots, x_{n-1}$ , and define  $\hat{x}_0, \dots, \hat{x}_{n-1}$  by

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

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

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

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

*Proof.* For complex numbers  $z$ ,

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

Thus if  $z = \exp\left(\frac{2\pi\iota t}{n}\right)$  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,

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

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  $\mathbf{x} = (x_0, \dots, x_{n-1})^T$  and  $\hat{\mathbf{x}} = (\hat{x}_0, \dots, \hat{x}_{n-1})^T$ . Then

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

where  $F$  is an  $n \times n$  matrix whose  $j$ th row and  $k$ th column is

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

The elements of  $F^{-1}$  are

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

**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  $n_1, \dots, n_k : n = n_1 \times \dots \times n_k$ .

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

Assume  $n$  is even, then

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

Hence, the DFT of  $x_0, \dots, x_{n-1}$  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

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

where  $\Omega$  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  $n_1, \dots, n_k$  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:

$$x_0, \dots, x_{n-1}, \underbrace{0, \dots, 0}_m$$

with  $n + m = n_1 \times \dots \times n_k$  where  $n_1, \dots, n_k$  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  $m \geq nl = 1$  where  $m$  is a product of small prime numbers, and follow the steps:

1. Define  $\hat{p}_i(0), \dots, \hat{p}_i(m-1)$  to be the DFT of  $p_i(0), \dots, p_i(m-1)$  for  $i = 1, \dots, n$ .
2. Define

$$\hat{p}_s(k) = \prod_{i=1}^n \hat{p}_i(k), k = 0, \dots, m-1$$

3. Inverse DFT:  $P(S=0), \dots, P(S=m-1)$  is the inverse DFT of  $\hat{p}_s(0), \dots, \hat{p}_s(m-1)$ .

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

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

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

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

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

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

**Note.** We choose  $f$  satisfying precision and expediency:

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

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

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

We generate independent pairs and have

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

#### 3.2 Generation of $\text{Unif}(0, 1)$

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

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

##### 3.2.1 Linear Congruential RNG

Define  $V_1, V_2, \dots$  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.**  $V_1, V_2, \dots$  take values in the set  $\{0, \dots, m-1\}$ .

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

**Note 4.** We have  $V_{k+p} = V_k$  for some  $p \leq m$ , 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:

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

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  $Z_1, Z_2, \dots$  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  $0 \leq a < b \leq 1$ ,

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

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

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

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

### 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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\hat{\gamma} = \frac{1}{n} \sum_{i=1}^n X_i^2 \left( \frac{1}{[X_i]} - \frac{1}{X_i} \right)$$

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.  $f(x) \leq Mg(x)$  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  $T = \frac{f(Y)}{Mg(Y)}$ . If  $U \leq T$ , then set  $X = Y$ ; if  $U > T$ , then reject and repeat until acceptance. The algorithm works: Given independent  $Y \sim g$  and  $U \sim \text{Unif}(0,1)$ ,

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

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

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

Therefore,

$$\begin{aligned} P(\text{Accept})P\left(U \leq \frac{f(Y)}{Mg(Y)}\right) &= \int_{-\infty}^{\infty} \int_0^{f(y)M^{-1}g^{-1}(y)} g(y) du dy \\ &= \frac{1}{M} \int_{-\infty}^{\infty} f(y) dy \\ &= \frac{1}{M} \end{aligned}$$

and

$$\begin{aligned} P\left(Y \leq x, U \leq \frac{f(Y)}{Mg(Y)}\right) &= \int_{-\infty}^x \int_0^{f(y)M^{-1}g^{-1}(y)} g(y) du dy \\ &= \frac{1}{M} \int_{-\infty}^x f(y) dy \\ &= \frac{P(X \leq x)}{M} \end{aligned}$$

**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  $f(x) \leq Mg(x)$  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

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

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

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

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

To find  $M$ , we need to maximize  $\frac{f(x)}{g(x)}$  over  $x \geq 0$ , 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 but 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

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

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

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  $\frac{f_1(x)}{g(x)}$  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  $f_1(x), \dots, f_k(x)$  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  $\text{Unif}(0, 1)$  r.v.). Given  $J = j$ , sample  $X$  from  $f_j(x)$ . The algorithm works best if  $k$  is small or if  $\lambda_1 = \cdots = \lambda_k = \frac{1}{k} : J = [kU]$ .

#### 3.6.1 Application: Walker's Alias Method

Suppose we want to sample  $X$  from a discrete distribution

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

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

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

$f_1, \dots, f_k$  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  $a_j \in \{x_1, \dots, x_k\}$  is called an alias.

Given  $\tau_1, \dots, \tau_k$  and  $a_1, \dots, a_k$ , we sample  $X$  from  $f$  as follows: Generate  $U_1 \sim \text{Unif}(0, 1)$  and set  $J = [kU_1]$ ; generate  $U_2 \sim \text{Unif}(0, 1)$  and define  $X = x_J$  if  $U_2 \leq \tau_J$  and  $X = a_J$  if  $U_2 > \tau_J$ .

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

**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  $X \sim \text{Binom}(3, 0.4)$ :

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

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

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

## 3.7 Generation of Normal Random Variables

### 3.7.1 Inverse Method

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

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

$\Phi(x)$  is strictly increasing so we can define its inverse by  $\Phi(\Phi^{-1}(t)) = t$  for  $0 \leq t \leq 1$ . 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  $(R, \Theta)$  has joint density

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

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  $\Theta$  from  $g_2(\theta)$ , and

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

If  $U_1$  and  $U_2$  are independent  $\text{Unif}(0, 1)$ , 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  $\text{Unif}(-1, 1)$ , then  $V = \frac{2.216|U_1 + U_2|}{2}$  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

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

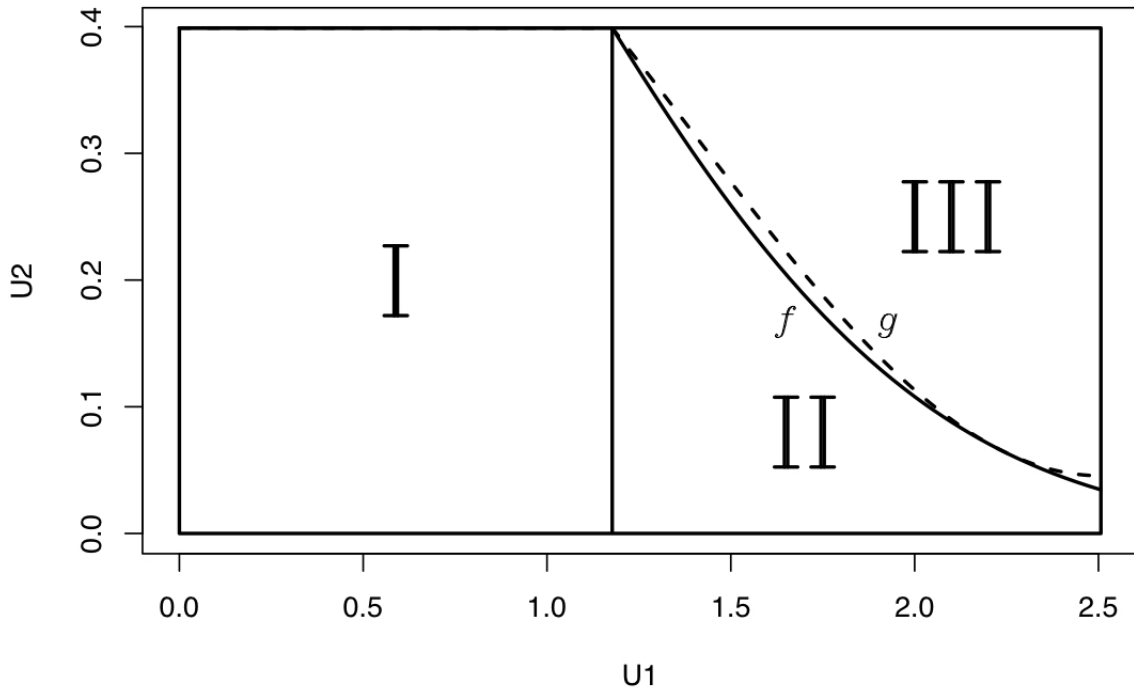
Define  $g(x)$  to be  $f(x)$  rotated and rescaled into  $\mathcal{B}$  for  $x$  s.t.

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

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  $\sqrt{\ln(4)} = a \leq x \leq b = \sqrt{2\pi}$ . 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  $\text{Unif}(0, 1)$  r.v.s.  $U_1, \dots, U_k$  and define

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

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

$$\alpha(\mathbf{x}, \mathbf{y}) = \frac{f(\mathbf{y})q(\mathbf{y}, \mathbf{x})}{f(\mathbf{x})q(\mathbf{x}, \mathbf{y})}$$

$$\alpha(\mathbf{y}, \mathbf{x}) = 1$$

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.

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

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  $U \sim \text{Unif}(0, 1)$  independent of  $\mathbf{Y}$ . If  $U \leq \alpha(\mathbf{X}_{i-1}, \mathbf{Y})$ , 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.

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

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.  $q(x, y) = \frac{1}{3}$  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  $\frac{f(\mathbf{y})}{f(\mathbf{x})}$  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

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

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

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  $\hat{\rho}(1), \dots$ .

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

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

Thus we have

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

and

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

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

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

**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

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

#### 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  $Y_1, \dots, Y_p$  independent  $\mathcal{N}(0, 1)$  r.v.s. and define  $\mathbf{Y} = (Y_1, \dots, Y_p)^T$ , 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  $\mathbf{x}_1, \mathbf{x}_2, \dots$  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  $i = 1, \dots, n$ , 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  $A = L + D + U$  :

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



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  $\mathbf{x}_k \rightarrow \mathbf{x}^*$  as  $k \rightarrow \infty$ , 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 (SOR) 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:

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

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  $\{(x_{i1}, \dots, x_{ip}, y) : i = 1, \dots, n\}$ , we assume that

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

where  $\{x_{ij} : i = 1, \dots, n; j = 1, \dots, p\}$  are predictor variables,  $f_1, \dots, f_p$  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

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

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

**Note 3.** Vector from:

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

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

$$\hat{\mathbf{g}} = \begin{pmatrix} \hat{g}(x_1) \\ \vdots \\ \hat{g}(x_n) \end{pmatrix} \text{ and } \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

We have

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

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  $\lambda_1, \dots, \lambda_n$  of a smoothing matrix  $S$  satisfy:

1.  $-1 < \lambda_1, \dots, \lambda_n \leq 1$ .
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  $\hat{\mathbf{g}}$  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

$$\hat{\mathbf{g}} = \underbrace{(I + \lambda A)^{-1}}_S \mathbf{y}$$

#### 4.4.2 The Backfitting Algorithm

Define smoothing matrices  $S_0, \dots, S_k$  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  $k = 1, \dots, p$ . Given current estimates  $\hat{\beta}_0, \hat{\mathbf{f}}_1, \dots, \hat{\mathbf{f}}_p$ , we update  $\hat{\mathbf{f}}_k$  as

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

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

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

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

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

and

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

We can write in matrix form:

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

#### 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  $\{(x_i, y_i) : i = 1, \dots, n\}$  from distribution and we can estimate  $\rho$  by

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

**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  $\psi$  and  $\phi$  to maximize the correlation between  $\psi(X)$  and  $\phi(Y)$ .

**Note.** The computation for  $\psi$  and  $\phi$  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  $\psi$  and  $\phi$  satisfying  $\mathbb{E}[\psi(X)] = \mathbb{E}[\phi(Y)] = 0$  and  $\mathbb{E}[\psi^2(X)] = \mathbb{E}[\phi^2(Y)] = 1$ , then  $\psi$  and  $\phi$  must satisfy the conditions

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

and

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

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 $\lambda$ : The Alternating Conditional Expectation (ACE) Algorithm

Given data  $(x_1, y_1), \dots, (x_n, y_n)$  we can estimate  $\psi$  and  $\phi$  by iterative smoothing:

1. Estimate  $\psi(x)$  by smoothing  $\{\hat{\phi}(y_i)\}$  as a function of  $\{x_i\}$ .
2. Estimate  $\phi(y)$  by smoothing  $\{\hat{\psi}(x_i)\}$  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  $\psi$  and  $\phi$  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 an 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  $\hat{\beta}$  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  $\hat{\beta}$  is the MLE of  $\beta$ . If we differentiate the objective function w.r.t.  $\beta$  and set the partial derivatives to 0, we get the **normal equations** for  $\hat{\beta}$

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

If  $X^T X$  is invertible then

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

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  $X^T X = LL^T$  where  $L$  is lower triangular, and we can solve  $LL^T \hat{\beta} = X^T \mathbf{y}$ .

**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

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

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.  $\alpha$ . The normal equations for  $\hat{\alpha}$  are

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

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

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

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  $\mathbf{v}_0, \dots, \mathbf{v}_p$  to get  $Q$ . Note that  $\mathbf{v}_0$  depends only on the first column of  $Q$ ,  $\mathbf{q}_0$  and for  $j = 1, \dots, p$ ,  $\mathbf{v}_j$  depends on  $\mathbf{q}_0, \dots, \mathbf{q}_j$ . Thus  $X = QR$  where  $R$  is upper triangular.

**Note 1.** The Gram-Schmidt algorithm is numerically unstable: Due to round-off error,  $\mathbf{q}_1, \dots, \mathbf{q}_r$  are not exactly orthogonal, i.e.,  $Q^T Q \neq I$ . 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  $\hat{\mathbf{y}} = X\hat{\beta}$  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

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

If  $\mathbf{v}_1, \dots, \mathbf{v}_n^T$  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  $\mathbf{V}_1, \dots, \mathbf{V}_m$  from some distribution, evaluate  $\mathbf{X}_1 = A\mathbf{V}_1, \dots, \mathbf{X}_m = A\mathbf{V}_m$ , and use the empirical distribution of  $\mathbf{X}_1, \dots, \mathbf{X}_m$  to infer properties of  $A$ . The random vectors  $\mathbf{V}_1, \dots, \mathbf{V}_m$  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  $\mathbf{V}_1, \dots, \mathbf{V}_m$  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  $\mathbf{V} = (V_1, \dots, V_n)^T$  where  $V_1, \dots, V_n$  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  $\hat{\mathbf{g}} = S\mathbf{y}$  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  $\Gamma$  is a diagonal matrix whose elements are the eigenvalues  $\lambda_1, \dots, \lambda_n$  of  $A$  and  $\Lambda$  is an orthogonal matrix whose columns are the eigenvectors of  $A$ .

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

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

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

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

If  $f$  is a polynomial

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

then approximating  $\text{trace}[f(A)]$  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

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

#### 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.

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

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  $\mathbf{V}_1, \dots, \mathbf{V}_m$  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  $D$ .