

Statistical Computation

Derek Li

Contents

1	Review	3
1.1	Complex Number	3
2	Basics	4
2.1	Floating Point	4
2.1.1	Floating Point Representation	4
2.1.2	Round-Off Error	4
2.1.3	Machine Epsilon and Other Constants	4
2.1.4	Overflow and Underflow Error	4
2.1.5	Catastrophic Cancellation	5
2.2	Sparse Matrices	5
2.3	Application: Computation of Probability Distributions	5
2.3.1	Brute Force Approach	5
2.3.2	Probability Generating Function	6
2.3.3	Discrete Fourier Transform (DFT)	6
2.4	Application: Image Processing	7
2.4.1	Transformation	7
2.4.2	Hadamard Matrices and Walsh-Hadamard Transform	8
2.5	Application: Denoising	9
2.5.1	Assumption	9
2.5.2	Thresholding	9
2.5.3	The Fast W-H Transform	9
2.5.4	R code for FWHT	10
2.6	Fast Fourier Transform (FFT)	10
2.6.1	FFT Derivation	11
2.6.1.1	Case I: Even Number and Product of Small Prime Numbers	12
2.6.1.2	Case II: Prime Number with Zero-Padding	12
2.6.2	Analysis of DFT Approach	12
3	Generation of Random Variates	14
3.1	Generation of Random Numbers	14
3.2	Generation of Unif(0, 1)	14
3.2.1	Linear Congruential RNG	14
3.2.2	Combining Unif(0, 1) RNGs	15
3.2.3	Shift Register Method	15
3.3	Testing Unif(0, 1) RNGs	16
3.4	RNGs in R	16

3.5	Methods for Continuous Distribution	16
3.5.1	Inverse Method	16

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

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 , $\text{fl}(x)$ 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 = 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 $x(i, j), i = 1, \dots, m, j = 1, \dots, n$, 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 ϕ_1, \dots, ϕ_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 ± 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 1×1 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 ± 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 $AAAx$ 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

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

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 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, \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 π). 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

$$a^{(m-1)/q} \not\equiv 1 \pmod{m}$$

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 $a^2 \pmod{5} \neq 1$ 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:

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

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

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

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 γ 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))
```
