Statistical Computation

Derek Li

Contents

1 Basics

1.1 Floating Point

1.1.1 Floating Point Representation

Definition 1.1. A ***floating point number*** is represented by three components: (S, F, E) where S is the sign of the number (± 1), F is a fraction (lying between 0 and 1), E is an exponent. S, F, E are all represented as binary digits (bits). The ***floating point representation*** of x, $\mathrm{fl}(x)$ is

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

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

1.1.2 Round-Off Error

Mathematical operations introduce further approximation errors

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

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

1.1.3 Machine Epsilon and Other Constants

For a given real number x, we have

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

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

Other machine dependent constants include:

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

1.1.4 Overflow and Underflow Error

Definition 1.2. If the result of a floating point operation exceeds $x_{\max}$, then the value returned is `Inf`.

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

Definition 1.3. If the result of a floating point operation is undefined then `NaN` is returned.

Definition 1.4. 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 1.1. For x close to 0, $\dfrac{\exp(x)-1}{x} \approx 1$. Naive computation of $\dfrac{\exp(x)-1}{x}$ is problematic for x close to 0 due to possible round-off and underflow errors:

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

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

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

1.1.5 Catastrophic Cancellation

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

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

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

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

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***.

1.2 Sparse Matrices

Definition 1.5. 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.

1.3 Application: Computation of Probability Distributions

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

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

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

1.3.1 Brute Force Approach

Start with $n = 2$ and proceed inductively:

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

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

$$\vdots$$

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

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

Thus the total number of multiplications is

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

1.3.2 Probability Generating Function

Definition 1.6. If X is a discrete r.v. taking values $0, 1, \cdots$, then its ***probability generating function*** is

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

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

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

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

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

1.3.3 Discrete Fourier Transform (DFT)

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

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

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

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

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

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

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

1.3.4 R Implementation with DFT

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

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

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

1.4 Application: Image Processing

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

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

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

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

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

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

1.4.1 Transformation

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

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

Therefore,

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

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

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

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

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

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

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

1.4.2 Hadamard Matrices and Walsh-Hadamard Transform

Definition 1.7. A ***Hadamard matrix*** is an $n \times n$ matrix whose elements are all ± 1 with orthogonal rows s.t. $HH^T = nI$.

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

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

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

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$
$$H_{2^k} = \begin{pmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{pmatrix}$$

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

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

Definition 1.8. 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 1.1. Assume below that any matrix sums, products or inverses are well-defined.

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

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

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

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

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

Definition 1.9. Given an $n \times n$ Hadamard matrix H and a vector $\mathbf{x}$ of length n, we define its ***Walsh-Hadamard transform*** by $\widehat{\mathbf{x}} = H\mathbf{x}$.

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

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

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

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

1.5 Application: Denoising

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

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

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

1.5.1 Assumption

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

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

1.5.2 Thresholding

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

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

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

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

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

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

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

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

1.5.3 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 1.2 ($n = 2^3 = 8$)**.** The 8×8 Hadamard matrix is

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

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

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

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

1.5.4 R code for FWHT

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

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

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