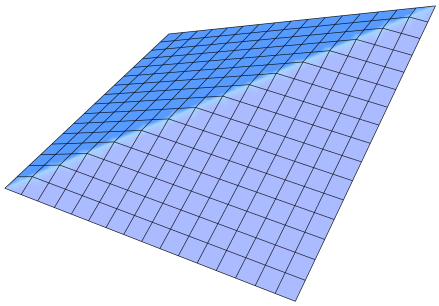

STA 3431H: Monte Carlo Methods

Prof. Jeffrey S. Rosenthal • Fall 2019 • University of Toronto
M 10:10AM–12:00PM in OISE 5150

Transcribed, LaTeXed, and edited by Rob Zimmerman

Note: These are personal lecture notes, not official course materials. They may contain errors (which by default you should attribute to me, Rob Zimmerman, rather than the course instructor). The permanent home of these — and all of my other lecture notes — is https://rob-zimmerman.github.io/coursenotes.

Contents

Lecture 1 — Monday, September 09, 2019

1. Foundations of Monte Carlo

Classical Monte Carlo Estimation

Monte Carlo methods use pseudorandomness on a computer to simulate random variables and thereby estimate quantities of interest. The basic method is intentionally simple: draw many copies of a random variable, compute the corresponding quantity for each copy, and average. Its great advantage is that essentially the same procedure works in high dimensions and in settings where direct numerical integration is impractical.

Example 1.1 Suppose that $Z \sim \mathcal{N}(0,1)$ and that the objective is to estimate

$$m := \mathbb{E}\big[Z^4 \cos(Z)\big].$$

Generate independent copies $Z_1, \ldots, Z_n \sim \mathcal{N}(0,1)$, set

$$X_i := Z_i^4 \cos(Z_i), \quad i = 1, \ldots, n,$$

and use the Monte Carlo estimator

$$\overline{X}_n := \frac{1}{n} \sum_{i=1}^{n} X_i.$$

Since $\mathbb{E}[\overline{X}_n] = m$, the estimator is unbiased. A small run can nevertheless be unstable. As n increases to one million, repeated estimates become consistently close to -1.213.

The following R code performs the simulation, reports its estimated standard error and confidence interval, and records a running estimate for the plot in Figure 1.

```r
set.seed(3431)

n <- 1000000
z <- rnorm(n)
x <- z^4 * cos(z)

estimate <- mean(x)
mc_se <- sd(x) / sqrt(n)
confidence_interval <- estimate + c(-1, 1) * qnorm(0.975) * mc_se

c(
  estimate = estimate,
  standard_error = mc_se,
  lower = confidence_interval[1],
  upper = confidence_interval[2]
)
```

Let

$$s_n^2 := \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \overline{X}_n)^2$$

be the sample variance. The estimated standard error of the sample mean is

$$\operatorname{se}(\overline{X}_n) = n^{-1/2} s_n = \sqrt{\frac{1}{n(n-1)} \sum_{i=1}^{n} (X_i - \overline{X}_n)^2}. \tag{1.1}$$

The central limit theorem gives, for large n,

$$\overline{X}_n \approx \mathcal{N}\left(m, \frac{\operatorname{Var}(X_1)}{n}\right) \approx \mathcal{N}\left(m, \operatorname{se}(\overline{X}_n)^2\right).$$

Consequently,

$$\mathbb{P}\left(-1.96 < \frac{m - \overline{X}_n}{\operatorname{se}(\overline{X}_n)} < 1.96\right) \approx 0.95,$$

so an approximate 95% confidence interval for m is

$$\left(\overline{X}_n - 1.96 \operatorname{se}(\overline{X}_n), \overline{X}_n + 1.96 \operatorname{se}(\overline{X}_n)\right). \tag{1.2}$$

A Student t-quantile is sometimes used when the variance is estimated, although it is exact only for normal observations. For this nonnormal example, both the normal and Student approximations rely on a large simulation, where their numerical difference is negligible.

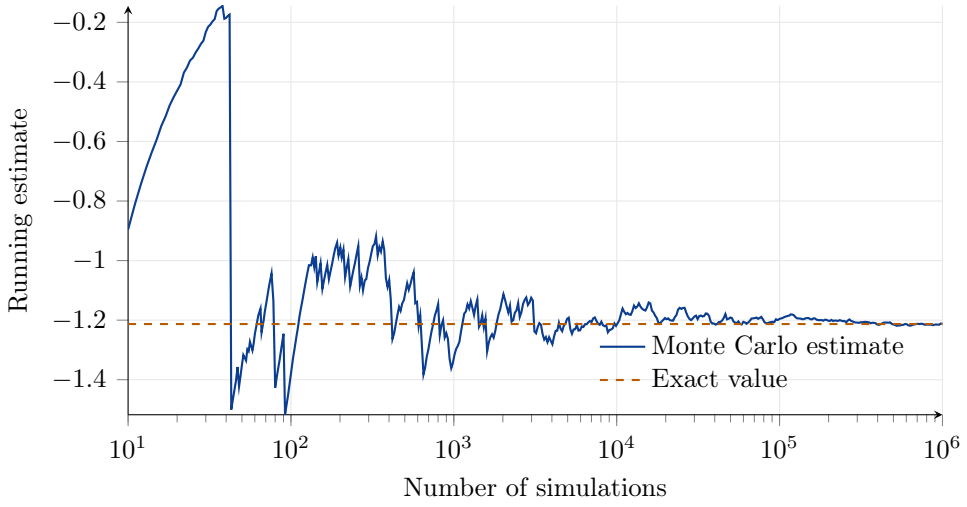

FIGURE 1

The stabilization in Figure 1 is the practical meaning of the law of large numbers. The exact expectation can also be written as

$$\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} z^4 \cos(z) e^{-z^2/2} \, \mathrm{d}z.$$

By viewing it as the real part of $\mathbb{E}[Z^4 e^{iZ}]$, we can show that its exact value is $-2/\sqrt{e} = -1.213061\ldots$. Such an analytic calculation is special to this example; the simulation method continues to apply when the dimension is large or the target distribution is complicated.

Remark 1.2 Monte Carlo integration is usually inferior to a reliable low dimensional numerical integration routine. Its role is to solve problems for which direct quadrature, exact analysis, or simpler sampling methods are no longer practical. A recurring principle throughout the course is therefore: repeat the experiment, and quantify the resulting variability.

Buffon's Needle and Pseudorandom Numbers

One of the earliest Monte Carlo calculations is Buffon's needle experiment. Parallel lines are separated by distance w, and a needle of length $\ell \le w$ is dropped at random. Let θ be the counterclockwise angle from the line direction and let H be the distance from the upper endpoint of the needle to the nearest line below it. The model takes

$$H \sim \text{Unif}[0, w] \qquad \text{and} \qquad \theta \sim \text{Unif}[0, \pi],$$

independently.

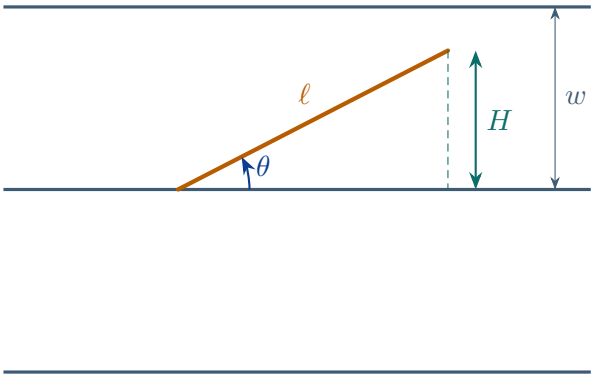

FIGURE 2

As Figure 2 shows, the needle touches a line if and only if $H < \ell \sin(\theta)$. Hence

$$\mathbb{P}(\text{the needle touches a line}) = \frac{1}{\pi}\int_0^{\pi} \frac{1}{w}\int_0^{w} \mathbb{1}_{\{h<\ell\sin(\theta)\}}\, dh\, d\theta = \frac{1}{\pi w}\int_0^{\pi} \ell\sin(\theta)\, d\theta = \frac{2\ell}{\pi w}.$$

If m of n independent drops touch a line, then the law of large numbers gives

$$\frac{m}{n} \approx \frac{2\ell}{\pi w},$$

and therefore

$$\pi \approx \frac{2n\ell}{mw}.$$

When $\ell = w$, this reduces to $\pi \approx 2n/m$. In 1864, Captain O. C. Fox used $\ell = 3$, $w = 4$, $n = 530$, and $m = 253$, obtaining $\pi \approx 3.1423$.

Modern simulations need a computational source of numbers that behave like independent Unif$[0, 1]$ draws. A simple construction is the linear congruential generator.

Definition 1.3 Choose positive integers m, a, b and an initial seed x_0. The **linear congruential generator** is defined recursively by

$$x_n = (ax_{n-1} + b) \bmod m \qquad \text{and} \qquad U_n = \frac{x_n}{m}.$$

Thus $0 \le x_n \le m - 1$, and the sequence (U_n) is intended to resemble independent Unif$[0, 1]$ observations.

The modulus m should be large, the multiplier a should avoid an obvious relationship between successive observations, the increment b should avoid short cycles, and the period before repetition should be long. Statistical test suites such as the diehard tests can reveal serial dependence and other defects.

Theorem 1.4 The linear congruential generator in Definition 1.3 has full period m if and only if the following two conditions hold:

1. $\gcd(b, m) = 1$.
2. Every prime divisor of m, as well as 4 when $4 \mid m$, divides $a - 1$.

For example, if $m = 2^{32}$, b is odd, and $a - 1$ is divisible by 4, then Theorem 1.4 gives the full period $2^{32} \approx 4.3 \times 10^9$. One commonly used parameter set is

$$m = 2^{32}, \qquad a = 69{,}069, \qquad \text{and} \qquad b = 23{,}606{,}797.$$

Another widely used textbook parameter set is $m = 2^{31}$, $a = 1{,}103{,}515{,}245$, and $b = 12{,}345$. Actual C-library generators using these constants may apply additional state or output transformations rather than returning the raw recurrence directly.

The historical RANDU generator used $m = 2^{31}$, $a = 65{,}539 = 2^{16} + 3$, and $b = 0$. Its successive

values satisfy

$$\begin{aligned} x_{n+2} &= a^2 x_n = (2^{16}+3)^2 x_n \\ &= (2^{32} + 6 \cdot 2^{16} + 9) x_n \equiv 6(2^{16}+3)x_n - 9x_n \pmod{2^{31}} \\ &= 6x_{n+1} - 9x_n \pmod{2^{31}}. \end{aligned}$$

After division by $m = 2^{31}$, every normalized triple satisfies

$$U_{n+2} - 6U_{n+1} + 9U_n \in \mathbb{Z}. \tag{1.3}$$

Since the left hand side lies strictly between -6 and 10, the triples can occupy at most fifteen parallel planes in the unit cube. This defect is difficult to notice in a plot of successive pairs and striking in three dimensions. The following R implementation produces both views in Figure 3.

```
randu <- function(count, seed = 1) {
  modulus <- 2^31
  multiplier <- 65539
  state <- numeric(count)
  state[1] <- seed

  for (index in 2:count) {
    state[index] <- (multiplier * state[index - 1]) %% modulus
  }

  state / modulus
}

u <- randu(2502)

pairs <- data.frame(
  first = u[1:2000],
  second = u[2:2001]
)

triples <- data.frame(
  first = u[1:2500],
  second = u[2:2501],
  third = u[3:2502]
)

# RANDU satisfies U_(n+2) - 6 U_(n+1) + 9 U_n = integer.
triples$plane <- round(triples$third - 6 * triples$second + 9 * triples$first)
```

```
sort(unique(triples$plane))
```

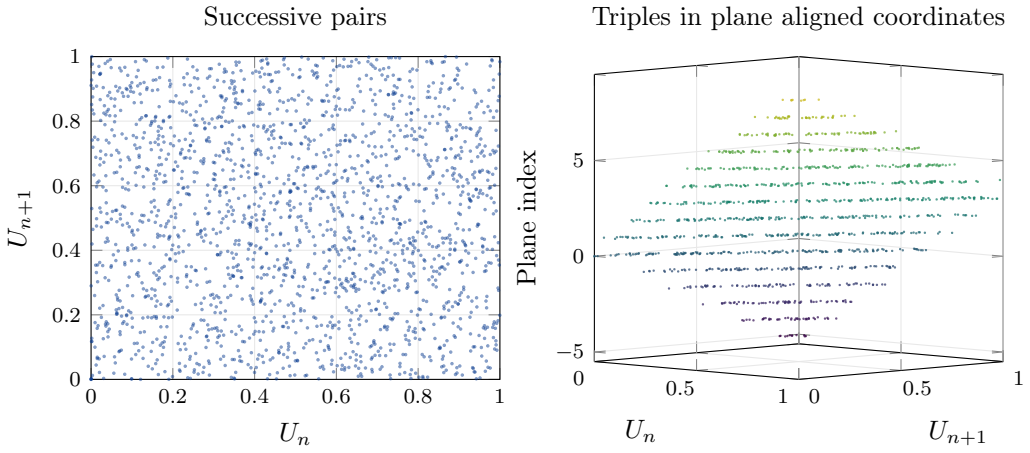

FIGURE 3

The pair plot in Figure 3 appears reasonably diffuse. In the triple plot, the vertical coordinate is $9U_n - 6U_{n+1} + U_{n+2}$, so the coordinate system is aligned with the normal vector to the planes. The fifteen parallel sheets imposed by (1.3) become unmistakable. A simulation that uses consecutive triples can therefore inherit a strong geometric bias. Earlier spreadsheet generators had additional problems, including periods below 10^6 and, in one floating point implementation, occasional negative outputs.

Pseudorandom numbers are deterministic and will fail sufficiently demanding tests. Some systems supplement them with physical randomness from processor temperature, kernel entropy, quantum effects, or atmospheric noise. Standard pseudorandom generators are nevertheless adequate for most statistical computations.

More elaborate constructions include multiply with carry generators. One such recurrence is

$$x_n = (ax_{n-r} + b_{n-1}) \bmod m \qquad \text{and} \qquad b_n = \left\lfloor \frac{ax_{n-r} + b_{n-1}}{m} \right\rfloor. \tag{1.4}$$

The KISS construction combines a sequence such as (1.4) with two shift register generators:

$$\begin{aligned} y_n &= (x_n + J_n + K_n) \bmod 2^{32}, \\ J_{n+1} &= (\mathbf{I} + \mathbf{L}^{15})(\mathbf{I} + \mathbf{R}^{17}) J_n \bmod 2^{32}, \\ K_{n+1} &= (\mathbf{I} + \mathbf{L}^{13})(\mathbf{I} + \mathbf{R}^{18}) K_n \bmod 2^{31}, \end{aligned}$$

where **L** and **R** denote bit shift operators. The Mersenne Twister instead uses a recurrence of the form

$$x_{n+k} = x_{n+s} \oplus \left(x_n^{\text{upper}} \mid x_{n+1}^{\text{lower}} \right) \mathbf{A},$$

where $1 \leq s < k$, the quantity $2^{kw-r} - 1$ is a Mersenne prime, $\mathbf{A}$ is a $w \times w$ binary matrix with a $(w-1) \times (w-1)$ identity block in its upper right corner, and all matrix arithmetic is performed bitwise modulo 2. R uses the Mersenne Twister by default; its internal state and available alternatives can be inspected through `?.Random.seed`.

Computer arithmetic introduces a separate approximation. In R,

```
2 + 1 - 2
2^10 + 1 - 2^10
2^100 + 1 - 2^100
```

returns 1, 1, and 0, respectively. The final computation loses the added unit because a floating point representation has finite precision. Similar behaviour occurs with floating point types in C, Java, and Python, whereas arbitrary precision integer arithmetic can retain the unit. Numerical output must therefore be treated as an approximation even before Monte Carlo error is considered.

Transformations and the Inverse Distribution Method

Suppose that $U_1, U_2, \ldots$ are independent Unif$[0,1]$ random variables. Transformations of these variables produce many familiar distributions. For $L < R$,

$$X = (R - L)U_1 + L$$

has the Unif$[L, R]$ distribution. Similarly,

$$X = \begin{cases} 1, & U_1 \leq p, \\ 0, & U_1 > p, \end{cases}$$

is Bernoulli with success probability p. A binomial random variable can be generated either by summing n independent Bernoulli variables or by applying the discrete inverse distribution construction below.

Suppose $x_1 < x_2 < \cdots$, $p_i \geq 0$, and $\sum_{i=1}^{\infty} p_i = 1$. Then, for $0 < U_1 < 1$,

$$Y = \min\left\{x_j : U_1 \leq \sum_{k=1}^{j} p_k\right\}$$

satisfies $\mathbb{P}(Y = x_i) = p_i$. For a binomial distribution this becomes

$$Y = \min\left\{0 \leq j \leq n : U_1 \leq \sum_{k=0}^{j} \binom{n}{k} p^k (1-p)^{n-k}\right\}.$$

For continuous examples, $Z = -\log(U_1)$ has the Exp(1) distribution because, for $x > 0$,

$$\mathbb{P}(Z > x) = \mathbb{P}(U_1 < e^{-x}) = e^{-x}.$$

Consequently, $-\log(U_1)/\lambda$ has density $\lambda e^{-\lambda x}$ for $x > 0$. If $r > 0$, then $X = U_1^{1/r}$ has distribution function x^r and density rx^{r-1} on $(0, 1)$.

The famous Box–Muller transform below was introduced by Box and Muller, "A note on the generation of random normal deviates" (1958). A rigorous treatment of the inverse distribution method appears in Rosenthal, *A First Look at Rigorous Probability Theory*, second edition.

Proposition 1.5 Let U_1, U_2 be independent Unif$[0, 1]$ random variables and define

$$X = \sqrt{2\log(1/U_1)}\cos(2\pi U_2) \qquad \text{and} \qquad Y = \sqrt{2\log(1/U_1)}\sin(2\pi U_2).$$

Then X and Y are independent $\mathcal{N}(0, 1)$ random variables.

Proof. Write $(x, y) = h(u_1, u_2)$ for the displayed transformation. The joint density of (U_1, U_2) equals 1 on $(0, 1)^2$. The Jacobian determinant is

$$J(u_1, u_2) = \det\begin{pmatrix} -\dfrac{\cos(2\pi u_2)}{u_1\sqrt{2\log(1/u_1)}} & -2\pi\sin(2\pi u_2)\sqrt{2\log(1/u_1)} \\ -\dfrac{\sin(2\pi u_2)}{u_1\sqrt{2\log(1/u_1)}} & 2\pi\cos(2\pi u_2)\sqrt{2\log(1/u_1)} \end{pmatrix} = -\frac{2\pi}{u_1}.$$

Since $u_1 = e^{-(x^2+y^2)/2}$, the change of variables theorem gives

$$f_{X,Y}(x, y) = \frac{1}{|J(h^{-1}(x, y))|} = \frac{1}{2\pi}e^{-(x^2+y^2)/2} = \left(\frac{1}{\sqrt{2\pi}}e^{-x^2/2}\right)\left(\frac{1}{\sqrt{2\pi}}e^{-y^2/2}\right).$$

The joint density factors into two standard normal densities, proving both normality and independence. □

The most general one dimensional transformation uses the inverse distribution function.

Theorem 1.6 Let F be a distribution function and define its generalized inverse by

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

If $U \sim \text{Unif}[0,1]$, then $X = F^{-1}(U)$ has distribution function F.

Proof. The values $U = 0$ and $U = 1$ have probability zero, so $F^{-1}(U)$ may be defined arbitrarily there. The generalized inverse satisfies $F^{-1}(U) \leq x$ if and only if $U \leq F(x)$. Therefore,

$$\mathbb{P}(X \leq x) = \mathbb{P}(U \leq F(x)) = F(x).$$

□

The construction in Theorem 1.6 is very general, but computing F^{-1} may itself be difficult. For standard one dimensional distributions, however, transformations and well tested library routines make independent simulation comparatively easy.

Lecture 2 — Monday, September 16, 2019

2. Monte Carlo Integration and Direct Sampling

Integration and Importance Sampling

An integral can be evaluated by rewriting it as an expectation. For example, if X and Y are independent $\text{Unif}[0,1]$ random variables, then

$$\int_0^1 \int_0^1 g(x,y)\,\mathrm{d}x\,\mathrm{d}y = \mathbb{E}[g(X,Y)].$$

Independent copies (X_i, Y_i) give the estimator

$$\widehat{I}_M = \frac{1}{M}\sum_{i=1}^{M} g(X_i, Y_i).$$

For $g(x,y) = \cos\bigl(\sqrt{xy}\bigr)$, this produces approximately 0.88 ± 0.003, while direct numerical integration gives 0.879544.

The integration region can be absorbed into the sampling distribution. If $X \sim \text{Unif}[0,5]$ and $Y \sim \text{Unif}[0,4]$ are independent, then

$$\int_0^5 \int_0^4 g(x,y)\,\mathrm{d}y\,\mathrm{d}x = \mathbb{E}[20g(X,Y)].$$

For the same g, one million simulations give approximately -4.11 ± 0.01, compared with the numerical value -4.11692.

Unbounded regions require a nonuniform sampling density. With

$$h(x,y) = e^{-y^2}\cos(\sqrt{xy}),$$

write

$$\int_0^1 \int_0^\infty h(x,y)\,\mathrm{d}y\,\mathrm{d}x = \int_0^1 \int_0^\infty e^{y} h(x,y) e^{-y}\,\mathrm{d}y\,\mathrm{d}x = \mathbb{E}[e^{Y} h(X,Y)],$$

where $X \sim \text{Unif}[0,1]$ and $Y \sim \text{Exp}(1)$ are independent. One million observations give approximately 0.767 ± 0.0004, close to the numerical value 0.767211.

More generally, if $Y \sim \text{Exp}(\lambda)$, then

$$\int_0^1 \int_0^\infty h(x,y)\,\mathrm{d}y\,\mathrm{d}x = \mathbb{E}\left[\frac{e^{\lambda Y}}{\lambda} h(X,Y)\right].$$

The choice $\lambda = 5$ yields a standard error near 0.0016, and $\lambda = 1/5$ gives about 0.0015. Both are worse than $\lambda = 1$. The choice $\lambda = 1.5$ has standard error near 0.00025, and a finer numerical search places the minimum near 1.8. The sampling law is therefore part of the design of the estimator.

The following R implementation recreates the three integration estimates and evaluates the importance sampling standard error over a grid of exponential rates. The results are displayed in Figure 4.

```r
running_summary <- function(values, point_count = 450) {
  sample_size <- length(values)
  index <- unique(round(exp(seq(log(10), log(sample_size), length.out = point_count))))
  data.frame(
    n = index,
    estimate = cumsum(values)[index] / index
  )
}

cosine_average <- function(y) {
  answer <- 2 * (sqrt(y) * sin(sqrt(y)) + cos(sqrt(y)) - 1) / y
  answer[y < 1e-7] <- 1
  answer
}

cosine_square_average <- function(y) {
  answer <- 1 / 2 + sin(2 * sqrt(y)) / (2 * sqrt(y)) +
    (cos(2 * sqrt(y)) - 1) / (4 * y)
  answer[y < 1e-7] <- 1
  answer
}

set.seed(3431)
sample_size <- 1000000

x <- runif(sample_size)
y <- runif(sample_size)
unit_running <- running_summary(cos(sqrt(x * y)))

x <- runif(sample_size, 0, 5)
y <- runif(sample_size, 0, 4)
rectangle_running <- running_summary(20 * cos(sqrt(x * y)))

x <- runif(sample_size)
y <- rexp(sample_size)
unbounded_running <- running_summary(
  exp(y - y^2) * cos(sqrt(x * y))
)

unit_truth <- 0.879544
rectangle_truth <- -4.11692
unbounded_truth <- integrate(
  function(y) exp(-y^2) * cosine_average(y),
  0,
```

```
  20,
  rel.tol = 1e-10
)$value

lambda_grid <- seq(0.15, 5.5, length.out = 90)
second_moment <- function(lambda) {
  integrate(
    function(y) {
      exp(-2 * y^2 + lambda * y) * cosine_square_average(y) / lambda
    },
    0,
    20,
    rel.tol = 1e-8
  )$value
}

importance_standard_errors <- data.frame(
  lambda = lambda_grid,
  standard_error = sqrt(
    vapply(lambda_grid, second_moment, numeric(1)) - unbounded_truth^2
  ) / sqrt(sample_size)
)

importance_standard_errors[
  which.min(importance_standard_errors$standard_error),
]
```

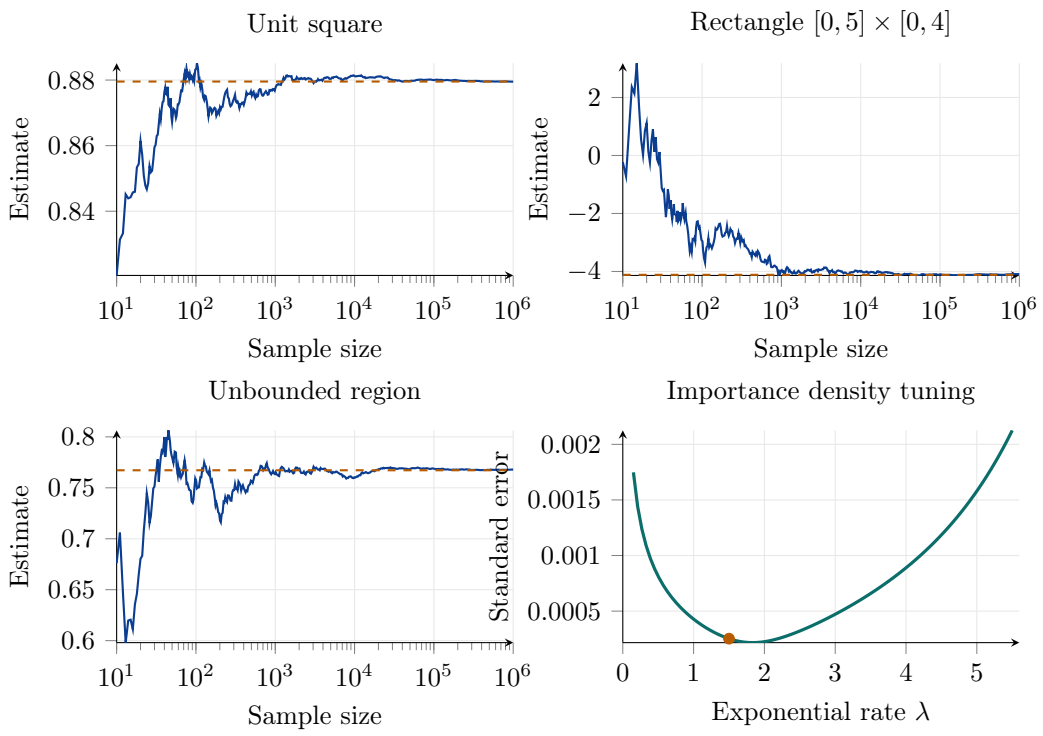

FIGURE 4

In the first three panels of Figure 4, the Monte Carlo estimates settle around the dashed numerical integration values. The last panel makes the tuning problem visible: proposals that are either too diffuse or too concentrated increase the variance, while the rate $\lambda = 1.5$ lies near the bottom of the curve.

Definition 2.1 Suppose that s is integrable and

$$I = \int_{\mathcal{X}} s(x)\,\mathrm{d}x,$$

and let f be an easily sampled density such that $f(x) > 0$ whenever $s(x) \neq 0$. Then

$$I = \int_{\mathcal{X}} \frac{s(x)}{f(x)} f(x)\,\mathrm{d}x = \mathbb{E}_f\left[\frac{s(X)}{f(X)}\right].$$

Estimating this expectation using independent $X_i \sim f$ is called **importance sampling**.

The importance density should make $s(x)/f(x)$ as nearly constant as possible. In the extreme

case,

$$\int_0^\infty e^{-3x}\,\mathrm{d}x = \mathbb{E}\left[\frac{1}{3}\right] = \frac{1}{3},$$

when $X \sim \mathrm{Exp}(3)$. The estimator then has zero variance, so simulation is unnecessary.

Importance sampling also handles an unnormalised target. Suppose that $g \geq 0$, $0 < \int g(x)\,\mathrm{d}x < \infty$, and

$$\pi(x) = c\,g(x) \qquad \text{and} \qquad c = \left(\int g(x)\,\mathrm{d}x\right)^{-1},$$

where g is known but c is not. For h satisfying $\int |h(x)|g(x)\,\mathrm{d}x < \infty$,

$$I := \mathbb{E}_\pi[h(X)] = \frac{\int h(x)g(x)\,\mathrm{d}x}{\int g(x)\,\mathrm{d}x}.$$

If $X_1, \ldots, X_M$ are independent with density f, where $f(x) > 0$ whenever $g(x) > 0$, then

$$\widehat{I}_M = \frac{\sum_{i=1}^M h(X_i)g(X_i)/f(X_i)}{\sum_{i=1}^M g(X_i)/f(X_i)}. \tag{2.1}$$

This ratio is not unbiased, even though its numerator and denominator separately estimate their corresponding integrals without bias. It is nevertheless consistent as $M \to \infty$. Using the same sample in both sums is computationally simpler and allows their covariance to contribute to the ratio's variance; whether that covariance helps depends on the particular integrands.

Example 2.2 Let

$$g(y) = y^3 \sin\left(y^4\right)\cos\left(y^5\right)\mathbb{1}_{\{0<y<1\}} \qquad \text{and} \qquad h(y) = y^2.$$

The objective is to estimate $\mathbb{E}_\pi[h(Y)]$ for $\pi = cg$. If $f(y) = 6y^5\mathbb{1}_{\{0<y<1\}}$, then $U^{1/6} \sim f$, and (2.1) becomes

$$\widehat{I}_M = \frac{\sum_{i=1}^M \sin\left(X_i^4\right)\cos\left(X_i^5\right)}{\sum_{i=1}^M \sin\left(X_i^4\right)\cos\left(X_i^5\right)/X_i^2}.$$

Alternatively, if $f(y) = 4y^3\mathbb{1}_{\{0<y<1\}}$, then $U^{1/4} \sim f$, and

$$\widehat{I}_M = \frac{\sum_{i=1}^M X_i^2 \sin\left(X_i^4\right)\cos\left(X_i^5\right)}{\sum_{i=1}^M \sin\left(X_i^4\right)\cos\left(X_i^5\right)}.$$

Both estimates are close to 0.766.

The following implementation records both running self normalized estimates and a random walk Metropolis estimate of the same expectation. Their common limit is shown in Figure 5.

```
running_ratio <- function(numerator, denominator, point_count = 450) {
  sample_size <- length(numerator)
  index <- unique(round(exp(seq(log(10), log(sample_size), length.out =
↪  point_count))))
  data.frame(
    n = index,
    estimate = cumsum(numerator)[index] / cumsum(denominator)[index]
  )
}

running_mean <- function(values, point_count = 450) {
  sample_size <- length(values)
  index <- unique(round(exp(seq(log(10), log(sample_size), length.out =
↪  point_count))))
  data.frame(
    n = index,
    estimate = cumsum(values)[index] / index
  )
}

set.seed(3431)
sample_size <- 200000

x <- runif(sample_size)^(1 / 6)
oscillation <- sin(x^4) * cos(x^5)
proposal_six <- running_ratio(
  oscillation,
  oscillation / x^2
)

x <- runif(sample_size)^(1 / 4)
oscillation <- sin(x^4) * cos(x^5)
proposal_four <- running_ratio(
  x^2 * oscillation,
  oscillation
)

log_target <- function(x) {
```

```
  if (x <= 0 || x >= 1) {
    return(-Inf)
  }
  3 * log(x) + log(sin(x^4)) + log(cos(x^5))
}

warmup <- 1000
state <- 0.5
metropolis_values <- numeric(sample_size)
for (iteration in seq_len(warmup + sample_size)) {
  proposal <- rnorm(1, state, 1)
  if (log(runif(1)) < log_target(proposal) - log_target(state)) {
    state <- proposal
  }
  if (iteration > warmup) {
    metropolis_values[iteration - warmup] <- state^2
  }
}
metropolis <- running_mean(metropolis_values)

normalizing_integral <- integrate(
  function(y) y^3 * sin(y^4) * cos(y^5),
  0,
  1,
  rel.tol = 1e-11
)$value
weighted_integral <- integrate(
  function(y) y^5 * sin(y^4) * cos(y^5),
  0,
  1,
  rel.tol = 1e-11
)$value
exact_expectation <- weighted_integral / normalizing_integral

c(
  proposal_six = tail(proposal_six$estimate, 1),
  proposal_four = tail(proposal_four$estimate, 1),
  metropolis = tail(metropolis$estimate, 1),
  exact = exact_expectation
)
```

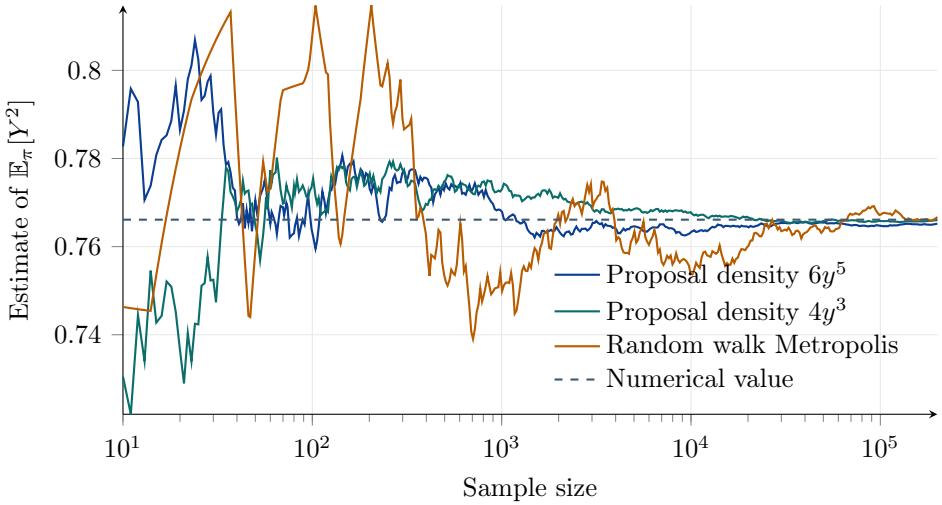

Figure 5

The three paths in Figure 5 fluctuate differently. The two independent estimators use different importance densities, while the random walk Metropolis estimator uses dependent observations and is visibly more variable. All three stabilize around 0.7661155, showing that consistency does not imply identical finite sample behaviour.

Lecture 3 — Monday, September 23, 2019

Rejection and Auxiliary Variable Sampling

Importance sampling estimates expectations without producing observations from the target itself. Rejection sampling instead converts proposals from an easy density into independent draws from the target.

Algorithm 1 Rejection sampler

Input: An unnormalised target $\pi = cg$ with $g \geq 0$ and $0 < \int g < \infty$, a density f that can be sampled directly, and a known constant $K > 0$ such that $Kf(x) \geq g(x)$ for every x

1: **repeat**
2: Generate $X \sim f$ and $U \sim \text{Unif}[0,1]$ independently.
3: **until** $U \leq g(X)/\{Kf(X)\}$

Output: The accepted proposal X

Proposition 3.1 Conditional on acceptance in Algorithm 1, the proposal X has density π. Moreover,

$$\mathbb{P}(\text{accept}) = \frac{1}{cK}.$$

Proof. Since $0 \le g(x)/(Kf(x)) \le 1$, conditioning on $X = x$ gives

$$\mathbb{P}\left(U \le \frac{g(X)}{Kf(X)} \,\middle|\, X = x\right) = \frac{g(x)}{Kf(x)}.$$

The double expectation formula therefore yields

$$\mathbb{P}(\text{accept}) = \mathbb{E}_f\left[\frac{g(X)}{Kf(X)}\right] = \frac{1}{K}\int_{-\infty}^{\infty} g(x)\,\mathrm{d}x = \frac{1}{cK}.$$

For any $y \in \mathbb{R}$,

$$\begin{aligned}\mathbb{P}(X \le y,\ \text{accept}) &= \mathbb{E}_f\left[\mathbb{1}_{\{X \le y\}}\frac{g(X)}{Kf(X)}\right] \\ &= \frac{1}{K}\int_{-\infty}^{y} g(x)\,\mathrm{d}x.\end{aligned}$$

Dividing the last expression by $\mathbb{P}(\text{accept})$ gives

$$\mathbb{P}(X \le y \mid \text{accept}) = c\int_{-\infty}^{y} g(x)\,\mathrm{d}x = \int_{-\infty}^{y} \pi(x)\,\mathrm{d}x.$$

Thus an accepted proposal has density π. □

In M attempts, Proposition 3.1 predicts approximately $M/(cK)$ accepted independent observations. Since c is fixed, efficiency requires making K as small as possible. In the unattainable ideal $f = \pi$, we can choose $K = 1/c$, every proposal is accepted, and the output is independent sampling from π.

The same importance, rejection, and auxiliary variable arguments apply on a discrete state space. In that setting, π, f, and g are probability mass functions, equivalently densities with respect to counting measure, and every integral in the derivations is replaced by a sum.

Example 3.2 Let the target be the standard normal density

$$g(x) = \pi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

and let $f(x) = e^{-|x|}/2$ be the Laplace density. The choice $K = 8$ is valid. When $|x| \leq 2$,

$$8f(x) = 4e^{-|x|} \geq 4e^{-2} > \frac{1}{\sqrt{2\pi}} \geq g(x),$$

and when $|x| \geq 2$,

$$8f(x) = 4e^{-|x|} \geq 4e^{-x^2/2} > \frac{1}{\sqrt{2\pi}} e^{-x^2/2} = g(x).$$

The following R implementation estimates $\mathbb{E}_\pi[X^4] = 3$ from the accepted proposals and creates the envelope shown in Figure 6.

```
set.seed(3431)

attempts <- 10000
K <- 8

proposal <- ifelse(runif(attempts) < 0.5, -1, 1) * rexp(attempts)
proposal_density <- 0.5 * exp(-abs(proposal))
u <- runif(attempts)
height <- u * K * proposal_density
accepted <- height <= dnorm(proposal)
sample <- proposal[accepted]

c(
  acceptance_rate = mean(accepted),
  theoretical_rate = 1 / K,
  fourth_moment = mean(sample^4)
)
```

The geometric interpretation in Figure 6 leads to an auxiliary variable method. Suppose (X, Y) is uniform under the graph of g:

$$(X, Y) \sim \text{Unif}\{(x, y) \in \mathbb{R}^2 : 0 \leq y \leq g(x)\}.$$

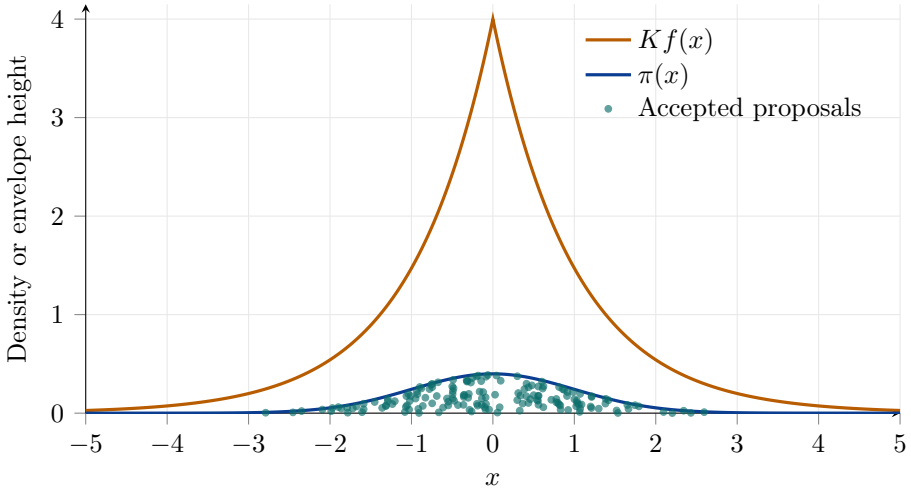

FIGURE 6

For $a < b$,

$$\mathbb{P}(a < X < b) = \frac{\int_a^b g(x)\,\mathrm{d}x}{\int_{-\infty}^{\infty} g(x)\,\mathrm{d}x} = \int_a^b \pi(x)\,\mathrm{d}x,$$

so the marginal density of X is π. If the support of g lies in $[L, R]$ and $g(x) \le K$, we may sample uniformly from the rectangle $[L, R] \times [0, K]$ and retain the point if $Y \le g(X)$. This is a geometric form of rejection sampling and is closely related to slice sampling.

For the target in Example 2.2, take $L = 0$, $R = 1$, and $K = 1$. Generate $X, Y \sim \mathrm{Unif}[0, 1]$ and keep X if $Y \le g(X)$. The mean of the squared accepted X-values estimates the same expectation as before.

Queueing and Financial Applications

Simulation is particularly useful when a familiar analytic model is modified in a way that destroys its closed form. Consider a single server queue and let $Q(t)$ be the number of customers in the system at time $t \ge 0$. If service times are $\mathrm{Exp}(\mu)$ and interarrival times are $\mathrm{Exp}(\lambda)$, then $Q(t)$ is the classical $M/M/1$ queue. If $\mu < \lambda$, then $Q(t) \to \infty$ almost surely. At the critical case $\mu = \lambda$, the queue is null recurrent: it has no stationary probability distribution and returns to zero infinitely often, although $Q(t) \to \infty$ in probability. If $\mu > \lambda$, then

$$\mathbb{P}(Q(t) = i) \longrightarrow \left(1 - \frac{\lambda}{\mu}\right)\left(\frac{\lambda}{\mu}\right)^i, \quad i = 0, 1, 2, \ldots. \tag{3.1}$$

The limiting distribution in (3.1) is easy to verify numerically, for instance with $\mu = 3$, $\lambda = 2$, and a long time horizon. The following event driven implementation advances directly from one arrival or departure to the next and accumulates the time spent at each queue size.

```
simulate_queue <- function(
    horizon,
    arrival_rate,
    service_rate,
    record_path = FALSE) {
  time <- 0
  queue <- 0L
  occupied_time <- numeric(1)
  event_times <- 0
  queue_sizes <- 0L

  while (time < horizon) {
    total_rate <- arrival_rate + service_rate * (queue > 0)
    next_time <- min(horizon, time + rexp(1, total_rate))

    if (length(occupied_time) < queue + 1L) {
      length(occupied_time) <- queue + 1L
      occupied_time[is.na(occupied_time)] <- 0
    }
    occupied_time[queue + 1L] <- occupied_time[queue + 1L] +
      next_time - time
    time <- next_time

    if (time >= horizon) {
      break
    }

    if (runif(1) < arrival_rate / total_rate) {
      queue <- queue + 1L
    } else {
      queue <- queue - 1L
    }
    if (record_path) {
      event_times <- c(event_times, time)
      queue_sizes <- c(queue_sizes, queue)
    }
  }

  list(
```

```
    occupancy = occupied_time / horizon,
    path = data.frame(time = c(event_times, horizon), queue = c(queue_sizes, queue))
  )
}

set.seed(3431)
arrival_rate <- 2
service_rate <- 3
rho <- arrival_rate / service_rate

long_run <- simulate_queue(200000, arrival_rate, service_rate)
display_states <- 0:12
stationary_comparison <- data.frame(
  queue = display_states,
  empirical = long_run$occupancy[display_states + 1L],
  exact = (1 - rho) * rho^display_states
)

short_run <- simulate_queue(50, arrival_rate, service_rate, record_path = TRUE)
queue_path <- short_run$path
```

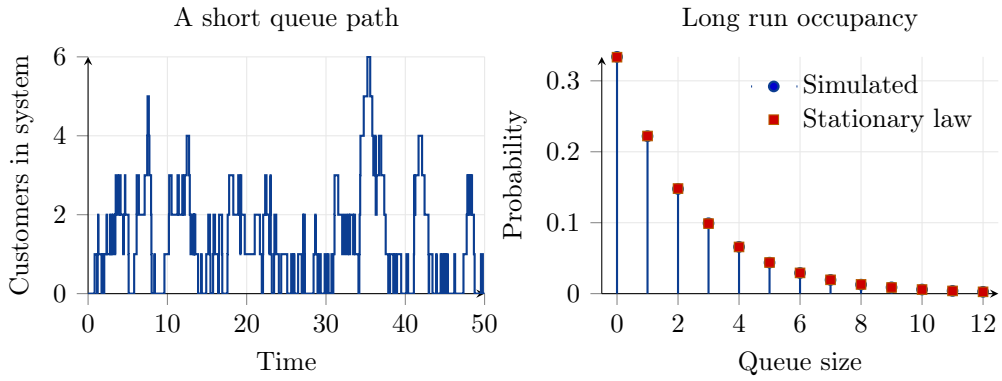

FIGURE 7

The left panel of Figure 7 shows the piecewise constant sample path, while the right panel compares long run time proportions with the geometric probabilities in (3.1). The agreement supplies a numerical check of the stationary calculation.

If the exponential assumptions are replaced by nonexponential service and interarrival laws, the limit is no longer so easily computed, but event driven simulation still applies. We can preserve the means of the original example by taking service times uniform on $[0, 2/3]$ and interarrival times distributed as $Z^2/2$, where $Z \sim \mathcal{N}(0, 1)$.

Monte Carlo methods also price financial derivatives when analytic formulas are unavailable. Suppose that X_t is a stock price governed by

$$\mathrm{d}X_t = bX_t\,\mathrm{d}t + \sigma X_t\,\mathrm{d}B_t, \quad X_0 = a > 0, \tag{3.2}$$

where B_t is Brownian motion. For a small time step $h > 0$,

$$X_{t+h} \mid X_t \approx \mathcal{N}\big(X_t + bX_t h, \sigma^2 X_t^2 h\big)\,. \tag{3.3}$$

A European call option permits the purchase of one share at time $T > 0$ for a strike price $q > 0$. Its payoff is $\max(0, X_T - q)$. In a frictionless market with no dividends, under the risk neutral probability measure, the fair price is

$$\mathbb{E}\big[e^{-rT}\max(0, X_T - q)\big], \tag{3.4}$$

where the drift b in (3.2) is replaced by the risk free rate r. When $\sigma > 0$ and r are constant, the Black–Scholes formula gives the price at time 0 as

$$C_0 = a\Phi(d_1) - qe^{-rT}\Phi(d_2),$$

where

$$d_1 = \frac{\log(a/q) + T(r + \sigma^2/2)}{\sigma\sqrt{T}} \qquad \text{and} \qquad d_2 = \frac{\log(a/q) + T(r - \sigma^2/2)}{\sigma\sqrt{T}}.$$

This payoff has the expected limiting behaviour. If the stock has a very large upward drift, the right to buy it later at the fixed strike price becomes valuable; if the stock is unlikely to exceed the strike price, the option is worth little. We can instead simulate many paths using (3.3) and average the discounted payoffs in (3.4). A single path is highly variable, but many independent paths give a useful estimate.

An Asian call option replaces X_T by the average

$$\overline{X}_{k,T} := \frac{1}{k}\sum_{i=1}^{k} X_{iT/k}.$$

The same risk neutral expectation principle applies, but the corresponding formula is no longer simple. Simulation therefore handles the Asian payoff with almost no conceptual change, illustrating the flexibility of Monte Carlo methods.

Lecture 4 — Monday, September 30, 2019

3. Markov Chain Monte Carlo Foundations

The Metropolis Algorithm and Optimal Scaling

Independent, importance, rejection, and auxiliary variable sampling can all fail in high dimensional problems. Markov chain Monte Carlo replaces independent observations with a dependent process whose limiting distribution is the target.

Suppose that $\pi = cg$ is a complicated density on a state space $\mathcal{X}$. Construct a Markov chain $X_0, X_1, \ldots$ so that X_n is approximately distributed according to π when n is large. Then

$$\mathbb{E}_\pi[h(X)] \approx \frac{1}{M-B} \sum_{i=B+1}^{M} h(X_i), \tag{4.1}$$

where the first B observations form a **warmup period**. The warmup should be long enough for the chain to approach its stationary regime, while $M - B$ should be long enough to control Monte Carlo error.

Algorithm 2 Metropolis algorithm

Input: An initial value X_0 with $g(X_0) > 0$, a proposal scale $\sigma > 0$, an unnormalised nonnegative target g, and a run length M

for $n = 1, \ldots, M$ **do**
  Generate $Y_n \sim \mathcal{N}(X_{n-1}, \sigma^2 \mathbf{I})$.
  Generate $U_n \sim \text{Unif}[0, 1]$ independently.
  Set $A_n \leftarrow g(Y_n)/g(X_{n-1})$.
  if $U_n < A_n$ **then**
    Set $X_n \leftarrow Y_n$. ▷ Accept the proposal.
  else
    Set $X_n \leftarrow X_{n-1}$. ▷ Reject the proposal.
  end if
end for

Output: The Markov chain $X_0, X_1, \ldots, X_M$

Only a ratio of target densities appears in Algorithm 2, so the unknown normalising constant c cancels. The Gaussian proposal is symmetric about the current state. If every proposal were accepted, the successive increments would form an ordinary random walk; this version is therefore called the **random walk Metropolis algorithm**.

Selecting B is difficult. In practice, one examines output from several runs and uses initial values drawn from an overdispersed distribution that covers the important parts of the state space. A central initial value is often helpful, but apparent stability from a single starting point is not decisive evidence of convergence.

The target from Example 2.2 can be sampled using $Y_n \sim \mathcal{N}(X_{n-1}, 1)$, after rejecting proposals outside $(0, 1)$. As Figure 5 shows, the estimate of $\mathbb{E}_\pi[X^2]$ remains near 0.766, but is more variable than the independent sampling estimates. A two dimensional example is

$$\pi(x_1, x_2) = C\,|\cos(\sqrt{x_1 x_2})|\,\mathbb{1}_{\{0 \le x_1 \le 5,\ 0 \le x_2 \le 4\}},$$

with

$$h(x_1, x_2) = e^{x_1} + x_2^2.$$

The following R implementation performs thirty independent random walk Metropolis runs. Each run contains 8,000 iterations, of which the first 1,000 are discarded as warmup. The same implementation evaluates the numerator and denominator defining $\mathbb{E}_\pi[h]$ by numerical integration.

```
target_kernel <- function(x) {
  if (x[1] < 0 || x[1] > 5 || x[2] < 0 || x[2] > 4) {
    return(0)
  }
  abs(cos(sqrt(x[1] * x[2])))
}

run_metropolis <- function(iterations, warmup, proposal_sd = 1) {
  state <- c(2.5, 2)
  trace <- matrix(NA_real_, nrow = iterations, ncol = 2)
  accepted <- 0L

  for (iteration in seq_len(iterations)) {
    proposal <- state + rnorm(2, sd = proposal_sd)
    log_ratio <- log(target_kernel(proposal)) - log(target_kernel(state))

    if (log(runif(1)) < log_ratio) {
      state <- proposal
      accepted <- accepted + 1L
```

```
    }
    trace[iteration, ] <- state
  }

  retained <- trace[(warmup + 1L):iterations, , drop = FALSE]
  list(
    retained = retained,
    estimate = mean(exp(retained[, 1]) + retained[, 2]^2),
    acceptance_rate = accepted / iterations
  )
}

set.seed(3431)
run_count <- 30
runs <- replicate(
  run_count,
  run_metropolis(iterations = 8000, warmup = 1000),
  simplify = FALSE
)

run_estimates <- data.frame(
  run = seq_len(run_count),
  estimate = vapply(runs, function(result) result$estimate, numeric(1))
)

display_sample <- runs[[1]]$retained
depleted_band <- data.frame(
  x1 = seq((pi / 2)^2 / 4, 5, length.out = 300)
)
depleted_band$x2 <- (pi / 2)^2 / depleted_band$x1

normalizing_integral <- integrate(
  function(x1) {
    vapply(
      x1,
      function(value) integrate(
        function(x2) abs(cos(sqrt(value * x2))),
        0,
        4,
        rel.tol = 1e-9
      )$value,
      numeric(1)
    )
  },
  0,
```

```
  5,
  rel.tol = 1e-8
)$value
weighted_integral <- integrate(
  function(x1) {
    vapply(
      x1,
      function(value) integrate(
        function(x2) {
          (exp(value) + x2^2) * abs(cos(sqrt(value * x2)))
        },
        0,
        4,
        rel.tol = 1e-9
      )$value,
      numeric(1)
    )
  },
  0,
  5,
  rel.tol = 1e-8
)$value
exact_expectation <- weighted_integral / normalizing_integral

c(
  minimum = min(run_estimates$estimate),
  maximum = max(run_estimates$estimate),
  exact = exact_expectation,
  acceptance_rate = mean(vapply(runs, function(result) {
    result$acceptance_rate
  }, numeric(1)))
)
```

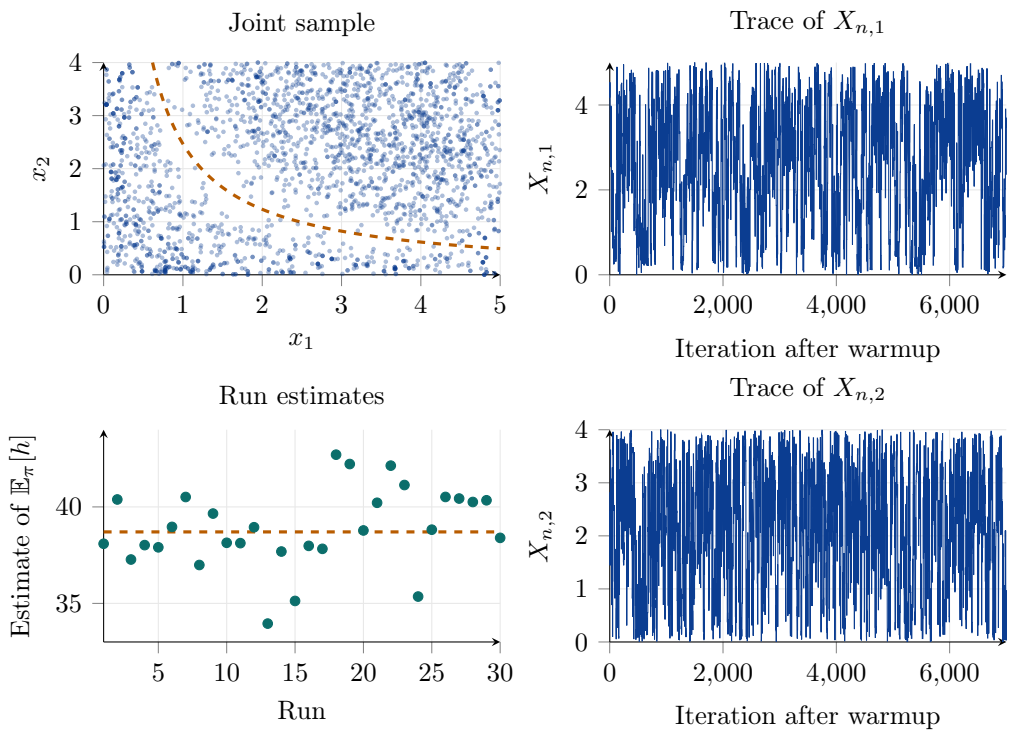

FIGURE 8

The numerical integration value is 38.70437, shown by the dashed line in the lower left panel of Figure 8. The thirty run estimates range from 33.95 to 42.71, illustrating why values between roughly 34 and 44 are typical for runs of this length. The coordinate traces remain within their respective intervals and appear individually unremarkable. In the joint sample, however, the depleted band follows the dashed curve

$$x_2 = \frac{(\pi/2)^2}{x_1},$$

on which the target density vanishes. This comparison shows why a joint diagnostic can reveal structure that neither marginal trace displays.

The proposal scale controls a basic tradeoff. If σ is too small, nearly every proposal is accepted but the chain moves slowly. If σ is too large, most proposals are rejected and the chain again moves slowly. A useful scale lies between these extremes.

Remark 4.1 Under a particular idealized high dimensional limit, the asymptotically optimal acceptance rate for random walk Metropolis is 0.234. This number is a guide rather than a universal prescription: the relevant principle is that the acceptance rate should be far from both 0 and 1. The scaling limit is developed by Roberts, Gelman, and Gilks (1997) and reviewed by Roberts and Rosenthal (2001).

The R code below compares three proposal scales for a standard normal target. Its output is displayed in Figure 9.

```
rw_metropolis <- function(iterations, proposal_sd, initial = 0) {
  trace <- numeric(iterations)
  trace[1] <- initial
  accepted <- 0

  for (iteration in 2:iterations) {
    proposal <- rnorm(1, trace[iteration - 1], proposal_sd)
    log_ratio <- dnorm(proposal, log = TRUE) -
      dnorm(trace[iteration - 1], log = TRUE)

    if (log(runif(1)) < log_ratio) {
      trace[iteration] <- proposal
      accepted <- accepted + 1
    } else {
      trace[iteration] <- trace[iteration - 1]
    }
  }

  list(
    trace = trace,
    acceptance_rate = accepted / (iterations - 1)
  )
}

set.seed(3431)
small <- rw_metropolis(5000, proposal_sd = 0.08)
balanced <- rw_metropolis(5000, proposal_sd = 2.4)
large <- rw_metropolis(5000, proposal_sd = 18)

c(
  small = small$acceptance_rate,
  balanced = balanced$acceptance_rate,
  large = large$acceptance_rate
```

)

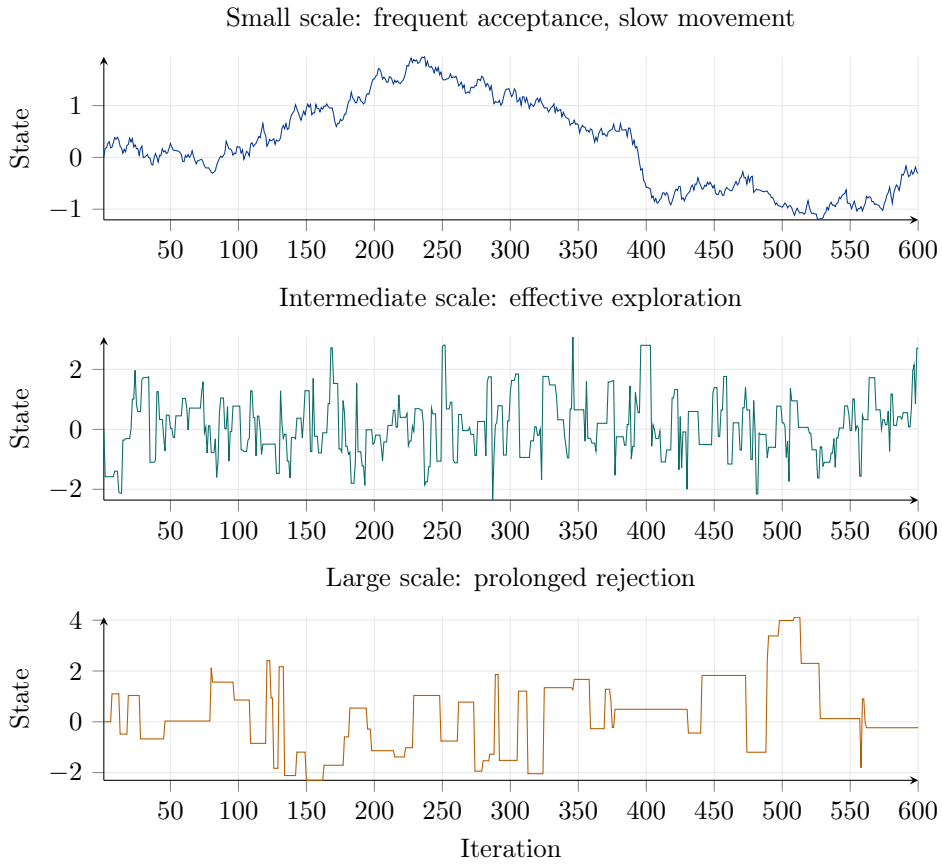

FIGURE 9

Monte Carlo Error and Confidence Intervals

Correlation usually makes the standard error of a Markov chain estimator larger than that of an independent estimator. One direct approach is to run the chain many times from independently chosen, overdispersed starting values, compute one estimate from each run, and then treat the collection of run level estimates as independent. More commonly, one estimates the uncertainty from a single long run.

Let

$$\widetilde{h}(x) := h(x) - \mathbb{E}_\pi[h(X)],$$

so that $\mathbb{E}_\pi[\widetilde{h}(X)] = 0$. Assume that B is large enough for X_i to be approximately stationary when

$i > B$, and write $N = M - B$. The variance of the estimator in (4.1) is approximately

$$\begin{aligned} v &:= \mathrm{Var}_\pi\left(\frac{1}{N}\sum_{i=B+1}^{M} h(X_i)\right) \\ &\approx \frac{1}{N^2}\left(N\mathbb{E}_\pi[\widetilde{h}(X_i)^2] + 2(N-1)\mathbb{E}_\pi[\widetilde{h}(X_i)\widetilde{h}(X_{i+1})] + 2(N-2)\mathbb{E}_\pi[\widetilde{h}(X_i)\widetilde{h}(X_{i+2})] + \cdots\right) \\ &\approx \frac{1}{N}\left(\mathrm{Var}_\pi(h) + 2\sum_{k=1}^{\infty}\mathrm{Cov}_\pi(h(X_0), h(X_k))\right) \\ &= \frac{\mathrm{Var}_\pi(h)}{N}\left(1 + 2\sum_{k=1}^{\infty}\rho_k\right), \end{aligned}$$

where

$$\rho_k := \mathrm{Corr}_\pi(h(X_0), h(X_k)).$$

Here $0 < \mathrm{Var}_\pi(h) < \infty$, and the passage to the infinite series requires the stationary autocovariance series to converge; absolute summability is a simple sufficient condition. The multiplicative factor

$$\text{varfact} := 1 + 2\sum_{k=1}^{\infty}\rho_k = \sum_{k=-\infty}^{\infty}\rho_k \tag{4.2}$$

is also called the **integrated autocorrelation time**. Thus

$$\mathrm{se}_{\mathrm{MCMC}} \approx \sqrt{\frac{\mathrm{Var}_\pi(h)}{N}\,\text{varfact}} = \mathrm{se}_{\text{independent}}\sqrt{\text{varfact}}. \tag{4.3}$$

Both $\mathrm{Var}_\pi(h)$ and the correlations in (4.2) can be estimated from a run. The R function `acf` displays estimated autocorrelations, and the same values can be computed directly from the empirical lagged covariance formula. The infinite sum must be truncated, perhaps when $|\widehat{\rho}_k| < 0.05$ or when $\widehat{\rho}_k$ first becomes negative. This cutoff is an approximation and can be unstable. A well designed chain usually has varfact > 1, although antithetic constructions can sometimes reduce it below 1.

Lecture 5 — Monday, October 07, 2019

Suppose the estimator in (4.1) has observed value e and estimated variance v. If an appropriate Markov chain central limit theorem holds, then

$$e \approx \mathcal{N}(u, v) \qquad \text{and} \qquad u := \mathbb{E}_\pi[h(X)],$$

so

$$(e - u)v^{-1/2} \approx \mathcal{N}(0, 1).$$

An approximate 95% confidence interval is therefore

$$\left(e - 1.96\sqrt{v}, e + 1.96\sqrt{v}\right). \tag{5.1}$$

When the variance is estimated from a modest number of approximately independent batch or run means, a Student t-quantile is often used as an additional approximation.

The independent central limit theorem does not justify (5.1). A Markov chain central limit theorem can fail. One important sufficient condition is that the chain be geometrically ergodic and that

$$\mathbb{E}_\pi[|h|^{2+\delta}] < \infty$$

for some $\delta > 0$. For a reversible geometrically ergodic chain, a second moment is sufficient. The geometric ergodicity conditions are discussed by Roberts and Rosenthal, "Geometric ergodicity and hybrid Markov chains" (1997). Consistent variance estimators can also be used to build slightly wider confidence intervals without assuming a central limit theorem, as developed by Rosenthal, "Simple confidence intervals for MCMC without CLTs" (2017).

Irreducibility, Aperiodicity, and Reversibility

The validity of the Metropolis algorithm follows from standard Markov chain theory. Write

$$\mathbb{P}(i, j) := \mathbb{P}(X_{n+1} = j \mid X_n = i)$$

in a discrete state space, and

$$\mathbb{P}(x, A) := \mathbb{P}(X_{n+1} \in A \mid X_n = x)$$

in a general state space. For a target density π, let

$$\Pi(A) := \int_A \pi(x)\,\mathrm{d}x.$$

Definition 5.1 A Markov chain with target probability measure Π is **Π-irreducible** if, from every state x, it has positive probability of eventually reaching every measurable set A with $\Pi(A) > 0$. In a discrete state space, this means that for every relevant pair of states i, j, there is an $n \in \mathbb{N}$ such that

$$\mathbb{P}(X_n = j \mid X_0 = i) > 0.$$

The chain is **aperiodic** if it is not forced to move cyclically through disjoint nonempty classes. It has **stationary distribution** Π if $X_0 \sim \Pi$ implies $X_1 \sim \Pi$.

In a discrete irreducible chain, aperiodicity is equivalent to

$$\gcd\{n : \mathbb{P}^n(i,i) > 0\} = 1$$

for every state i. Metropolis chains are usually aperiodic because rejection gives $\mathbb{P}(x, \{x\}) > 0$. Normal proposals also typically give positive transition density throughout a set of positive target measure.

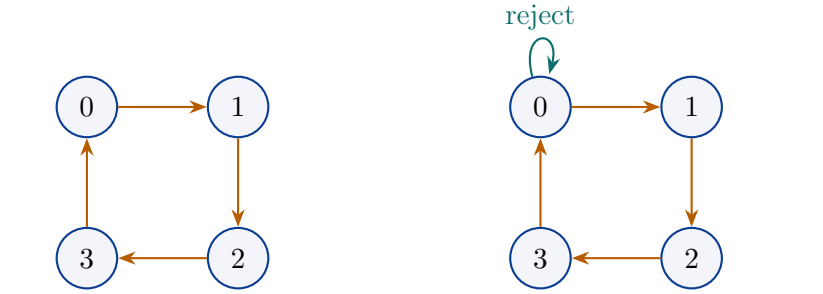

FIGURE 10

The left panel of Figure 10 illustrates a failure of aperiodicity: the greatest common divisor of the possible return times is four. A deterministic proposal from x to $x + 1 \bmod 4$ produces this

behaviour when every proposal is accepted. By contrast, a symmetric nearest neighbour walk on the same cycle has period two. The rejection loop in the right panel introduces a return time of one, so the greatest common divisor becomes one and the chain is aperiodic.

Theorem 5.2 (Markov chain ergodic theorem) Suppose that a Markov chain is Π-irreducible and has stationary probability density π. For almost every initial state $X_0 = x$ with respect to Π, the following statements hold.

1. If $\mathbb{E}_\pi[|h|] < \infty$, then
$$\frac{1}{n}\sum_{i=1}^{n} h(X_i) \xrightarrow{\text{a.s.}} \mathbb{E}_\pi[h(X)] = \int_{\mathcal{X}} h(x)\pi(x)\,\mathrm{d}x.$$
2. If the chain is also aperiodic, then
$$\mathbb{P}_x(X_n \in A) \longrightarrow \Pi(A)$$
for every measurable $A \subseteq \mathcal{X}$.

To verify stationarity of the Metropolis chain, begin with a discrete state space. Let $q(i,j)$ be the probability of proposing j from i, and suppose that $q(i,j) = q(j,i)$. For $i \neq j$, the transition probability is
$$\mathbb{P}(i,j) = q(i,j)\alpha(i,j) \qquad \text{and} \qquad \alpha(i,j) = \min\left\{1, \frac{\pi(j)}{\pi(i)}\right\}.$$
It follows that
$$\pi(i)\mathbb{P}(i,j) = q(i,j)\min\{\pi(i),\pi(j)\} = q(j,i)\min\{\pi(i),\pi(j)\} = \pi(j)\mathbb{P}(j,i).$$
The same identity is automatic when $i = j$.

Definition 5.3 A Markov chain is **reversible** with respect to π if it satisfies the detailed balance equations
$$\pi(i)\mathbb{P}(i,j) = \pi(j)\mathbb{P}(j,i)$$
for every pair of states i, j. In a general state space, detailed balance is written
$$\Pi(\mathrm{d}x)\mathbb{P}(x,\mathrm{d}y) = \Pi(\mathrm{d}y)\mathbb{P}(y,\mathrm{d}x).$$

Reversibility implies stationarity. Indeed, if $X_0 \sim \pi$, then

$$\begin{aligned}\mathbb{P}(X_1 = j) &= \sum_{i \in \mathcal{X}} \mathbb{P}(X_0 = i)\mathbb{P}(i, j) = \sum_{i \in \mathcal{X}} \pi(i)\mathbb{P}(i, j) \\ &= \sum_{i \in \mathcal{X}} \pi(j)\mathbb{P}(j, i) = \pi(j).\end{aligned}$$

Thus $X_1 \sim \pi$. Together with irreducibility and aperiodicity, Theorem 5.2 now proves convergence of the Metropolis chain.

The same argument works in a continuous state space. For symmetric proposal density $q(x, y) = q(y, x)$, and for $x \neq y$,

$$\mathbb{P}(x, \mathrm{d}y) = q(x, y) \min\left\{1, \frac{\pi(y)}{\pi(x)}\right\} \mathrm{d}y.$$

Consequently,

$$\pi(x)\mathbb{P}(x, \mathrm{d}y)\,\mathrm{d}x = q(x, y) \min\{\pi(x), \pi(y)\}\,\mathrm{d}y\,\mathrm{d}x = \pi(y)\mathbb{P}(y, \mathrm{d}x)\,\mathrm{d}y.$$

If $X_0 \sim \Pi$, then for every measurable A,

$$\mathbb{P}(X_1 \in A) = \int_{\mathcal{X}} \int_A \pi(x)\mathbb{P}(x, \mathrm{d}y)\,\mathrm{d}x = \int_A \int_{\mathcal{X}} \pi(y)\mathbb{P}(y, \mathrm{d}x)\,\mathrm{d}y = \int_A \pi(y)\,\mathrm{d}y = \Pi(A).$$

The target is therefore stationary in the general case as well.

Validity Examples and Initial Distributions

The following examples isolate the roles of symmetry, irreducibility, and aperiodicity.

Example 5.4

1. Let $\mathcal{X} = \mathbb{Z}$, let $\pi(x) = 2^{-|x|}/3$, and propose either neighbour with probability $1/2$. This is a reversible Metropolis chain. Rejection makes it aperiodic, and every state communicates with every other state, so it is irreducible and converges to π. The transition structure appears in Figure 11.

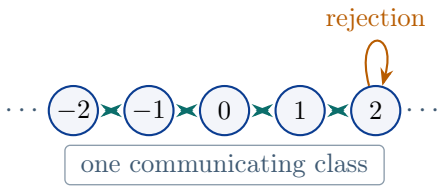

FIGURE 11

2. Retain the same proposal but take $\pi(0) = 0$ and $\pi(x) = 2^{-|x|-1}$ for $x \neq 0$. The chain is reversible, stationary, and aperiodic, but it cannot cross from the positive integers to the negative integers. It is not irreducible and need not converge to the full target, as Figure 12 makes clear.

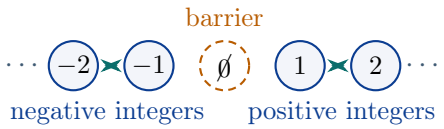

FIGURE 12

3. Retain the target from (2), but propose each y with $1 \leq |x - y| \leq 2$ with probability $1/4$. The chain can now jump over 0, so irreducibility is restored; see Figure 13.

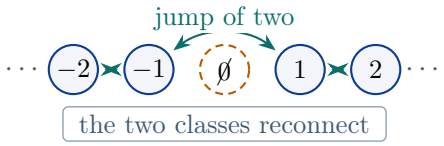

FIGURE 13

4. On $\mathbb{R}$, take $\pi(x) = Ce^{-x^6}$ and propose uniformly on $[x-1, x+1]$. The proposal is symmetric. The chain is irreducible because its n-step transition has positive density throughout $(x-n, x+n)$, and it is aperiodic because rejection has positive probability. The proposal and rejection mechanism are shown in Figure 14.

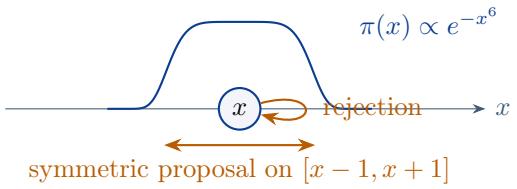

FIGURE 14

5. Modify (4) by restricting the target to $(-\infty, 2] \cup [4, \infty)$. A proposal of radius 1 cannot cross the gap, so the chain is not irreducible, as shown in Figure 15.

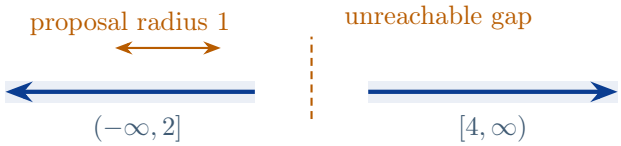

FIGURE 15

6. In (5), enlarge the proposal to $[x-5, x+5]$. The chain can cross the gap and is again irreducible; see Figure 16.

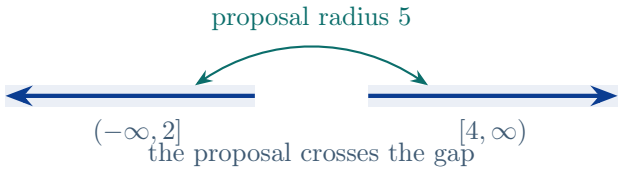

FIGURE 16

7. If the proposal is instead uniform on $[x-5, x+10]$, it is not symmetric, so the ordinary Metropolis acceptance probability is invalid. Some moves can also be proposed in one direction but not the reverse. The Metropolis–Hastings correction remains valid: it assigns such a move acceptance probability zero. Figure 17 isolates this one-way support problem, which can still impede irreducibility or efficiency.

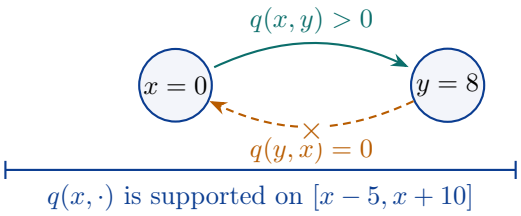

FIGURE 17

The diagrams in (1)–(3) show why a state with zero target mass can be either a barrier or merely a state that the chain jumps over. The diagrams in (4)–(6) show the analogous distinction on a continuous state space. The one-way transition in (7) isolates the additional support problem caused by an asymmetric proposal. Irreducibility depends jointly on the support of the target and the reach of the proposal, not on either one in isolation.

The qualification concerning almost every initial state with respect to Π in Theorem 5.2 cannot always be removed. Let

$$\mathcal{X} = \{1, 2, 3, \ldots\},$$

with $\mathbb{P}(1, \{1\}) = 1$, while for $x \geq 2$,

$$\mathbb{P}(x, \{1\}) = \frac{1}{x^2} \qquad \text{and} \qquad \mathbb{P}(x, \{x+1\}) = 1 - \frac{1}{x^2}.$$

The stationary distribution is the point mass δ_1. The chain is δ_1-irreducible and aperiodic. However, if $X_0 = x \geq 2$, then

$$\mathbb{P}(X_n = x + n \text{ for all } n) = \prod_{j=x}^{\infty} \left(1 - \frac{1}{j^2}\right) > 0,$$

because $\sum_{j=x}^{\infty} j^{-2} < \infty$. Hence $\mathbb{P}(X_n = 1)$ does not converge to 1 from these exceptional states. Convergence does hold from $X_0 = 1$, which accounts for almost every initial state with respect to δ_1. Stronger conditions, such as Harris recurrence, often restore convergence from every state encountered in practical algorithms.

Lecture 6 — Monday, October 21, 2019

4. General MCMC Algorithms and Bayesian Computation

Metropolis–Hastings and Its Variants

The ordinary Metropolis algorithm assumes a symmetric proposal density. The correction introduced by Hastings, "Monte Carlo sampling methods using Markov chains and their applications" (1970), permits an asymmetric proposal $q(x, y)$. Reciprocal proposal support is convenient but not necessary: if $q(x, y) > 0$ and $q(y, x) = 0$, the proposed move is simply rejected.

Algorithm 3 Metropolis–Hastings algorithm

Input: An initial value X_0 with $\pi(X_0) > 0$, a proposal density q, a nonnegative target density π known up to normalization, and a run length M

for $n = 1, \dots, M$ **do**
  Write $x \leftarrow X_{n-1}$ and generate $Y_n = y \sim q(x, \cdot)$.
  Generate $U_n \sim \mathrm{Unif}[0, 1]$ independently.
  Set $A_n \leftarrow \pi(y)q(y, x)/\{\pi(x)q(x, y)\}$, with a zero numerator interpreted as $A_n = 0$.
  if $U_n < A_n$ **then**
    Set $X_n \leftarrow y$.
  else
    Set $X_n \leftarrow x$.
  end if
end for

Output: The Markov chain $X_0, X_1, \dots, X_M$

The Hastings factor compensates for an imbalance between forward and reverse proposal probabilities. If $q(x, y)$ is much larger than $q(y, x)$, the uncorrected algorithm would move from x to y too often, so the correction lowers the acceptance probability.

Proposition 6.1 The Metropolis–Hastings transition in Algorithm 3 is reversible with respect to π.

Proof. For a move that can be proposed, so that $q(x, y) > 0$, the acceptance probability is

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

If $q(x, y) = 0$, the flux from x to y is zero; the reverse move, if it can be proposed, has zero acceptance probability because its Hastings numerator contains $q(x, y)$. Thus detailed balance holds for this pair. When $q(x, y) > 0$,

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

which is symmetric in x and y. Thus detailed balance holds. Reversibility follows, and π is stationary. □

If the resulting chain is irreducible and aperiodic, Theorem 5.2 applies. The method again needs only an unnormalised target because the normalising constant cancels from the acceptance ratio.

For the two dimensional target considered after Algorithm 2, we can use a state dependent proposal

$$Y_n \sim \mathcal{N}\big(X_{n-1}, \sigma^2(1 + \|X_{n-1}\|^2)^2\mathbf{I}\big).$$

Its density is proportional to

$$q(\boldsymbol{x}, \boldsymbol{y}) \propto (1 + \|\boldsymbol{x}\|^2)^{-2} \exp\left(-\frac{\|\boldsymbol{y} - \boldsymbol{x}\|^2}{2\sigma^2(1 + \|\boldsymbol{x}\|^2)^2}\right).$$

The proposal spreads out as the chain moves away from the centre. It is not symmetric, so the full Hastings correction is required. Simulations typically estimate 38.7044 with a standard error near 2.

Several useful algorithms are special cases of Algorithm 3.

Definition 6.2 In the **independence sampler**, proposals Y_n are independent draws from a fixed density q, independently of X_{n-1}. The acceptance ratio is

$$A_n = \frac{\pi(Y_n)q(X_{n-1})}{\pi(X_{n-1})q(Y_n)}.$$

If $q = \pi$, every proposal is accepted and the output is independent. Proposal quality matters even

in one dimension. For

$$\pi(x) = e^{-x}\mathbb{1}_{\{x>0\}} \qquad \text{and} \qquad q(y) = ke^{-ky}\mathbb{1}_{\{y>0\}},$$

the acceptance ratio from x to y is

$$A(x, y) = e^{(k-1)(y-x)}.$$

When $k = 1$, sampling is independent. With $k = 0.01$, proposals are much too large but performance is tolerable; with $k = 5$, proposals are too concentrated and performance is extremely poor. The convergence rate analysis later explains this asymmetry.

The following implementation starts each sampler at 10, away from the bulk of the target, and produces the traces in Figure 18.

```
independence_metropolis <- function(iterations, proposal_rate, initial = 10) {
  trace <- numeric(iterations)
  trace[1] <- initial
  accepted <- 0L

  for (iteration in 2:iterations) {
    proposal <- rexp(1, proposal_rate)
    log_ratio <- (proposal_rate - 1) *
      (proposal - trace[iteration - 1])

    if (log(runif(1)) < log_ratio) {
      trace[iteration] <- proposal
      accepted <- accepted + 1L
    } else {
      trace[iteration] <- trace[iteration - 1]
    }
  }

  list(
    trace = trace,
    acceptance_rate = accepted / (iterations - 1)
  )
}

set.seed(3431)
diffuse <- independence_metropolis(5000, proposal_rate = 0.01)
matched <- independence_metropolis(5000, proposal_rate = 1)
concentrated <- independence_metropolis(5000, proposal_rate = 5)
```

```
c(
  diffuse = diffuse$acceptance_rate,
  matched = matched$acceptance_rate,
  concentrated = concentrated$acceptance_rate
)
```

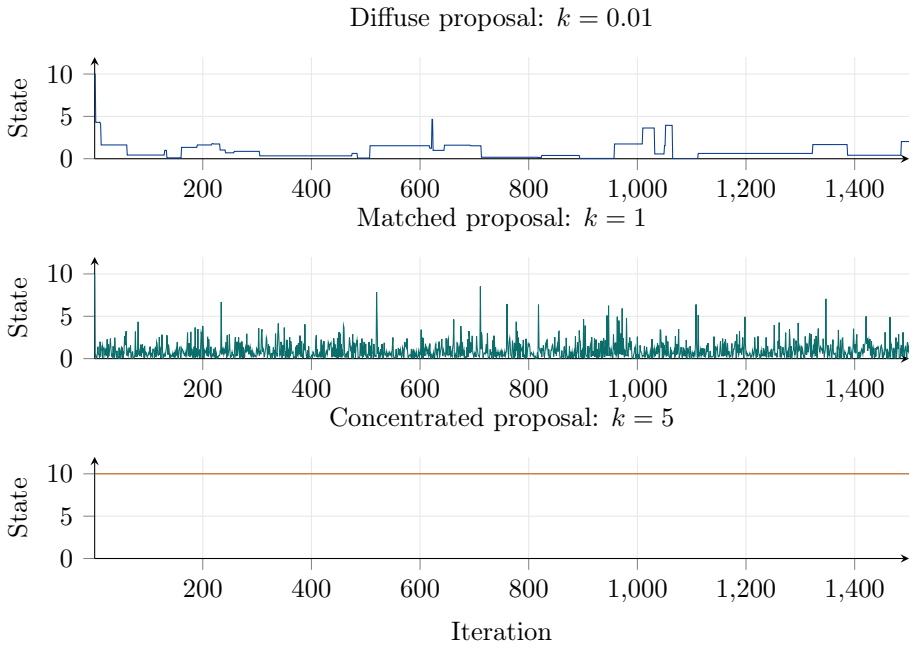

FIGURE 18

The diffuse sampler in Figure 18 accepts only about 2.2% of its proposals, but it nevertheless reaches the target's high density region. The matched proposal produces independent observations and accepts every proposal. By contrast, the concentrated sampler accepts none of the first 5,000 proposals in this seeded run and remains at its initial state. This finite run preview is consistent with the convergence theory developed later.

Definition 6.3 For a positive differentiable target π, the **Langevin algorithm** is the Metropolis–Hastings algorithm with proposal

$$Y_n \sim \mathcal{N}\left(X_{n-1} + \frac{\sigma^2}{2}\nabla \log \pi(X_{n-1}), \sigma^2 \mathbf{I}\right).$$

The gradient shifts proposals toward increasing target density. The construction is based on a

discrete approximation to a Langevin diffusion. It is often more efficient than a random walk proposal, but evaluating $\nabla \log \pi$ may be difficult. Langevin convergence theory is developed by Roberts and Tweedie (1996), while Roberts and Rosenthal (1998) study optimal scaling.

Definition 6.4 In **componentwise Markov chain Monte Carlo**, one coordinate is proposed at a time and every other coordinate remains fixed. For example, when updating coordinate i,

$$Y_{n,i} \sim \mathcal{N}(X_{n-1,i}, \sigma^2) \qquad \text{and} \qquad Y_{n,j} = X_{n-1,j} \quad \text{for } j \neq i.$$

The coordinates may be updated sequentially in systematic scan, or a coordinate may be selected uniformly at random at each iteration in random scan. One full systematic sweep corresponds to d individual random scan updates. With symmetric proposals this method is called componentwise Metropolis or Metropolis within Gibbs. With asymmetric proposals it is called componentwise Metropolis–Hastings or Metropolis–Hastings within Gibbs.

Each coordinate update is a reversible Metropolis–Hastings step and therefore preserves π. A composition of these steps also preserves π, even though a full systematic sweep need not itself be reversible. Subject to irreducibility and aperiodicity, both systematic and random scans converge to the target. For the two dimensional target above, either scan can estimate 38.7044 while avoiding simultaneous movement of both coordinates.

Bayesian Hierarchical Models

Bayesian statistics begins with an unknown parameter $\boldsymbol{\theta}$, a likelihood $f_{\boldsymbol{Y}\mid\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{\theta})$, and a prior density $f_{\boldsymbol{\theta}}(\boldsymbol{\theta})$. Their product gives the joint density, and conditioning on the observed data gives the posterior:

$$\pi(\boldsymbol{\theta}) = f_{\boldsymbol{\theta}\mid\boldsymbol{Y}}(\boldsymbol{\theta} \mid \boldsymbol{y}) = C f_{\boldsymbol{\theta}}(\boldsymbol{\theta}) f_{\boldsymbol{Y}\mid\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{\theta}), \tag{6.1}$$

where the normalising constant C is often intractable. This is precisely the setting in which density ratio methods are useful.

Consider a variance components model. A population has unknown mean μ and contains K groups. For group i, observations $Y_{i1}, \ldots, Y_{iJ_i}$ satisfy

$$Y_{ij} \mid \theta_i, W \sim \mathcal{N}(\theta_i, W),$$

conditionally independently. The group means are linked through

$$\theta_i \mid \mu, V \sim \mathcal{N}(\mu, V),$$

also conditionally independently.

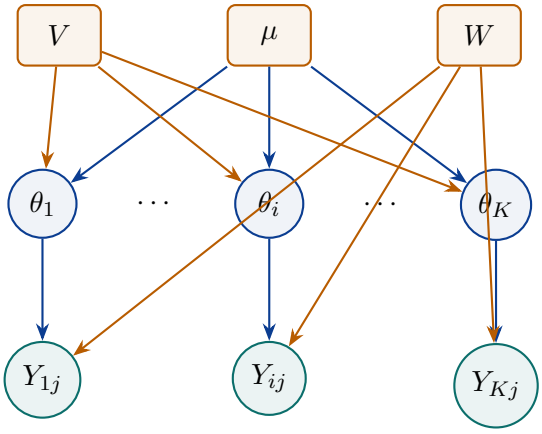

FIGURE 19

The dependency structure is summarized by Figure 19. Assign independent conjugate priors

$$V \sim \text{IG}(a_1, b_1), \qquad W \sim \text{IG}(a_2, b_2), \qquad \text{and} \qquad \mu \sim \mathcal{N}(a_3, b_3),$$

where $a_1, b_1, a_2, b_2, b_3 > 0$, and where $\text{IG}(a, b)$ has density

$$\frac{b^a}{\Gamma(a)} e^{-b/x} x^{-a-1}, \quad x > 0.$$

Let $J_\bullet = \sum_{i=1}^K J_i$. The complete joint density is

$$\begin{aligned} & f(V, W, \mu, \theta_1, \ldots, \theta_K, \boldsymbol{Y}) \\ &= \frac{b_1^{a_1}}{\Gamma(a_1)} e^{-b_1/V} V^{-a_1-1} \frac{b_2^{a_2}}{\Gamma(a_2)} e^{-b_2/W} W^{-a_2-1} \frac{e^{-(\mu-a_3)^2/(2b_3)}}{\sqrt{2\pi b_3}} \\ &\quad \times (2\pi V)^{-K/2} \exp\left(-\frac{1}{2V} \sum_{i=1}^K (\theta_i - \mu)^2\right) (2\pi W)^{-J_\bullet/2} \exp\left(-\frac{1}{2W} \sum_{i=1}^K \sum_{j=1}^{J_i} (Y_{ij} - \theta_i)^2\right). \end{aligned} \tag{6.2}$$

Consequently, up to a constant independent of the unknowns, the joint density of the parameters and data is

$$\begin{aligned} & f(V, W, \mu, \theta_1, \ldots, \theta_K, \boldsymbol{Y}) \\ &\quad \propto e^{-b_1/V} V^{-a_1-1} e^{-b_2/W} W^{-a_2-1} e^{-(\mu-a_3)^2/(2b_3)} V^{-K/2} W^{-J_\bullet/2} \end{aligned}$$

$$\times \exp\left(-\frac{1}{2V}\sum_{i=1}^{K}(\theta_i-\mu)^2\right)\exp\left(-\frac{1}{2W}\sum_{i=1}^{K}\sum_{j=1}^{J_i}(Y_{ij}-\theta_i)^2\right). \tag{6.3}$$

The posterior density is proportional to the same expression because the marginal density of the observed data contributes only another normalising constant.

Variance components models have been used to predict law school performance across 82 schools, study melanoma recurrence across 19 patient groups, compare 12 baseball players, and analyze fabric dyes from 6 batches. These applications are documented in Rubin's law school study, the melanoma recurrence study, and Albert's baseball analysis. The dyestuff example is associated with Davies and Goldsmith, Box and Tiao, and Gelfand and Smith. The posterior dimension is $K+3$. Direct numerical integration becomes infeasible, importance densities are difficult to design, and rejection samplers can accept essentially no proposals.

For a large posterior, computations should be performed on the logarithmic scale. In particular, the Metropolis acceptance test can be written

$$\log U_n < \log \pi(Y_n) - \log \pi(X_{n-1}),$$

which avoids overflow and often simplifies products into sums. For example, on a bounded state space, a target kernel of the form

$$\pi(\boldsymbol{x}) = \exp\left(\sum_{i<j}|x_j - x_i|\right)$$

has log kernel $\sum_{i<j}|x_j - x_i|$, so it is preferable to enter the latter expression directly when coding density ratios.

Lecture 7 — Monday, October 28, 2019

The Gibbs Sampler

Definition 7.1 The **Gibbs sampler** is a componentwise Metropolis–Hastings algorithm in which the proposal for coordinate i is its full conditional distribution under π, conditional on the current values of every other coordinate.

For a proposed move from $\boldsymbol{x}$ to $\boldsymbol{y}$ that changes only coordinate i, write $\boldsymbol{x}^{(-i)}$ for all coordinates except i. The proposal density has the form

$$q_i(\boldsymbol{x}, \boldsymbol{y}) = C(\boldsymbol{x}^{(-i)})\pi(\boldsymbol{y})$$

whenever $\boldsymbol{x}^{(-i)} = \boldsymbol{y}^{(-i)}$, and it is zero otherwise. Since

$$C(\boldsymbol{x}^{(-i)}) = C(\boldsymbol{y}^{(-i)}),$$

the Hastings ratio is

$$\frac{\pi(\boldsymbol{y})q_i(\boldsymbol{y}, \boldsymbol{x})}{\pi(\boldsymbol{x})q_i(\boldsymbol{x}, \boldsymbol{y})} = \frac{\pi(\boldsymbol{y})C(\boldsymbol{y}^{(-i)})\pi(\boldsymbol{x})}{\pi(\boldsymbol{x})C(\boldsymbol{x}^{(-i)})\pi(\boldsymbol{y})} = 1.$$

Every Gibbs proposal is therefore accepted. Intuitively, replacing one coordinate by a draw from its stationary conditional distribution leaves the joint stationary distribution unchanged.

For the variance components model, treating all quantities except μ as constant in (6.3) gives

$$\pi(\mu \mid \text{rest}) \propto \exp\left(-\frac{(\mu - a_3)^2}{2b_3} - \frac{1}{2V}\sum_{i=1}^{K}(\theta_i - \mu)^2\right).$$

Collecting the quadratic and linear terms in μ gives

$$\pi(\mu \mid \text{rest}) \propto \exp\left(-\mu^2\left(\frac{1}{2b_3} + \frac{K}{2V}\right) + \mu\left(\frac{a_3}{b_3} + \frac{1}{V}\sum_{i=1}^{K}\theta_i\right)\right).$$

Thus

$$\mu \mid \text{rest} \sim \mathcal{N}(m_\mu, v_\mu), \qquad v_\mu = \frac{b_3 V}{V + Kb_3}, \qquad \text{and} \qquad m_\mu = \frac{a_3 V + b_3\sum_{i=1}^{K}\theta_i}{V + Kb_3}. \tag{7.1}$$

The remaining full conditionals follow from the same calculation:

$$V \mid \text{rest} \sim \text{IG}\left(a_1 + \frac{K}{2}, b_1 + \frac{1}{2}\sum_{i=1}^{K}(\theta_i - \mu)^2\right), \tag{7.2}$$

$$W \mid \text{rest} \sim \text{IG}\left(a_2 + \frac{J_\bullet}{2}, b_2 + \frac{1}{2}\sum_{i=1}^{K}\sum_{j=1}^{J_i}(Y_{ij} - \theta_i)^2\right), \tag{7.3}$$

$$\theta_i \mid \text{rest} \sim \mathcal{N}(m_i, v_i), \tag{7.4}$$

$$v_i = \frac{VW}{W + J_i V} \qquad \text{and} \qquad m_i = \frac{W\mu + V\sum_{j=1}^{J_i} Y_{ij}}{W + J_i V}.$$

A systematic scan Gibbs iteration updates V using (7.2), then W using (7.3), then μ using (7.1), and finally each θ_i using (7.4). A random scan implementation instead selects one of the $K+3$ coordinates uniformly and updates only that coordinate. Roughly $K+3$ random scan updates correspond to one systematic sweep.

Algorithm 4 Systematic scan Gibbs sampler for the variance components model

Input: Initial values $V^{(0)}, W^{(0)}, \mu^{(0)}, \theta_1^{(0)}, \ldots, \theta_K^{(0)}$ and a run length M

1: **for** $n = 1, \ldots, M$ **do**
2: Draw $V^{(n)}$ from the conditional distribution in (7.2) using $\boldsymbol{\theta}^{(n-1)}$ and $\mu^{(n-1)}$.
3: Draw $W^{(n)}$ from the conditional distribution in (7.3) using $\boldsymbol{\theta}^{(n-1)}$.
4: Draw $\mu^{(n)}$ from the conditional distribution in (7.1) using $V^{(n)}$ and $\boldsymbol{\theta}^{(n-1)}$.
5: **for** $i = 1, \ldots, K$ **do**
6: Draw $\theta_i^{(n)}$ from the conditional distribution in (7.4) using $V^{(n)}, W^{(n)}$, and $\mu^{(n)}$.
7: **end for**
8: **end for**

Output: The Gibbs chain after M systematic sweeps

The following reproducible example simulates a small hierarchical data set and runs the systematic scan sampler.

```
set.seed(3431)

K <- 8
J <- 6
true_mu <- 1.5
true_V <- 1.2
```

```r
true_W <- 0.5

true_theta <- rnorm(K, true_mu, sqrt(true_V))
y <- matrix(
  rnorm(K * J, rep(true_theta, each = J), sqrt(true_W)),
  nrow = K,
  byrow = TRUE
)

a1 <- 2
b1 <- 1
a2 <- 2
b2 <- 1
a3 <- 0
b3 <- 10

iterations <- 2500
burn_in <- 1000
mu <- 0
V <- 1
W <- 1
theta <- rowMeans(y)
draws <- matrix(NA_real_, nrow = iterations, ncol = 3)
colnames(draws) <- c("mu", "V", "W")

rinvgamma <- function(shape, scale) {
  1 / rgamma(1, shape = shape, rate = scale)
}

for (iteration in seq_len(iterations)) {
  V <- rinvgamma(
    a1 + K / 2,
    b1 + sum((theta - mu)^2) / 2
  )
  W <- rinvgamma(
    a2 + K * J / 2,
    b2 + sum((y - theta)^2) / 2
  )

  mu_variance <- b3 * V / (V + K * b3)
  mu_mean <- (a3 * V + b3 * sum(theta)) / (V + K * b3)
  mu <- rnorm(1, mu_mean, sqrt(mu_variance))

  theta_variance <- V * W / (W + J * V)
  theta_mean <- (W * mu + V * rowSums(y)) / (W + J * V)
```

```
  theta <- rnorm(K, theta_mean, sqrt(theta_variance))

  draws[iteration, ] <- c(mu, V, W)
}

retained <- draws[(burn_in + 1):iterations, , drop = FALSE]
colMeans(retained)
```

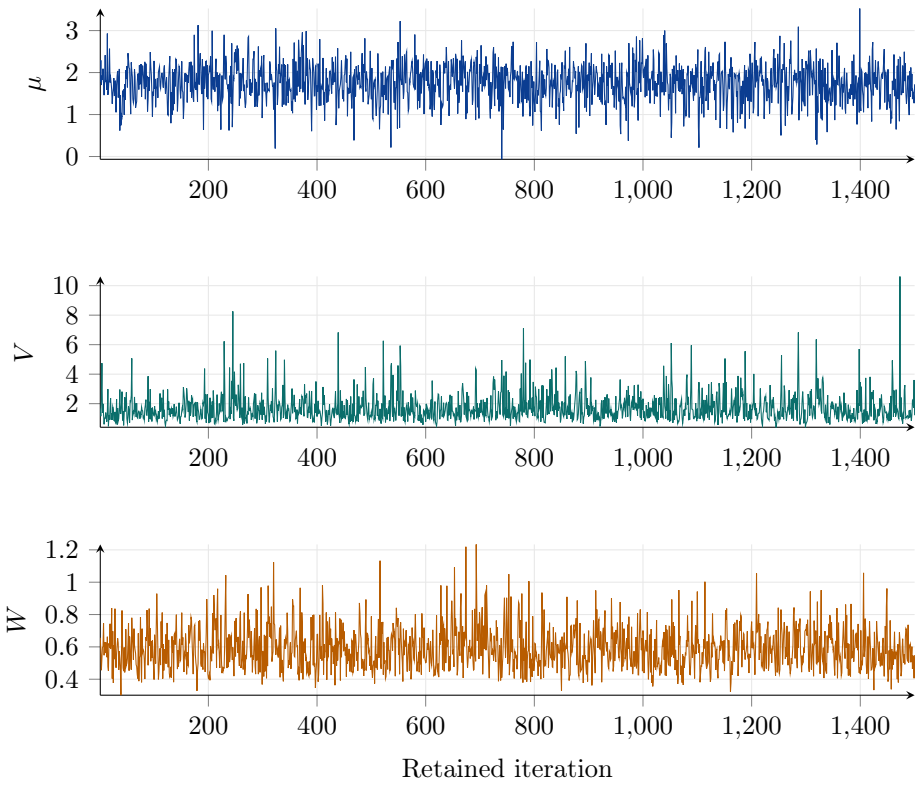

FIGURE 20

The three retained traces in Figure 20 fluctuate around stable regions. That visual check is useful but not conclusive; autocorrelation, multiple starting points, and problem specific diagnostics remain important.

Lecture 8 — Monday, November 11, 2019

5. Tempering, Optimization, and Convergence

Tempered and Parallel Chains

A random walk chain can appear to mix well within one mode while almost never visiting another. Trace and autocorrelation plots can then be misleading because they describe only the portion of the target that the chain has reached. Tempering addresses this failure by introducing flatter distributions through which the simulation can travel.

Suppose, for example, that

$$\Pi = \frac{1}{2}\mathcal{N}(0,1) + \frac{1}{2}\mathcal{N}(20,1).$$

A proposal $Y_n \sim \mathcal{N}(X_{n-1}, 1)$ explores either mode well but rarely crosses the low density gap. Define a family

$$\Pi_\tau = \frac{1}{2}\mathcal{N}(0,\tau^2) + \frac{1}{2}\mathcal{N}(20,\tau^2), \quad \tau = 1,\dots,m.$$

The distribution $\Pi_1 = \Pi$ is cold, while larger τ gives a hotter, flatter target.

Tempered Markov chain Monte Carlo builds a joint chain (X_n, T_n) on

$$\mathcal{X} \times \{1,\dots,m\}$$

with stationary distribution

$$\widetilde{\Pi}(S \times \{\tau\}) = \frac{1}{m}\Pi_\tau(S).$$

The temperatures may instead be assigned unequal fixed weights, provided the acceptance ratios are modified to target the resulting joint distribution. We may alternate between two kinds of Metropolis moves:

1. Propose $X' = x' \sim \mathcal{N}(x, 1)$ at the current temperature τ and accept with probability

$$\min\left\{1, \frac{\pi_\tau(x')}{\pi_\tau(x)}\right\}.$$

2. Use a symmetric nearest neighbour proposal for temperature, for example by proposing $\tau' = \tau + 1$ or $\tau - 1$ with equal probability and treating a proposal outside $\{1, \ldots, m\}$ as a self transition. Accept a valid neighbouring proposal with probability

$$\min\left\{1, \frac{\pi_{\tau'}(x)}{\pi_\tau(x)}\right\}.$$

After convergence, only observations recorded at $T_n = 1$ are counted toward expectations under the original target.

Usually the convenient family Π_τ is not known explicitly. A natural choice is

$$\pi_\tau(x) = c_\tau \pi(x)^{1/\tau},$$

which flattens the target as τ increases. If π is $\mathcal{N}(\mu, \sigma^2)$, then $\pi^{1/\tau}$, after normalization, is $\mathcal{N}(\mu, \tau\sigma^2)$. The temperature move ratio, however, contains

$$\frac{\pi_{\tau'}(x)}{\pi_\tau(x)} = \frac{c_{\tau'}}{c_\tau}\pi(x)^{1/\tau' - 1/\tau},$$

so unknown temperature dependent normalising constants make ordinary simulated tempering difficult.

Parallel tempering avoids those constants. Run m chains simultaneously, with state

$$\boldsymbol{X}_n = (X_{n,1}, \ldots, X_{n,m})$$

and product stationary distribution

$$\widetilde{\Pi} = \Pi_1 \times \cdots \times \Pi_m.$$

This method is also called *replica exchange* or *Metropolis coupled Markov chain Monte Carlo*. Replica exchange was introduced by Swendsen and Wang (1986), and the statistical formulation follows Geyer (1991). Each coordinate can be updated at its own temperature. In addition, choose temperatures τ and τ', propose to swap $X_{n,\tau}$ and $X_{n,\tau'}$, and accept with probability

$$\min\left\{1, \frac{\pi_\tau(X_{n,\tau'})\pi_{\tau'}(X_{n,\tau})}{\pi_\tau(X_{n,\tau})\pi_{\tau'}(X_{n,\tau'})}\right\}. \tag{8.1}$$

When $\pi_\tau(x) = c_\tau \pi(x)^{1/\tau}$, the constants c_τ and $c_{\tau'}$ cancel from (8.1). The hotter chains move readily between modes and can exchange those locations with the cold chain.

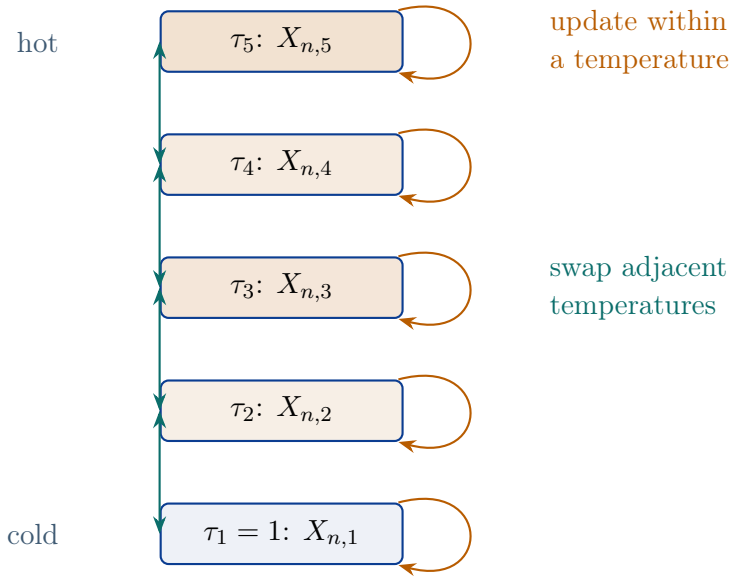

FIGURE 21

The vertical arrows in Figure 21 represent proposed exchanges between adjacent temperatures, while each loop represents an ordinary Metropolis–Hastings update at a fixed temperature. Information moves down the ladder when a state discovered by a hot chain is swapped toward the cold target.

Algorithm 5 Parallel tempering with adjacent swaps

Input: Initial states $X_{0,1}, \dots, X_{0,m}$, targets $\pi_1, \dots, \pi_m$, proposal kernels for the individual chains, and a run length M

1: **for** $n = 1, \dots, M$ **do**
2: **for** $\tau = 1, \dots, m$ **do**
3: Apply one Metropolis–Hastings update targeting π_τ to obtain $X_{n,\tau}$.
4: **end for**
5: Choose j uniformly from $\{1, \dots, m-1\}$ and propose to exchange $X_{n,j}$ with $X_{n,j+1}$.
6: Compute the swap acceptance probability in (8.1) with $\tau = j$ and $\tau' = j+1$.
7: Accept or reject the proposed exchange using an independent Unif$[0,1]$ draw.
8: **end for**

Output: The cold chain $X_{0,1}, X_{1,1}, \dots, X_{M,1}$, together with the auxiliary heated chains

The following R example compares a cold random walk with parallel tempering and produces Figure 22.

```r
mixture_density <- function(x, temperature) {
  0.5 * dnorm(x, 0, temperature) +
    0.5 * dnorm(x, 20, temperature)
}

rw_mixture <- function(iterations, proposal_sd = 1, initial = 0) {
  trace <- numeric(iterations)
  trace[1] <- initial

  for (iteration in 2:iterations) {
    proposal <- rnorm(1, trace[iteration - 1], proposal_sd)
    ratio <- mixture_density(proposal, 1) /
      mixture_density(trace[iteration - 1], 1)

    if (runif(1) < min(1, ratio)) {
      trace[iteration] <- proposal
    } else {
      trace[iteration] <- trace[iteration - 1]
    }
  }

  trace
}

parallel_tempering <- function(iterations, temperatures) {
  chain_count <- length(temperatures)
  states <- rep(0, chain_count)
  cold_trace <- numeric(iterations)
  cold_trace[1] <- states[1]

  for (iteration in 2:iterations) {
    for (index in seq_len(chain_count)) {
      proposal <- rnorm(1, states[index], 1)
      ratio <- mixture_density(proposal, temperatures[index]) /
        mixture_density(states[index], temperatures[index])

      if (runif(1) < min(1, ratio)) {
        states[index] <- proposal
      }
    }
```

```
    lower <- sample.int(chain_count - 1, 1)
    upper <- lower + 1
    swap_ratio <-
      mixture_density(states[upper], temperatures[lower]) *
      mixture_density(states[lower], temperatures[upper]) /
      (
        mixture_density(states[lower], temperatures[lower]) *
          mixture_density(states[upper], temperatures[upper])
      )

    if (runif(1) < min(1, swap_ratio)) {
      temporary <- states[lower]
      states[lower] <- states[upper]
      states[upper] <- temporary
    }

    cold_trace[iteration] <- states[1]
  }

  cold_trace
}

set.seed(3431)
plain_trace <- rw_mixture(4000)
parallel_trace <- parallel_tempering(
  4000,
  temperatures = c(1, 2, 4, 7, 10)
)
```

The upper panel of Figure 22 shows how increasing temperature bridges the two modes. The lower panels show that a seemingly calm cold trace can be trapped, while swaps allow the parallel tempering cold chain to visit both modes.

Simulated Annealing and Search Applications

Tempering can also be used for optimization rather than sampling. The mode of π is unchanged by taking a positive power. In the parametrization

$$\pi_\tau(x) \propto \pi(x)^{1/\tau},$$

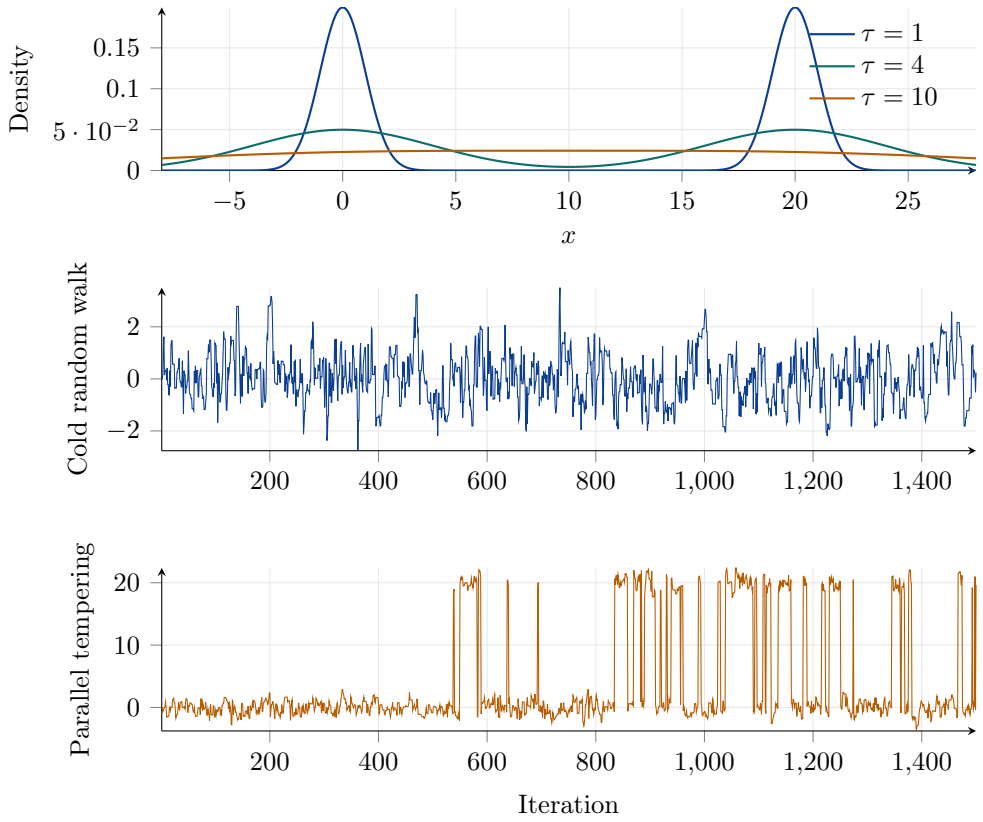

FIGURE 22

large τ produces a flat distribution that explores widely, while small τ concentrates around the modes. Simulated annealing uses a decreasing temperature sequence $\tau_n \downarrow 0$. The chain explores early and increasingly concentrates on high density states.

Common cooling schedules include

$$\tau_n = \tau_0 r^n \quad (0 < r < 1),$$

$$\tau_n = \tau_0 - dn \quad \text{over a finite run while this is positive,} \qquad \text{and} \qquad \tau_n = \frac{\tau_0}{\log(1+n)}.$$

The schedule must be slow enough to avoid trapping the chain at a suboptimal local mode.

In the standard finite state formulation, let

$$H(x) := -\log \pi(x)$$

and use the proposal graph to define paths between states. For a nonglobal local minimum x,

define its depth by

$$D(x) := \min_{\substack{\gamma: x \rightsquigarrow y \\ H(y) < H(x)}} \left\{ \max_{z \in \gamma} H(z) - H(x) \right\},$$

and let $d^\star$ be the largest of these depths. Thus $d^\star$ is the deepest energy barrier that must be crossed to escape a nonglobal local minimum.

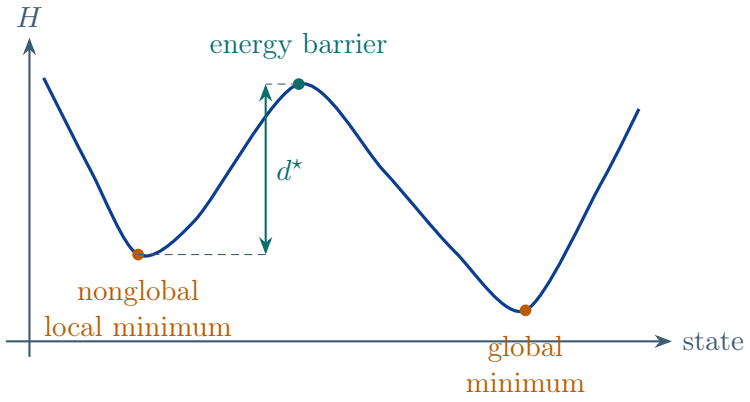

FIGURE 23

In Figure 23, a chain trapped near the nonglobal minimum must accept enough uphill motion to cross a barrier of height $d^\star$. A high temperature makes such moves plausible early in the run; the decreasing schedule then concentrates the chain near the global minimum.

Algorithm 6 Simulated annealing

Input: An initial state X_0, an energy function $H = -\log \pi$, a proposal kernel q, a decreasing temperature schedule $\tau_1, \ldots, \tau_M$, and a run length M

for $n = 1, \ldots, M$ **do**
  Generate $Y_n = y \sim q(X_{n-1}, \cdot)$ and $U_n \sim \text{Unif}[0,1]$ independently.
  Set
  $A_n \leftarrow \exp(-[H(y) - H(X_{n-1})]/\tau_n)\, q(y, X_{n-1})/q(X_{n-1}, y)$.
  if $U_n < \min\{1, A_n\}$ **then**
    Set $X_n \leftarrow y$.
  else
    Set $X_n \leftarrow X_{n-1}$.
  end if
end for

Output: The annealing path $X_0, X_1, \ldots, X_M$

Theorem 8.1 For an irreducible finite state simulated annealing chain satisfying the usual regularity conditions, if $c \geq d^\star$, then the cooling schedule

$$\tau_n = \frac{c}{\log(1+n)}$$

causes the chain to converge in probability to the set of global maximizers of π as $n \to \infty$.

The logarithmic schedule in Theorem 8.1 is theoretically reliable but very slow. The precise cooling criterion is due to Hajek, "Cooling schedules for optimal annealing" (1988). For

$$\Pi_\tau = 0.3\mathcal{N}(0, \tau^2) + 0.7\mathcal{N}(20, \tau^2),$$

the component near 20 has the larger mixture weight. At low temperatures the dominant mode lies close to 20; at sufficiently high temperatures the two components can merge into one broad mode between their centres. A fixed large temperature moves readily across the gap, whereas a fixed small temperature can become trapped near either component. Gradual cooling generally locates the dominant low-temperature mode. A more realistic implementation uses

$$\pi_\tau(x) \propto \{0.3\phi(x) + 0.7\phi(x-20)\}^{1/\tau},$$

which usually needs a longer run and a smaller final temperature.

The following R implementation compares fixed high and low temperatures with geometric cooling from 10 to 0.5. Its output appears in Figure 24.

```
annealing_density <- function(x, temperature) {
  0.3 * dnorm(x, 0, temperature) +
    0.7 * dnorm(x, 20, temperature)
}

annealing_run <- function(temperatures, proposal_sd, initial = 0) {
  iterations <- length(temperatures)
  trace <- numeric(iterations)
  trace[1] <- initial

  for (iteration in 2:iterations) {
    proposal <- rnorm(1, trace[iteration - 1], proposal_sd)
    ratio <- annealing_density(proposal, temperatures[iteration]) /
      annealing_density(trace[iteration - 1], temperatures[iteration])
```

```
    if (runif(1) < min(1, ratio)) {
      trace[iteration] <- proposal
    } else {
      trace[iteration] <- trace[iteration - 1]
    }
  }

  trace
}

set.seed(3431)
iterations <- 6000
high_temperature <- rep(10, iterations)
low_temperature <- rep(1, iterations)
cooling_temperature <- 10 * (0.05)^(
  (seq_len(iterations) - 1) / (iterations - 1)
)

high_trace <- annealing_run(high_temperature, proposal_sd = 4)
low_trace <- annealing_run(low_temperature, proposal_sd = 1)
set.seed(3431)
cooling_trace <- annealing_run(cooling_temperature, proposal_sd = 2.5)

c(
  high_final = tail(high_trace, 1),
  low_final = tail(low_trace, 1),
  cooling_final = tail(cooling_trace, 1)
)
```

At the fixed high temperature in Figure 24, the chain ranges widely and repeatedly crosses the gap between the two modes. At the fixed low temperature, the chain remains near the lower mode at 0. The cooling run explores broadly at first, then settles near the higher mode at 20 as its temperature approaches 0.5. A single successful run does not prove convergence, but it illustrates the exploration and concentration roles of the schedule.

Substitution cipher decoding provides a discrete optimization example. Let the encoded text be $s_1, \ldots, s_N$, with symbols in

$$\mathcal{A} = \{A, B, \ldots, Z, \text{space}\}.$$

The state space contains all bijections $f : \mathcal{A} \to \mathcal{A}$ that fix the space character. From a reference text, form

$$M(x, y) = 1 + \text{the number of times that } y \text{ follows } x.$$

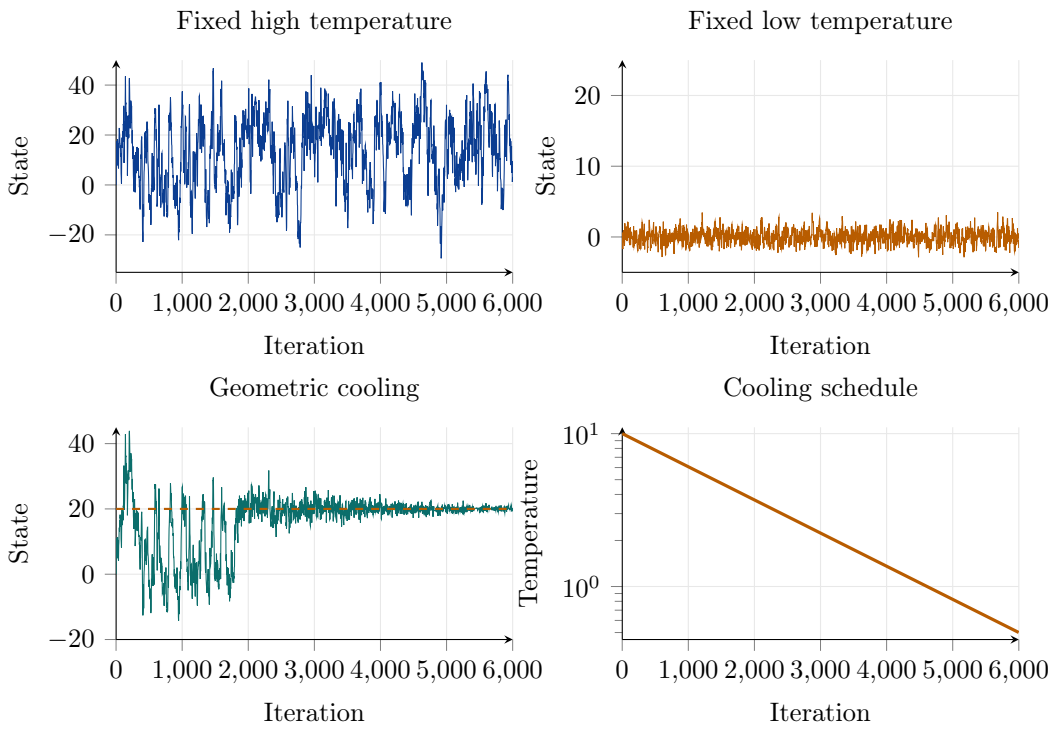

FIGURE 24

For a candidate decoding map f, define

$$\pi(f) = \prod_{i=1}^{N-1} M\big(f(s_i), f(s_{i+1})\big),$$

or use a fractional power such as $\pi(f)^{0.25}$. A larger score means that the decoded pair frequencies more closely resemble the reference text.

To search, retain the current map with probability $1/2$. Otherwise choose two distinct nonspace symbols a, b uniformly and propose a new bijection g that swaps $f(a)$ and $f(b)$, leaving every other image fixed. The proposal is symmetric, so accept a proposed swap with probability

$$\min\left\{1, \frac{\pi(g)}{\pi(f)}\right\}.$$

The swap moves make the chain irreducible, the explicit self transition makes it aperiodic, and symmetry gives reversibility. It rapidly concentrates on plausible decodings. The same idea extends beyond substitution ciphers to transposition ciphers. Further examples and analysis appear in Connor, "Simulation and solving substitution codes", Diaconis, "The Markov chain Monte Carlo revolution", and Chen and Rosenthal, "Decrypting classical cipher text using Markov chain Monte

Carlo".

Pattern detection in images gives a continuous search problem. Represent a possible face location by a parameter vector $\boldsymbol{\theta}$ containing the centre, eye separation, nose height, and other geometric features. A score $S(\boldsymbol{\theta})$ measures agreement between the image and the proposed face. A mixed search alternates local random walk moves with occasional fresh random proposals and records the best value seen. Because the objective is to maximize S, rather than sample from a prescribed target, the search moves need not preserve a stationary distribution. A detailed implementation and analysis are given by Rosenthal, "Optimising Monte Carlo search strategies for automated pattern detection".

Convergence Rates and Heavy Tails

For a Markov chain with stationary distribution Π, let

$$\mathbb{P}^n(x, A) := \mathbb{P}(X_n \in A \mid X_0 = x)$$

and define the total variation distance

$$D(x, n) := \|\mathbb{P}^n(x, \cdot) - \Pi(\cdot)\| := \sup_{A \text{ measurable}} |\mathbb{P}^n(x, A) - \Pi(A)|\,. \tag{8.2}$$

Definition 8.2 The chain is **ergodic** if $D(x, n) \to 0$ for almost every x with respect to Π. It is **geometrically ergodic** if there are a $\rho \in (0, 1)$ and a function $M : \mathcal{X} \to [0, \infty]$, finite almost everywhere with respect to Π, such that

$$D(x, n) \leq M(x)\rho^n$$

for every $n \in \mathbb{N}$. A **quantitative convergence bound** is a concrete value n^* such that, for example, $D(x, n^*) < 0.01$.

Quantitative bounds are often difficult to obtain, while geometric ergodicity can be easier to verify. As stated earlier, geometric ergodicity together with a $2 + \delta$ moment gives a central limit theorem; reversibility of the geometrically ergodic chain reduces the moment requirement to a second moment. Irreducibility, aperiodicity, and stationarity yield asymptotic ergodicity but do not themselves provide a useful rate. General quantitative bounds are surveyed by Rosenthal (2002) and Rosenthal (1995).

The independence sampler admits a particularly sharp criterion.

Theorem 8.3 An independence sampler with target density π and proposal density q is geometrically ergodic if and only if there is a $\delta > 0$ such that

$$q(x) \geq \delta\pi(x)$$

for almost every x with respect to Π. Under this condition,

$$D(x, n) \leq (1 - \delta)^n$$

for almost every x with respect to Π.

Return to the exponential example in Definition 6.2, with

$$\pi(x) = e^{-x} \qquad \text{and} \qquad q(x) = ke^{-kx}, \quad x > 0.$$

If $0 < k \leq 1$, then $q(x) \geq k\pi(x)$, so Theorem 8.3 applies with $\delta = k$. For $k = 0.01$,

$$D(x, 459) \leq 0.99^{459} = 0.0099 < 0.01.$$

If $k > 1$, no positive uniform minorization constant exists, and the sampler is not geometrically ergodic. If $k > 2$, the analysis of Roberts (1999) gives standard ergodic averages for which a central limit theorem fails; this does not mean that every bounded functional lacks a central limit theorem. When $k = 5$, convergence to total variation distance 0.01 takes between roughly four million and fourteen million iterations, explaining the severe empirical failure seen in Figure 18. The acceptance rate and convergence results for this example are developed by Roberts, "A note on acceptance rate criteria for CLTs for Metropolis–Hastings algorithms" (1999) and Roberts and Rosenthal (2011).

Every irreducible and aperiodic Markov chain on a finite state space is geometrically ergodic. For random walk Metropolis on an unbounded state space, tail behaviour is decisive. The results of Mengersen and Tweedie (1996) and Roberts and Tweedie (1996) show that exponentially light tails are necessary and, together with appropriate tail regularity, sufficient for geometric ergodicity. Exponential lightness means that there are $a, b, c > 0$ such that

$$\pi(x) \leq ae^{-b|x|}$$

whenever $|x| > c$. The usual conditions require positive continuous target and proposal densities, a proposal with finite first moment, and a target that decreases in the tails; higher dimensional versions also impose a curvature condition.

The consequences can be seen in four examples with proposals $Y_n \sim \mathcal{N}(X_{n-1}, \sigma^2)$.

1. If $\Pi = \mathcal{N}(5, 4^2)$ and $h(y) = y^2$, then $\mathbb{E}_\pi[h] = 41$. The target is exponentially light and has moments of every order, so a central limit theorem holds. Proposal scales 1, 4, and 16 give very different efficiencies, but properly estimated confidence intervals usually cover 41.

2. If
$$\pi(y) = \frac{c}{1 + y^4} \qquad \text{and} \qquad h(y) = y^2,$$
then
$$\mathbb{E}_\pi[h] = \frac{\int_{-\infty}^{\infty} y^2/(1 + y^4)\,\mathrm{d}y}{\int_{-\infty}^{\infty} 1/(1 + y^4)\,\mathrm{d}y} = 1.$$
The polynomial tails prevent geometric ergodicity; estimates are less stable, and confidence intervals often miss 1.

3. If $\pi(y) = 1/\{\pi(1 + y^2)\}$ is Cauchy and
$$h(y) = \mathbb{1}_{\{-10<y<10\}},$$
then
$$\mathbb{E}_\pi[h] = \mathbb{P}_\pi(|Y| < 10) = \frac{2\arctan(10)}{\pi} = 0.93655.$$
The chain is not geometrically ergodic, and nominal 95% confidence intervals often miss the target value.

4. For the same Cauchy target and
$$h(y) = \min\{y^2, 100\},$$
numerical integration gives $\mathbb{E}_\pi[h] \approx 11.77$. Random walk Metropolis intervals again often miss the target, while independent sampling performs better.

The following R experiment makes the contrast in these four cases concrete. It performs twenty four independent runs for each target, retains 20,000 observations after warmup, and estimates the standard error from twenty batch means. The resulting nominal 95% intervals appear in Figure 25.

```
rw_interval <- function(
    log_target,
    functional,
    truth,
    proposal_sd,
    initial,
    warmup = 2000,
    retained_count = 20000,
```

```
    batch_count = 20) {
  total <- warmup + retained_count
  state <- initial
  values <- numeric(retained_count)

  for (iteration in seq_len(total)) {
    proposal <- rnorm(1, state, proposal_sd)
    log_ratio <- log_target(proposal) - log_target(state)
    if (log(runif(1)) < log_ratio) {
      state <- proposal
    }
    if (iteration > warmup) {
      values[iteration - warmup] <- functional(state)
    }
  }

  batch_size <- retained_count / batch_count
  batch_means <- colMeans(matrix(values, nrow = batch_size))
  estimate <- mean(values)
  standard_error <- sd(batch_means) / sqrt(batch_count)
  interval <- estimate + c(-1, 1) * qnorm(0.975) * standard_error

  data.frame(
    estimate = estimate,
    lower = interval[1],
    upper = interval[2],
    truth = truth,
    covered = interval[1] <= truth && truth <= interval[2]
  )
}

cases <- list(
  normal = list(
    log_target = function(x) dnorm(x, 5, 4, log = TRUE),
    functional = function(x) x^2,
    truth = 41,
    proposal_sd = 4,
    initial = 5
  ),
  polynomial = list(
    log_target = function(x) -log1p(x^4),
    functional = function(x) x^2,
    truth = 1,
    proposal_sd = 1,
    initial = 0
```

```
  ),
  cauchy_indicator = list(
    log_target = function(x) dcauchy(x, log = TRUE),
    functional = function(x) as.numeric(abs(x) < 10),
    truth = 2 * atan(10) / pi,
    proposal_sd = 1,
    initial = 0
  ),
  cauchy_square = list(
    log_target = function(x) dcauchy(x, log = TRUE),
    functional = function(x) min(x^2, 100),
    truth = integrate(
      function(x) pmin(x^2, 100) * dcauchy(x),
      -Inf,
      Inf
    )$value,
    proposal_sd = 1,
    initial = 0
  )
)

set.seed(3431)
replication_count <- 24
interval_results <- lapply(cases, function(case) {
  result <- do.call(
    rbind,
    replicate(
      replication_count,
      do.call(rw_interval, case),
      simplify = FALSE
    )
  )
  result$run <- seq_len(replication_count)
  result
})

vapply(interval_results, function(result) mean(result$covered), numeric(1))
```

In this seeded experiment, twenty two of the twenty four normal intervals in Figure 25 cover the dashed target value. Coverage falls to ten of twenty four for the polynomial tailed target, eighteen of twenty four for the Cauchy indicator, and twelve of twenty four for the truncated square. These counts are themselves random, but the repeated misses make the instability visible.

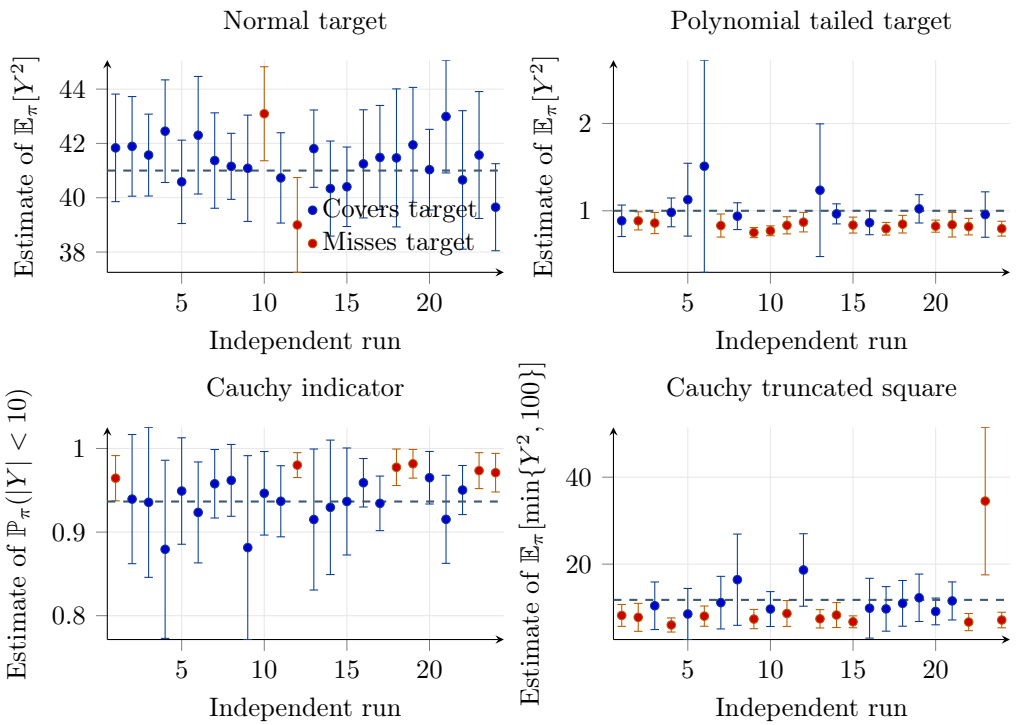

Figure 25

Even when a central limit theorem holds, the uncertainty estimate can be unreliable if the chain has not converged or if the variance factor in (4.2) is poorly estimated. Repeated independent runs provide one check. Another method divides a long chain into several large batches and treats the batch means as approximately independent.

Proposal Shape and Adaptive MCMC

For a d-dimensional random walk Metropolis algorithm with proposal

$$Y_n \sim \mathcal{N}(X_{n-1}, \boldsymbol{\Sigma}),$$

the covariance shape can matter as much as the overall scale. Suppose $\Pi = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ in dimension 5. Representative experiments give:

1. With proposal covariance $\mathbf{I}$, the acceptance rate is about 0.06 and the variance factor is about 220.
2. With $0.1\mathbf{I}$, the acceptance rate is about 0.234, but the variance factor is about 300.

3. With $\boldsymbol{\Sigma}_0$, the acceptance rate is about 0.31 and the variance factor is about 15.
4. With $1.4\boldsymbol{\Sigma}_0$, the acceptance rate is about 0.234 and the variance factor is about 7.

In dimension 20, a proposal covariance $0.025\mathbf{I}$ can achieve acceptance near 0.234 but have variance factor above 400, while $0.283\boldsymbol{\Sigma}_0$ has the same acceptance rate and variance factor near 50.

The numerical comparisons are recorded and plotted by the following R code.

```
proposal_experiments <- data.frame(
  dimension = c(5, 5, 5, 5, 20, 20),
  shape = c(
    "spherical",
    "spherical",
    "target shaped",
    "target shaped",
    "spherical",
    "target shaped"
  ),
  acceptance_rate = c(0.06, 0.234, 0.31, 0.234, 0.234, 0.234),
  variance_factor = c(220, 300, 15, 7, 400, 50)
)

proposal_experiments
```

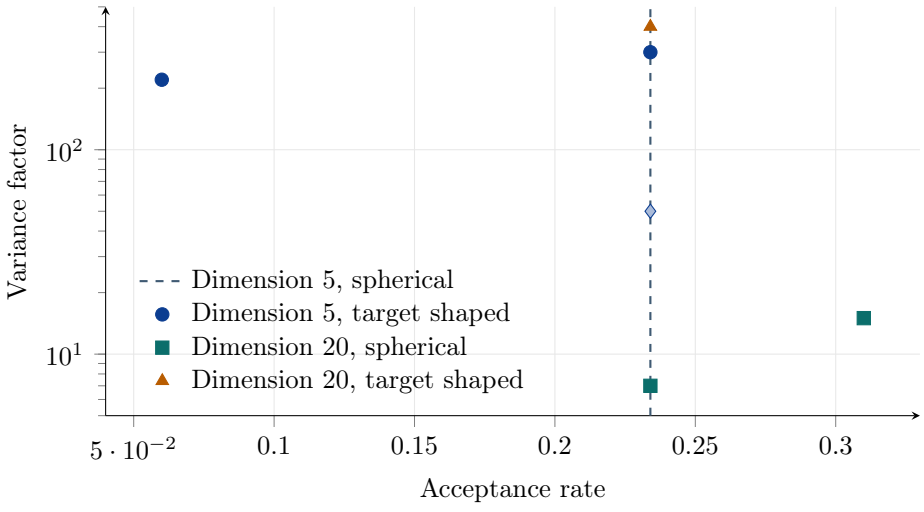

FIGURE 26

The vertical dashed line in Figure 26 isolates four experiments with acceptance rate near 0.234. Their variance factors range from about 7 to above 400, so acceptance alone cannot distinguish a well shaped proposal from a poorly shaped one.

Thus an acceptance rate near 0.234 does not by itself guarantee efficiency. The proposal should also resemble the target's shape. In a broad high dimensional class, the asymptotically optimal covariance is

$$\boldsymbol{\Sigma}_{\text{proposal}} = \frac{(2.38)^2}{d}\boldsymbol{\Sigma}_0. \tag{8.3}$$

This is an asymptotic optimal-scaling result for product targets and their appropriate linear transformations, not an exact finite-dimensional rule. It is often approximately useful for regular unimodal targets.

The target covariance $\boldsymbol{\Sigma}_0$ is generally unknown. Adaptive Markov chain Monte Carlo estimates it using the history of the chain. Let $\boldsymbol{\Sigma}_n$ be the empirical covariance after n iterations. A natural proposal is

$$Y_n \sim \mathcal{N}\left(X_{n-1}, \frac{(2.38)^2}{d}\boldsymbol{\Sigma}_n + \varepsilon \mathbf{I}\right), \tag{8.4}$$

where a small $\varepsilon > 0$, such as 0.05, improves numerical stability. Adaptation can require a much longer warmup. In a five dimensional normal example, discarding the first 20,000 iterations as warmup produces a variance factor near 18 over the next 4,000 draws, competitive with a well tuned nonadaptive chain.

One diagnostic is the slowdown factor

$$s_n := d\,\frac{\sum_{i=1}^{d} \lambda_{i,n}^{-2}}{\left(\sum_{i=1}^{d} \lambda_{i,n}^{-1}\right)^2},$$

where $\lambda_{i,n}$ are the eigenvalues of

$$\boldsymbol{\Sigma}_n^{1/2}\boldsymbol{\Sigma}_0^{-1/2}.$$

If $\boldsymbol{\Sigma}_n = \boldsymbol{\Sigma}_0$, every eigenvalue is 1 and $s_n = 1$. In dimensions as high as 200, adaptation can work well but may require many iterations before improving the proposal substantially.

Adaptation changes the transition rule over time. The process is generally no longer Markovian and does not preserve stationarity at every step. It can therefore fail to converge without additional conditions.

Theorem 8.4 Suppose that every fixed kernel $\mathbb{P}_\gamma$ in the adaptive family has the same stationary target Π. The adaptive algorithm is guaranteed to converge to Π under the following two conditions.

1. **Diminishing adaptation:** if Γ_n denotes the tuning parameter at time n, then
$$\sup_{x \in \mathcal{X}} \left\| \mathbb{P}_{\Gamma_{n+1}}(x, \cdot) - \mathbb{P}_{\Gamma_n}(x, \cdot) \right\| \longrightarrow 0$$
in probability.
2. **Containment:** for every $\varepsilon > 0$, the time required for the fixed kernel $\mathbb{P}_{\Gamma_n}$, started from X_n, to come within ε of stationarity remains bounded in probability as $n \to \infty$.

Condition (1) can be enforced by the user because it requires successively smaller changes to the tuning rule. Condition (2) prevents the algorithm from escaping into regions where its current kernel would converge arbitrarily slowly. Compact state and adaptation spaces provide one sufficient setting, and more refined checkable conditions are available. Together the two conditions also yield a weak law of large numbers for bounded functionals; stronger hypotheses give additional laws of large numbers and central limit theorems.

Adaptive methods can therefore learn an effective proposal while sampling, but the adaptation must be controlled. The same caution that applies to every Markov chain Monte Carlo diagnostic applies here with additional force: apparent motion is not by itself a proof of convergence. Extensions of optimal scaling include Bédard (2007). Adaptive covariance examples and convergence conditions appear in Roberts and Rosenthal (2009), Roberts and Rosenthal (2007), Bai, Roberts, and Rosenthal (2011), and Craiu, Gray, Łatuszyński, Madras, Roberts, and Rosenthal (2015).

Appendix: Lecture and Tutorial Index