How to Estimate 1 and Other Interesting Quantities[1]

Mini-Project for STA3431
Robert Zimmerman
Second-Year PhD Student, Department of Statistical Sciences
1005054600

In applied statistics, we often need to generate random numbers in order to perform simulations of complicated systems whose solutions are hard to compute explicitly. For example, MCMC-based methods draw random variables from pre-specified distributions, producing Markov chains with desirable asymptotic properties. True randomness is usually unobtainable outside of physical methods (such as generating random bits based on radioactive decay), but a wide variety of methods known as pseudo-random number generators (PRNGs) have been designed to output sequences of numbers on a computer that appear to be random. In the setup typically used for statistical purposes, PRNGs output a sequence of numbers which appear to follow a Uniform$[0, 1]$ distribution. The past few decades have seen a growth of interest in randomness tests such as the Marsaglia Diehard tests ([**MT+02**]), which attempt in various ways to test sequences of PRNG outputs for randomness. Some of these tests measure the complexity and are grounded in information theory, while others are based on statistical tests.

During the course of a previous assignment, we tested a PRNG of our choice using several tests including the Wald–Wolfowitz runs test, Shapiro-Wilk test, and Anderson-Darling test. While the distributions of the test statistics of the first two under the null hypothesis have easy closed forms, a literature search revealed no such form for the Anderson-Darling test. However, despite this, Anderson and Darling were able to derive an asymptotic distribution for the test statistic in [**AD+52**], which we present later. We wondered how it might be possible to sample from it, since direct sampling using the inverse cdf method is completely out of the question. Thus, this distribution is a good candidate to apply MCMC algorithms, which are typically used to sample from complicated distributions.

The Anderson-Darling test is a goodness-of-fit test that examines whether a set of (ordered) data arises from an arbitrary specified distribution F. More specifically, the test statistic

$$A_n = n \int_{-\infty}^{\infty} \frac{(F_n(x) - F(x))^2}{F(x)(1 - F(x))}\, dx$$

is computed and compared against the theoretical distribution of A_n ([**AD54**]), where $F_n(x) = \frac{1}{n}\sum_{j=1}^{n} \mathbf{1}_{x_i \le x}$ is the empirical distribution function, and the data are sorted in increasing order (with no repeated values). In the particular case of the uniform distribution, $F(x) = x \cdot \mathbf{1}_{x\in[0,1]} + \mathbf{1}_{x>1}$, and under the null hypothesis that $x_1, \ldots, x_n \overset{iid}{\sim} \text{Uniform}[0, 1]$, the test statistic reduces to the well-known[2]

$$A_n = n \int_{-\infty}^{\infty} \frac{(F_n(x) - x)^2}{x(1 - x)}\, dx = -n - \frac{1}{n}\sum_{j=1}^{n} (2j - 1) \log(x_j \cdot (1 - x_j)).$$

What *is* the distribution of A_n? For $n = 1$, simple algebra shows that

$$\mathbb{P}(A_1 < z) = \sqrt{1 - 4e^{-1-z}} \text{ for } z > \log(4) - 1.$$

As mentioned above, a closed form for the distribution of A_n for larger (but finite) values of n is not known. Remarkably, however, the asymptotic distribution of A_n as $n \to \infty$ was shown by Anderson and

[1] While the objective of this mini-project (which we hope we have fulfilled) was to find an interesting and challenging quantity to compute, we recognize that the quantity 1 is neither challenging to compute nor particularly interesting. A more descriptive–but less provocative–title might be "How to Estimate the Expectation of Functionals with Respect to the Limiting Distribution of the Anderson-Darling Test Statistic for Uniformity Under the Null Hypothesis".

[2] In [**AD+52**], to which we refer below, the authors develop their asymptotic theory based on the more general statistic $n\int_{-\infty}^{\infty}(F_n(x) - F(x))^2 \cdot \psi(F(x))\, dx$, where $\psi(\cdot)$ is some non-negative weighting function. Taking $\psi \equiv 1$, for example, recovers test statistic for the Cramèr-von Mises criterion, which is often used as an alternative to the Kolmogorov-Smirnoff test.

Darling in [**AD+52**] to be

$$\lim_{n\to\infty} \mathbb{P}(A_n < z) = \frac{\sqrt{2\pi}}{z} \sum_{j\geq 0} \binom{-\frac{1}{2}}{j} (4j+1) e^{-\frac{(4j+1)^2\pi^2}{8z}} \int_0^\infty e^{\frac{z}{8(1+w^2)} - \frac{w^2(4j+1)^2\pi^2}{8z}}\, dw. \tag{1}$$

In the sequel, we denote this cdf as F_∞, and in a slight abuse of notation, we write A_∞ for an arbitrary random variable distributed as F_∞. With a sample $X_1, \ldots, X_M \overset{iid}{\sim} F_\infty$, we can estimate expectations of functionals with respect to F_∞ using the standard Monte Carlo estimate $\frac{1}{M}\sum_{j=1}^M h(X_j)$. In particular, we can estimate the expectation of $X \sim F_\infty$ by $\mathbb{E}[X] \approx \frac{1}{M}\sum_{j=1}^M X_j$ and the variance by $\mathrm{Var}(X) \approx \frac{1}{M}\sum_{j=1}^M X_j^2 - (\frac{1}{M}\sum_{j=1}^M X_j)^2$. Estimating these quantities allows us to evaluate the quality of our sampling methods, because their true values are known; moreover, they are extremely simple quantities, as we demonstrate below.

In [**AD+52**], Anderson and Darling showed that for the particular Gaussian process $x(t) = \sum_{j\geq 1} Z_j \frac{f_j(t)}{\sqrt{j(j+1)}}$ where $Z_1, Z_2, \ldots \overset{iid}{\sim} \mathcal{N}(0,1)$ and $\{f_j(t)\}_{j\geq 1}$ is orthonormal on $[0,1]$, it is possible to write[3]

$$A_\infty = \int_0^1 z^2(t)\, dt = \sum_{j\geq 1} \frac{Z_j^2}{j(j+1)}. \tag{2}$$

Thus, the monotone convergence theorem yields

$$\mathbb{E}[A_\infty] = \mathbb{E}\left[\sum_{j\geq 1} \frac{Z_j^2}{j(j+1)}\right] = \sum_{j\geq 1} \frac{\mathbb{E}\left[Z_j^2\right]}{j(j+1)} = \sum_{j\geq 1} \frac{1}{j(j+1)} = \sum_{j\geq 1} \left(\frac{1}{j-1} - \frac{1}{j}\right) = 1.$$

Furthermore,

$$\begin{aligned}
\mathbb{E}\left[A_\infty^2\right] &= \mathbb{E}\left[\sum_{j\geq 1} \frac{Z_j^4}{j^2(j+1)^2} + 2\sum_{j\geq 1}\sum_{i<j} \frac{X_i^2 X_j^2}{i(i+1)j(j+1)}\right] \\
&= \sum_{j\geq 1} \frac{\mathbb{E}\left[Z_j^4\right]}{j^2(j+1)^2} + 2\sum_{j\geq 1}\sum_{i<j} \frac{\mathbb{E}\left[X_i^2\right]\mathbb{E}\left[X_j^2\right]}{i(i+1)j(j+1)} \\
&= \sum_{j\geq 1} \frac{3}{j^2(j+1)^2} + 2\sum_{j\geq 1} \frac{1}{j(j+1)} \sum_{i<j} \frac{1}{i(i+1)} \\
&= \sum_{j\geq 1} \frac{3}{j^2(j+1)^2} + 2\sum_{j\geq 1} \frac{j-1}{j^2(j+1)} \\
&= (\pi^2 - 9) + 2\left(2 - \frac{\pi^2}{6}\right) \\
&= \frac{2\pi^2}{3} - 5.
\end{aligned}$$

Again, we have appealed to the monotone convergence theorem, and then manipulated the sums using simple fractions. Thus, $\mathrm{Var}(A_\infty) = \mathbb{E}\left[A_\infty^2\right] - \mathbb{E}[A_\infty]^2 = \frac{2\pi^2}{3} - 6 = 0.57973\ldots$.

Now, all of the methods of sampling from distributions that we have learned require us to know their *densities* (up to a normalizing constant), rather than their cdfs. Marsaglia in [**Mar04**] made an important contribution by deriving an efficient method to calculate the theoretical cdfs of A_n and A_∞. We mimic his approach and derive a similar method to calculate the limiting density f_∞.

[3] Their development of this result, which essentially uses functional analysis and Donsker's theorem, also leads to the functional form of F_∞. After proving Equation 2, they derive the moment generating function of $\sum_{j\geq 1} \frac{Z_j^2}{j(j+1)}$, and use the complex inversion formula from the theory of Laplace transforms to recover F_∞.

To begin with, Marsaglia rewrote Equation 1 as

$$F_\infty(z) = \frac{1}{z}\sum_{j\geq 0}\binom{-\frac{1}{2}}{j}(4j+1)f(z,j),$$

where

$$\begin{aligned} f(z,j) &= \sqrt{2\pi}e^{-t_j}\int_0^\infty e^{\frac{z}{8(1+w^2)}-w^2t_j}\,dw, && t_j = \frac{(4j+1)^2\pi^2}{8z} \\ &= \sqrt{2\pi}e^{-t_j}\int_0^\infty \sum_{n\geq 0}\frac{z^n e^{-w^2t_j}}{8^n(1+w^2)^n n!}\,dw \\ &= \sum_{n\geq 0}\frac{d_{j,n}(z)\cdot z^n}{8^n n!}, && d_{j,n}(z) = \sqrt{2\pi}e^{-t_j}\int_0^\infty \frac{e^{-w^2t_j}}{(1+w^2)^n}\,dw \end{aligned}$$

Marsaglia provided the initial values $d_{j,0}(z) = \pi e^{-t_j}/\sqrt{2t_j}$ and $d_{j,1}(z) = \pi\sqrt{\pi/2}\cdot\operatorname{erfc}(\sqrt{t_j})$. His insight was that the functions $\{d_{j,n}(z)\}_{n\geq 2}$ satisfy the recursion

$$d_{j,n+1}(z) = \frac{(n-\frac{1}{2}-t_j)\cdot d_{j,n}(z) + t_j\cdot d_{j,n-1}(z)}{n},$$

which makes calculating F_∞ easy; the terms $\frac{d_{j,n}(z)\cdot z^n}{8^n n!}$ rapidly approach 0 as n grows, so $f(z,j)$ can be approximated by only a few of those terms without loss of precision.

It is now up to us to calculate the derivative of F_∞. To begin with, the product rule and yet another application of the monotone convergence theorem (respectively) yield

$$\begin{aligned} f_\infty(z) = \frac{dF_\infty(z)}{dz} &= -\frac{1}{z^2}\sum_{j\geq 0}\binom{-\frac{1}{2}}{j}(4j+1)f(z,j) + \frac{1}{z}\frac{d}{dz}\sum_{j\geq 0}\binom{-\frac{1}{2}}{j}(4j+1)f(z,j) \\ &= -\frac{1}{z^2}\sum_{j\geq 0}\binom{-\frac{1}{2}}{j}(4j+1)f(z,j) + \frac{1}{z}\sum_{j\geq 0}\binom{-\frac{1}{2}}{j}(4j+1)\frac{\partial f(z,j)}{\partial z} \\ &= \sum_{j\geq 0}\binom{-\frac{1}{2}}{j}(4j+1)\left(\frac{1}{z}\cdot\frac{\partial f(z,j)}{\partial z} - \frac{f(z,j)}{z^2}\right). \end{aligned}$$

Now,

$$\frac{\partial f(z,j)}{\partial z} = \sum_{n\geq 0}\frac{d}{dz}\left(\frac{d_n z^n}{8^n n!}\right) = \sum_{n\geq 0}\frac{d'_{j,n}(z)\cdot z^n + d_{j,n}(z)\cdot nz^{n-1}}{8^n n!}.$$

With $\{d_{j,n}(z)\}_n$ already available and

$$d'_{j,0}(z) = \frac{\mathrm{d}d_{j,n}(z)}{\mathrm{d}z} = \frac{\pi e^{-t_j}}{\sqrt{2}z}\left(\sqrt{t_j} + \frac{1}{2\sqrt{t_j}}\right) \qquad \text{and} \qquad d'_{j,1}(z) = \frac{\mathrm{d}d_{j,n}(z)}{\mathrm{d}z} = \frac{\pi e^{-t}\sqrt{t}}{\sqrt{2}z},$$

we can identify a new recursion by differentiating Marsaglia's recursion:

$$d'_{j,n+1}(z) = \frac{(n-\frac{1}{2}-t_j)\cdot d'_{j,n}(z) + t'_j\cdot(d_{j,n-1}(z) - d_{j,n}(z)) + t_j\cdot d'_{j,n-1}(z)}{n}, \qquad t'_j = -\frac{(4j+1)^2\pi^2}{8z^2}.$$

Thus,

$$\begin{aligned} f_\infty(z) &= \sum_{j\geq 0}\binom{-\frac{1}{2}}{j}(4j+1)\left(\frac{1}{z}\sum_{n\geq 0}\frac{d'_{j,n}(z)\cdot z^n + d_{j,n}(z)\cdot nz^{n-1}}{8^n n!} - \frac{1}{z^2}\sum_{n\geq 0}\frac{d_{j,n}(z)\cdot z^n}{8^n n!}\right) \\ &= \sum_{j\geq 0}\binom{-\frac{1}{2}}{j}(4j+1)\left(\sum_{n\geq 0}\frac{d'_{j,n}(z)\cdot z^{n-1} + d_{j,n}(z)\cdot(n-1)z^{n-2}}{8^n n!}\right). \end{aligned}$$

We have implemented this function in R efficiently; while the calculations of the functions $\{d_n\}_n$ and $\{d'_n\}_n$ do require loops due to their recursive definitions, we have computed the remainder of $f_\infty(z)$ purely through vectorization, without using any loops. We have plotted f_∞ in Figure 1 below using our implementation. We note that these plots are visually indistinguishable from Marsaglia's plots of the same function in [**Mar04**], which gives us some assurance that our implementation is correct.[4] We observe that f_∞ is smooth and unimodal, but the mode (at around 1) is quite peaked.

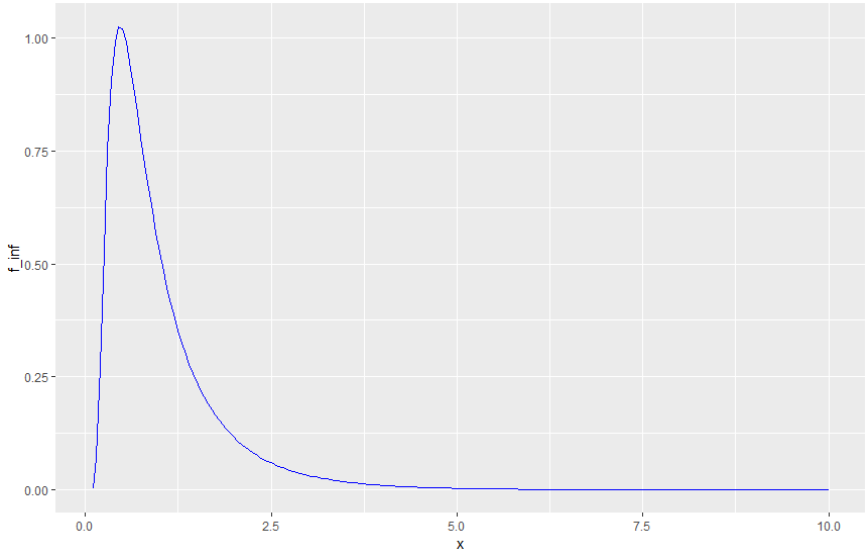


Figure 1: Plot of f_∞ for $x \in (0, 10)$

With f_∞ defined, we can finally start simulating from it. Our goal is to obtain estimates $\hat{I}$ and $\hat{V}$ of $\mathbb{E}[A_\infty] = 1$ and $\mathrm{Var}(A_\infty) = \frac{2\pi^3}{3} - 6$, respectively. Because the functional form of f_∞ is complicated, direct sampling methods such as rejection sampling (which relies on bounding f_∞ above by another density) or importance sampling (which relies on writing $f_\infty = \frac{f_\infty}{g} g$ for some easily-sampled density g with $\mathrm{supp}(g) \supseteq (0, \infty)$) are out of reach. An auxiliary variable sampler, which requires f_∞ to have bounded support, is also unusable. Thus, we rely on MCMC methods. In particular, we use a random-walk Metropolis algorithm, a Metropolis-Hastings algorithm, and parallel tempering. While the latter method is well-suited to multimodal densities, it is still interesting to observe its performance on f_∞.

To implement the random-walk Metropolis algorithm, we pick an initial value $X_0 \sim |\mathcal{N}(0, 1)|$, and proceed by simulating $Y_n \sim \mathcal{N}(X_{n-1}, \sigma^2)$, generating $U_n \sim \mathrm{Uniform}[0, 1]$, and accepting $X_n = Y_n$ when $U_n < \min\left(1, \frac{f_\infty(Y_n)}{f_\infty(X_{n-1})}\right)$. Taking $M = 5000$ and $B = 1000$, we obtain $\hat{I} = 0.9602$ with an approximate 95% confidence interval $(0.8699, 1.0504)$, and $\hat{V} = 0.5694$ with an approximate 95% confidence interval $(0.1870, 0.9517)$. Surprisingly, despite the width of the latter interval, our estimate of the variance is closer to the true value than our estimate of the mean is: $|\hat{V} - \frac{2\pi^2}{3} + 6| \approx 0.01$ while $|\hat{I} - 1| \approx 0.04$. Nevertheless, both estimates are very accurate, and the true values fall well within our confidence intervals. The trace plots and autocorrelation plots, shown in Figure 2 in Appendix A, reveal very little autocorrelation and what appears to be quite good mixing.

[4] Marsaglia included a supplementary file (which we did not view), that calculated F_∞ using the C language. Completely apart from the fact that we know almost no C, we find it much more rewarding to program these functions on our own from scratch.

Next, we use a Metropolis-Hastings algorithm with (very) asymmetric proposal density Lognormal$(x, \sigma^2 x)$. That is, our proposal density is

$$q(x,y) = \frac{1}{xy\sigma\sqrt{2\pi}} \exp\left(-\frac{(\log y - x)^2}{2\sigma^2 x}\right).$$

We have chosen this distribution because the log-normal density very roughly has a shape similar to that of f_∞, the two densities share the same support, and simply because it is unusual (and yet satisfies the requirements of a Metropolis-Hastings proposal distribution).

To implement the Metropolis-Hastings algorithm, we again pick an initial value $X_0 \sim |\mathcal{N}(0,1)|$, and proceed by simulating $Y_n \sim \text{Lognormal}(X_{n-1}, \sigma^2 X_{n-1})$, generating $U_n \sim \text{Uniform}[0,1]$, and accepting $X_n = Y_n$ when $U_n < \min\left(1, \frac{f_\infty(Y_n)\cdot q(Y_n, X_{n-1})}{f_\infty(X_{n-1})\cdot q(X_{n-1}, Y_n)}\right)$. Again taking $M = 5000$ and $B = 1000$, we obtain $\hat{I} = 0.9798$ with an approximate 95% confidence interval $(0.9032, 1.0563)$, and $\hat{V} = 0.5056$ with an approximate 95% confidence interval $(0.1988, 0.8123)$. Our estimates are in line with those produced by the random-walk Metropolis algorithm above (here $\hat{I}$ is slightly more accurate, while $\hat{V}$ is slightly less so), and our confidence intervals are in fact slightly narrower. The trace plots and autocorrelation plots, shown in Figure 3 in Appendix A, exhibit roughly the same features as those generated previously (with perhaps slightly better mixing).

To implement parallel tempering, we first define the range of "temperatures" $\tau_j = j$, for $j = 1, \ldots, m$. After picking initial values $X_{01}, \ldots, X_{0m} \overset{iid}{\sim} |\mathcal{N}(0,1)|$, we run m chains in parallel, where we alternate between the following two steps:

1. For each τ, simulate $Y_{n,\tau} \sim \mathcal{N}(X_{n-1,\tau}, \sigma^2)$, generating $U_{n,\tau} \sim \text{Uniform}[0,1]$ and accept $X_{n,\tau} = Y_{n,\tau}$ when $U_{n,\tau} < \min\left(1, \frac{f_\infty(Y_{n,\tau})}{f_\infty(X_{n-1,\tau})}\right)$
2. Choose τ and τ' at random, generate $U_n \sim \text{Uniform}[0,1]$, and swap $X_{n,\tau}$ and $X_{n,\tau'}$ when $U_n < \min\left(1, \frac{f_\infty(X_{n,\tau'})^{1/\tau}\cdot f_\infty(X_{n,\tau})^{1/\tau'}}{f_\infty(X_{n,\tau})^{1/\tau}\cdot f_\infty(X_{n,\tau'})^{1/\tau'}}\right)$

Due to the "for" loops required to calculate $f_\infty(z)$, our choices of m and M are necessarily restricted to allow for a practical running time). Thus we choose $m = 10$ and take $M = 1000$ and $B = 200$. With these, we obtain the estimates

$$\hat{I} = \begin{bmatrix} 0.8837 \\ 1.0636 \\ 1.3037 \\ 1.2170 \\ 1.1733 \\ 1.1803 \\ 1.2995 \\ 1.3014 \\ 0.9079 \\ 0.2789 \end{bmatrix} \quad \text{and} \quad \hat{V} = \begin{bmatrix} 0.2629 \\ 0.2646 \\ 0.3283 \\ 0.3332 \\ 0.3181 \\ 0.2792 \\ 0.3442 \\ 0.3199 \\ 0.4271 \\ 0.3512 \end{bmatrix}.$$

Interestingly, the best estimate of $\mathbb{E}[A_\infty]$ comes from the chain corresponding to $\tau = 2$ rather than $\tau = 1$ (as expected), for which the estimate is much worse. On the other hand, the best estimate of $\text{Var}(A_\infty)$ comes from the chain corresponding to $\tau = 9$ (although it is still not very good). The trace plots and autocorrelation plots are shown for each chain in Figure 4. While most of the plots look very similar to each other, we do note that both the trace plot and the autocorrelation plot corresponding to $\tau = 10$ look quite poor, which is in agreement with the estimate $\hat{I}$ produced by that chain.

References

[**AD+52**] Theodore W Anderson, Donald A Darling, et al. "Asymptotic theory of certain "goodness of fit" criteria based on stochastic processes". In: *The annals of mathematical statistics* 23.2 (1952), pp. 193–212.

[**AD54**] Theodore W Anderson and Donald A Darling. "A test of goodness of fit". In: *Journal of the American statistical association* 49.268 (1954), pp. 765–769.

[**MT+02**] George Marsaglia, Wai Wan Tsang, et al. "Some difficult-to-pass tests of randomness". In: *Journal of Statistical Software* 7.3 (2002), pp. 1–9.

[**Mar04**] George Marsaglia. "Evaluating the Anderson-Darling Distribution". In: *Journal of Statistical Software, Articles* 9.2 (2004), pp. 1–5. ISSN: 1548-7660. DOI: 10.18637/jss.v009.i02. URL: https://www.jstatsoft.org/v009/i02.

Appendix A: Additional Plots

Figure 2: Diagnostic plots for random-walk Metropolis

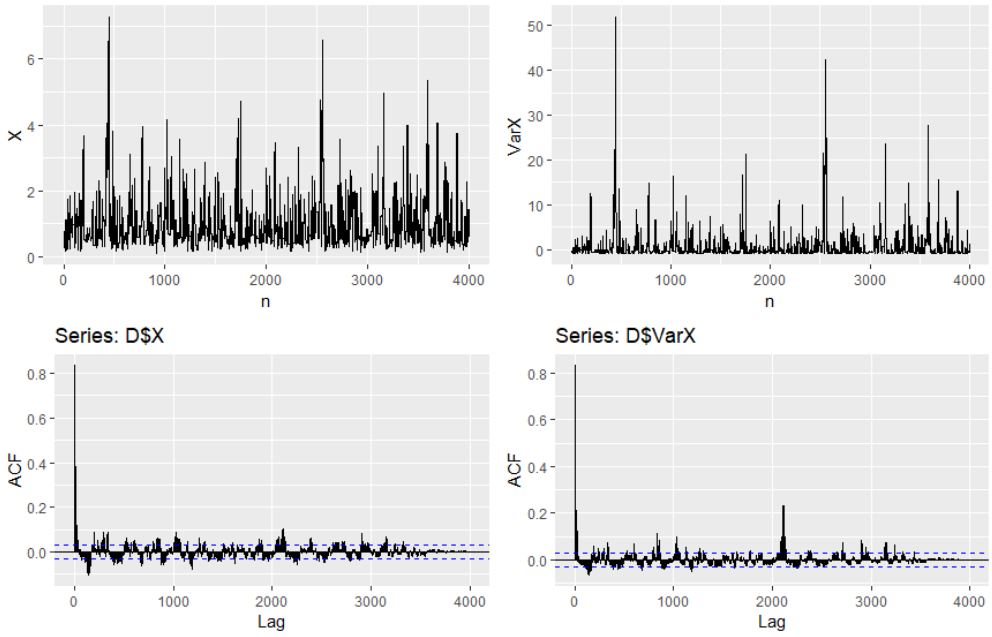


Figure 3: Diagnostic plots for Metropolis-Hastings

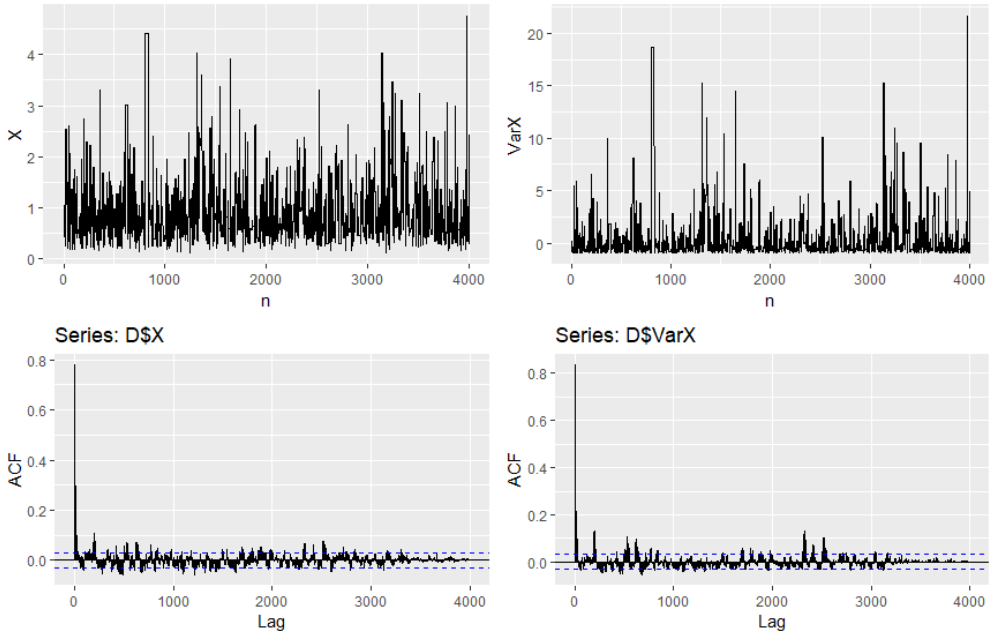
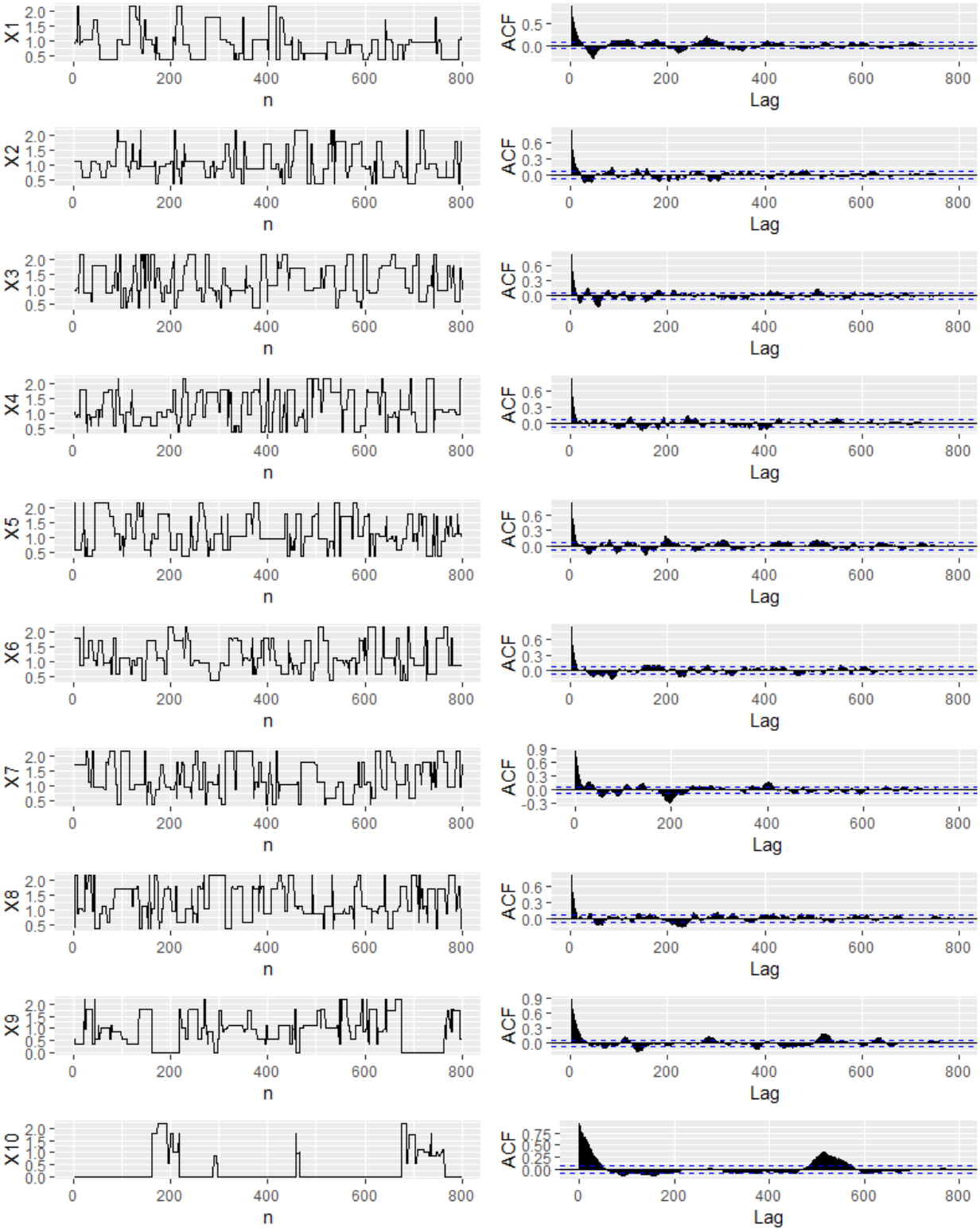


Figure 4: Diagnostic plots for parallel tempering



Appendix B: R Code

```
library(forecast)
library(tidyverse)
library(gridExtra)

N <- 150 # number of terms for inner num over n
J <- 50 # number of terms for outer sum over j

erfc <- function(x) {2*pnorm(-sqrt(2)*x)} # complementary error function

coeffs <- function(z,j) # calculate the functions d_{j,n}(z) and d'_{j,n}(z)
{
  t <- (4*j + 1)^2*pi^2/(8*z)
  tp <- -(4*j + 1)^2*pi^2/(8*z^2)
  d <- vector(length=(N+1))
  d[1] <- pi*exp(-t)/sqrt(2*t) # d0
  d[2] <- pi*sqrt(pi/2)*erfc(sqrt(t)) # d1
  dp <- vector(length=(N+1))
  dp[1] <- pi*exp(-t)*(sqrt(t) + 1/(2*sqrt(t)))/(z*sqrt(2)) # d'0
  dp[2] <- pi*exp(-t)*sqrt(t)/(z*sqrt(2)) # d'1

  for (n in 2:N) # use our recursions
  {
    d[n+1] <- ((n-1 - 1/2 - t)*d[n] + t*d[n-1])/(n-1)
    dp[n+1] <- ((n-1 - 1/2 - t)*dp[n] + tp*(d[n-1] - d[n]) + t*dp[n-1])/(n-1)
  }

  return(rbind(d, dp))
}

g <- function(z,j) # auxiliary function for use in sapply later
{
  C <- coeffs(z,j)
  d <- C[1,]
  dp <- C[2,]
  return(choose(-1/2, j)*(4*j + 1)
         *sum((dp*z^(-1:(N-1)) + (-1:(N-1))*d*z^(-2:(N-2)))
              /(8^(0:N)*factorial(0:N)))) # always vectorize!
}

f <- function(z) # calculate f_infinity
{
  if (z <= 0)
  {
    return(0)
  } else
  {
    s <- sum(sapply(c(0:J), FUN=g2 <- function(j) g(z,j)))
    return(s)
  }
}
```

```
x <- as.array(seq(0.1, 10, 0.05))
y <- apply(X=x, MARGIN=c(1), FUN=f)
Z <- data.frame(x, f_inf = y)
ggplot(data=Z, aes(x=x, y=f_inf)) + geom_line(color='blue')


# Some MCMC algorithms

varfact <- function(x) {2*sum(acf(x, plot=FALSE)$acf)-1} # to compute varfact

# Random-walk Metropolis

acc <- 0 # keep track of acceptance counts
M <- 5000 # number of simulations
B <- M/5 # number of burn-offs
sigma <- 1.8 # for scaling
X <- vector(length=M)
X[1] <- abs(rnorm(n=1)) # initalize X
for (j in 2:M)
{
  Y <- X[j-1] + rnorm(n=1, mean=0, sd=sigma) # propose new X
  if (runif(1) < min(1, f(Y)/f(X[j-1])))
  {
    X[j] <- Y
    acc <- acc+1
  } else
  {
    X[j] <- X[j-1]
  }
}

accrate <- acc/M
Y <- X[(B+1):M] # burn-off first B samples
Ihat <- mean(Y) # estimate 1
CI_Ihat_l <- Ihat - 1.96*sd(Y)*sqrt(varfact(Y)/(M-B))
CI_Ihat_r <- Ihat + 1.96*sd(Y)*sqrt(varfact(Y)/(M-B))
varhat <- var(Y) # estimate 2*pi^2/3 - 6
CI_varhat_l <- varhat - 1.96*sd(Y^2)*sqrt(varfact(Y^2)/(M-B))
CI_varhat_r <- varhat + 1.96*sd(Y^2)*sqrt(varfact(Y^2)/(M-B))

D <- data.frame(n=1:(M-B), X=Y, VarX=Y^2 - mean(Y)^2)
tr_X <- ggplot(D, aes(n, y=X)) + geom_path()
tr_VarX <- ggplot(D, aes(n, y=VarX)) + geom_path()
acf_X <- ggAcf(D$X, lag.max=10000)
acf_VarX <- ggAcf(D$VarX, lag.max=10000)
grid.arrange(tr_X, tr_VarX, acf_X, acf_VarX, nrow=2)

# RWM output

> Ihat
[1] 0.9601668
> CI_Ihat_l
[1] 0.8699388
> CI_Ihat_r
```

```
[1] 1.050395
> varhat
[1] 0.5693503
> CI_varhat_l
[1] 0.1870014
> CI_varhat_r
[1] 0.9516992
> accrate
[1] 0.3046
```

```
# Metropolis-Hastings with weird log-Normal proposal

acc <- 0 # keep track of acceptance counts
M <- 5000 # number of simulations
B <- M/5 # number of burn-offs
sigma <- 1.5 # for scaling
X <- vector(length=M)
X[1] <- abs(rnorm(n=1)) # initalize X

# q <- function(x,y) {return(dexp(x=y, rate=sigma*x))} # proposal distribution
q <- function(x,y) {return(dlnorm(x=y, meanlog=x, sdlog=sigma*x))} # proposal distribution

for (j in 2:M)
{
  Y <- rlnorm(n=1, meanlog=X[j-1], sdlog=sigma*X[j-1]) # propose new X
  if (runif(1) < min(1, (f(Y)*q(Y, X[j-1]))/(f(X[j-1])*q(X[j-1],Y))))
  {
    X[j] <- Y
    acc <- acc+1
  } else
  {
    X[j] <- X[j-1]
  }
}

accrate <- acc/M
Y <- X[(B+1):M] # burn-off first B samples
Ihat <- mean(Y) # estimate 1
CI_Ihat_l <- Ihat - 1.96*sd(Y)*sqrt(varfact(Y)/(M-B))
CI_Ihat_r <- Ihat + 1.96*sd(Y)*sqrt(varfact(Y)/(M-B))
varhat <- var(Y) # estimate 2*pi^2/3 - 6
CI_varhat_l <- varhat - 1.96*sd(Y^2)*sqrt(varfact(Y^2)/(M-B))
CI_varhat_r <- varhat + 1.96*sd(Y^2)*sqrt(varfact(Y^2)/(M-B))
```

```
> Ihat
[1] 0.9797862
> CI_Ihat_l
[1] 0.9032351
> CI_Ihat_r
[1] 1.056337
> varhat
[1] 0.5055809
```

```
> CI_varhat_l
[1] 0.1988189
> CI_varhat_r
[1] 0.8123429
> accrate
[1] 0.2852
```

```
# Parallel tempering

gT = function(x, tau) {
  if ( tau < 1 || tau > maxtau )
  {
    return(0)
  } else
  {
    return( f(x)^(1/tau))
  }
}


M <- 1000
B <- M/5
sigma <- 1 # proposal scaling
maxtau <- 10 # maximum temperature
X <- matrix(rep(0,M*maxtau), ncol=maxtau)
Z <- abs(rnorm(n=maxtau)) # initialize Markov chain
accX <- rep(0, maxtau) # counts probabilities for each chain
acctau <- 0 # acceptance counts for temperature swaps

for (j in 2:M)
{
  print(j)
  for (tau in 1:maxtau)
  {
    Y <- Z[tau] + rnorm(n=1, mean=0, sd=sigma)
    if (runif(1) < min(1, gT(Y, tau)/gT(Z[tau], tau)))
    {
      X[tau] <- Y
      accX[tau] <- accX[tau] + 1
    }
  }

  m <- sample(1:maxtau, size=1) # pick random temperatures
  n <- sample(1:maxtau, size=1)
  if (runif(1) < min(1, (gT(Z[m],n)*gT(Z[n],m))/(gT(Z[n],n)*gT(Z[m],m))))
  { # propose to swap Xtau with Xtau'
    t <- Z[m]
    Z[m] <- Z[n]
    Z[n] <- t
    acctau <- acctau + 1
  }
```

```
  X[j,] <- Z
}

Ihat <- vector(length=maxtau)
varhat <- vector(length=maxtau)
CI_Ihat_l <- vector(length=maxtau)
CI_Ihat_r <- vector(length=maxtau)
for (tau in 1:maxtau)
{
  Ihat[tau] <- mean(X[((B+1):M),tau]) # burn-off first B samples
  varhat[tau] <- var(X[((B+1):M),tau]) # burn-off first B samples
  CI_Ihat_l[tau] <- Ihat[tau] - 1.96*sd(X[((B+1):M),tau])*sqrt(varfact(X[((B+1):M),tau])/(M-B))
  CI_Ihat_r[tau] <- Ihat[tau] + 1.96*sd(X[((B+1):M),tau])*sqrt(varfact(X[((B+1):M),tau])/(M-B))
}

D <- data.frame(n=1:(M-B), X1=X[((B+1):M),1],
                X2=X[((B+1):M),2],
                X3=X[((B+1):M),3],
                X4=X[((B+1):M),4],
                X5=X[((B+1):M),5],
                X6=X[((B+1):M),6],
                X7=X[((B+1):M),7],
                X8=X[((B+1):M),8],
                X9=X[((B+1):M),9],
                X10=X[((B+1):M),10])
tr_X1 <- ggplot(D, aes(n, y=X1)) + geom_path()
tr_X2 <- ggplot(D, aes(n, y=X2)) + geom_path()
tr_X3 <- ggplot(D, aes(n, y=X3)) + geom_path()
tr_X4 <- ggplot(D, aes(n, y=X4)) + geom_path()
tr_X5 <- ggplot(D, aes(n, y=X5)) + geom_path()
tr_X6 <- ggplot(D, aes(n, y=X6)) + geom_path()
tr_X7 <- ggplot(D, aes(n, y=X7)) + geom_path()
tr_X8 <- ggplot(D, aes(n, y=X8)) + geom_path()
tr_X9 <- ggplot(D, aes(n, y=X9)) + geom_path()
tr_X10 <- ggplot(D, aes(n, y=X10)) + geom_path()

acf_X1 <- ggAcf(D$X1, lag.max=1000, main = NULL)
acf_X2 <- ggAcf(D$X2, lag.max=1000, main = NULL)
acf_X3 <- ggAcf(D$X3, lag.max=1000, main = NULL)
acf_X4 <- ggAcf(D$X4, lag.max=1000, main = NULL)
acf_X5 <- ggAcf(D$X5, lag.max=1000, main = NULL)
acf_X6 <- ggAcf(D$X6, lag.max=1000, main = NULL)
acf_X7 <- ggAcf(D$X7, lag.max=1000, main = NULL)
acf_X8 <- ggAcf(D$X8, lag.max=1000, main = NULL)
acf_X9 <- ggAcf(D$X9, lag.max=1000, main = NULL)
acf_X10 <- ggAcf(D$X10, lag.max=1000, main = NULL)

grid.arrange(tr_X1, acf_X1,
             tr_X2, acf_X2,
             tr_X3, acf_X3,
             tr_X4, acf_X4,
             tr_X5, acf_X5,
             tr_X6, acf_X6,
             tr_X7, acf_X7,
```

```
tr_X8, acf_X8,
tr_X9, acf_X9,
tr_X10, acf_X10, nrow=10)
```