9.1. Returning to Example 9.1, set the target distribution to \( X \sim \text{Rayleigh}(\sigma) \), now for \( \sigma = 2 \).


```r
f901 <- function(x, sigma) {
    if (any(x < 0)) return (0)
    stopifnot(sigma > 0)
    return((x / sigma^2) * exp(-x^2 / (2*sigma^2)))
}   #end function

set.seed(901)
m <- 10000
sigma <- 2          #sigma now set to 2
x <- numeric(m)
x[1] <- rchisq(1, df=1)
k <- 0
u <- runif(m)

for (i in 2:m) {
    xt <- x[i-1]
y <- rchisq(1, df = xt)
    num <- f901(y, sigma) * dchisq(xt, df = y)
    den <- f901(xt, sigma) * dchisq(y, df = xt)
    if (u[i] <= num/den) x[i] <- y else {
        x[i] <- xt
        k <- k+1     #y is rejected
    }   #end if/else
} #end for loop

print(k/m)  #rejection rate
```

The rejection rate is given as

```
[1] 0.5244
```

(so acceptance is about 49% ⇒ OK mixing). For a burn-in of \( b_0 = 2000 \) draws, further \texttt{R} code produces the trace plot:

```r
b0 = 2000                #burn-in
index <- (b0+1):m        #chain after burn-in
y1 <- x[index]
plot( y1~index, type="l", main="", ylab="x" )
```

plot follows →
9.1. (cont’d)

Notice the range of the retained chain:

```r
> range(x)
[1]  0.05046647 8.81537099
```

Compare this to a trace plot with $\sigma = 4$ (as in Example 9.1, but extended to visualize the entire post-burn chain) we see a rather similar plot, but not surprisingly, over a much wider range:

```r
> range(x)
[1]  0.2188613 16.2859098
```
9.2. Returning to Example 9.1, continue with the target distribution as \( X \sim \text{Rayleigh}(\sigma) \) with \( \sigma = 4 \), but now employ \( Y \sim \text{Gamma}(X, 1) \) as the proposal distribution. Sample \texttt{R} code modified from the textbook’s website \url{http://personal.bgsu.edu/~mrizzo/SCR/SCRch9.R} is

```r
f902 <- function(x, sigma) {
  if (any(x < 0)) return (0)
  stopifnot(sigma > 0)
  return((x / sigma^2) * exp(-x^2 / (2*sigma^2)))
} # end function

set.seed(902)
m <- 10000
sigma <- 4
x <- numeric(m)
x[1] <- rgamma(1, 1, 1)
k <- 0
u <- runif(m)

for (i in 2:m) {
  xt <- x[i-1]
  y <- rgamma(1, shape = xt, rate = 1)
  num <- f902(y, sigma) * dgamma(xt, shape = y, rate = 1)
  den <- f902(xt, sigma) * dgamma(y, shape = xt, rate = 1)
  if (u[i] <= num/den) x[i] <- y else {
    x[i] <- xt
    k <- k+1     # y is rejected
  } # end if/else
} # end for loop

print(k/m)  # rejection rate
```

The rejection rate is given as

\[ 0.3118 \] (lower than that with the \( \chi^2 \) proposal, suggesting higher acceptance and inefficient mixing). For a burn-in of \( b_0 = 2000 \) draws, further \texttt{R} code produces the trace plot:

```r
b0 = 2000 # burn-in
index <- (b0+1):m # chain after burn-in
y1 <- x[index]
plot( y1~index, type="l", main="", ylab="x" )
```

The plot appears reasonable, and very much similar to its counterpart (with \( \sigma = 4 \)) in Exercise 9.1

plot follows →
9.2. (cont’d)

A histogram of the retained chain, along with the target Rayleigh(4) p.d.f., from the R code

```r
hist( y1, prob=T, main='', xlab='x', ylim=c(0,.155) )
xarg = seq( 0, max(y1), 0.1 )
lines( xarg, f902(xarg, sigma), ylim=c(0,.155) )
```

shows good agreement:
9.3. For a target distribution \( X \sim \text{Cauchy}(\theta = 1, \eta = 0) \) consider \( Y \sim \text{N}(X, \sigma^2) \) as the proposal distribution, since it has the same support as the Cauchy and is easy to sample from. Since the normal is a conditionally symmetric proposal density, this then reduces to the simple Metropolis algorithm. Sample \( \mathbf{R} \) code similar to that in Exercise 9.1 is

```r
f903 <- function(x, eta=0, theta=1) {
  stopifnot( theta > 0 )
  return( 1 / (pi*theta * (1 + ((x-eta)/theta)^2)) )
}  #end function
```

```r
set.seed(903)
m <- 10000
sigma = 1                 #try sigma = 1 for proposal scale
x <- numeric(m)
x[1] <- rnorm(1,0,sigma)  #initialize with \( X_0 \sim \text{N}(0,\sigma^2) \)
k <- 0

u <- runif(m)
for (i in 2:m) {
  xt <- x[i-1]
y <- rnorm(1, mean = xt, sd = sigma)
num <- f903(y) * dnorm(xt, mean = y, sd = sigma)
den <- f903(xt) * dnorm(y, mean = xt, sd = sigma)
if (u[i] <= num/den) x[i] <- y else {
  x[i] <- xt
  k <- k+1                #y is rejected
}   #end if/else
}                 #end for loop

print(k/m)                       #rejection rate
```

(One could just use \( \text{dcauchy}(x) \) instead of the constructed function \( f903(x) \) for the Cauchy p.d.f.) The rejection rate is given as

```
[1] 0.239
```

which is rather low (i.e., higher-than-desired acceptance). Setting the burn-in to \( b_o = 1000 \) draws, the retained chain is found via

```r
b0 = 1000                        #burn-in
index = (b0+1):m
y1 = x[index]                   #chain after burn-in
```

To compare deciles with the theoretical Cauchy(0,1) p.d.f., sample \( \mathbf{R} \) code is

```r
p10 = seq(.1,.9, .1)
round( rbind( quantile(y1, p10), qcauchy(p10) ), 3 )
```

which produces not-unreasonable comparisons, but with some discrepancies in the tails:

```
 10%   20%   30%   40%   50%   60%   70%   80%   90%
[1,] -2.951 -1.361 -0.704 -0.304 -0.009 0.303 0.675 1.247 2.753
[2,] -3.078 -1.376 -0.727 -0.325  0.000 0.325 0.727 1.376 3.078
```

cont’d
9.3. (cont’d)

The trace plot, from

```r
plot( y1~index, type="l", main="", ylab="x" )
```

shows an unusual instability in the first 1000 or so elements of the retained chain. (It may be better to set the burn-in to \( b_0 = 2000 \).)

Interestingly, a histogram of the retained chain (using \( b_0 = 1000 \), with the target Cauchy(0,1) p.d.f., from the R code

```r
hist( y1, prob=T, main="", xlab='x', ylim=c(0,.35), breaks=50 )
xarg = seq( min(y1), max(y1), 0.1 )
lines( xarg, f903(xarg), ylim=c(0,.35) )
```

displays OK agreement:
Moving to, say, \( m = 50,000 \) draws leads to slight, but not satiating improvement (try it!). Living up to its reputation, the Cauchy is a difficult distribution with which to operate.

9.4. The standard Laplace p.d.f. is \( f(x) = \frac{1}{2} \exp\{-|x|\} \). Sample \texttt{R} code for a random walk Metropolis sampler, using \( Y \sim N(X_0, \sigma^2) \) for the proposal p.d.f. to produce the ‘random walk’ feature, is

\[
\begin{align*}
\texttt{f904} & \leftarrow \text{function}(x) \{ -0.5 \cdot \exp(-\text{abs}(x)) \} \\
\texttt{set.seed}(904)
\end{align*}
\]

\[
\begin{align*}
\texttt{rw.MetropolisL} & \leftarrow \text{function}(\sigma, x0, m) \{  \\
& \texttt{x} \leftarrow \text{numeric}(m)  \\
& \texttt{x}[1] \leftarrow x0  \\
& \texttt{u} \leftarrow \text{runif}(m)  \\
& \texttt{k} \leftarrow 0  \\
& \text{for } (i \text{ in } 2:m) \{  \\
& \quad \texttt{y} \leftarrow \text{rnorm}(1, x[i-1], \sigma)  \\
& \quad \text{if } (\texttt{u}[i] \leq (\texttt{f904}(y) / \texttt{f904}(x[i-1])))  \\
& \quad \quad \texttt{x}[i] \leftarrow y \text{ else } \{  \\
& \quad \quad \texttt{x}[i] \leftarrow \texttt{x}[i-1]  \\
& \quad \quad \texttt{k} \leftarrow k + 1  \\
& \quad \}  \\
& \}  \\
& \text{return(list(x=x, k=k))}  \\
\}
\end{align*}
\]
Mimicking the approach in Example 9.3, set \( m = 2000 \) and vary \( \sigma \) over 0.05, 0.50, 2.0, 16.0. Use an arbitrary initial point of \( X_0 = 25 \). Then, find the corresponding four chains via the sample R code

```r
m <- 2000
sigma <- c(.05, .5, 2, 16)
x0 <- 25
rw1 <- rw.MetropolisL(sigma[1], x0, m)
rw2 <- rw.MetropolisL(sigma[2], x0, m)
rw3 <- rw.MetropolisL(sigma[3], x0, m)
rw4 <- rw.MetropolisL(sigma[4], x0, m)
accept = 1-c(rw1$k/m, rw2$k/m, rw3$k/m, rw4$k/m)  #acceptance rates
round( rbind(sigma, accept), 3)
```

This gives

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sigma</td>
<td>0.050</td>
<td>0.500</td>
<td>2.000</td>
</tr>
<tr>
<td>accept</td>
<td>0.982</td>
<td>0.844</td>
<td>0.516</td>
</tr>
</tbody>
</table>

which are clearly quite disparate. The raw trace plots (without burn-in) are generated via

```r
par(mfrow=c(2,2))                #display 4 graphs together
rw <- cbind(rw1$x, rw2$x, rw3$x, rw4$x)
for (j in 1:4) {
  plot(rw[,j], type="l", xlab=bquote(sigma == .(round(sigma[j],3))),
       ylab="X", ylim=range(rw[,j]))
}   #end for loop
```

The multi-plot graphic follows:
The effect is similar to that seen in Figure 9.3 of Example 9.3; the standard Laplace p.d.f. has median at $x = 0$ and varies symmetrically from that point. We see the chain settles around $x = 0$ and also varies symmetrically about it, as seen best when $\sigma = 2$.

9.6. Set the prior on $\theta$ to $\theta \sim U(0,1)$ so that $\pi(\theta) = I_{(0,1)}(\theta)$. The multinomial likelihood is $f(X|\theta) \propto \prod_i p_i(\theta)^{x_i}/x_i!$, where $p_1(\theta) = (2+\theta)/4$, $p_2(\theta) = p_3(\theta) = (1-\theta)/4$, and $p_4(\theta) = \theta/4$. Combining these via Bayes’ Rule produces the posterior

$$f(\theta|X) = \pi(\theta)f(X|\theta)/f(X) \propto (2+\theta)^{x_1}(1-\theta)^{x_2+x_3}\theta^{x_4}.$$ 

Notice that posterior specification to proportionality is all that is needed here, since any constants in $f(\theta|X)$ will cancel in the construction of $r(\theta_t,Y)$ in the McMC sampler.

For the proposal density, we usually desire a p.d.f. $g(y|\theta)$ defined over a space the same as that for $\theta$; here this is $0 < y < 1$. The simple choice of $g(y|\theta) = I_{(0,1)}(y)$, i.e., a uniform proposal independent of $\theta$, corresponds to an independence sampler. However, independence samplers are not typically recommended. Instead, try a beta p.d.f. proposal with expected value equal to $\theta_t$. Different possibilities exist; one is $Y|\theta_t \sim \beta(\theta_t/(1-\theta_t), 1)$, where $E[Y|\theta_t] = [\theta_t/(1-\theta_t)]/[1 + (\theta_t/(1-\theta_t))] = \theta_t$. (For $0 < \theta < 1$, this is a right-skewed, decreasing p.d.f.) Sample R code for an McMC sampler from $f(\theta|X)$ (with portions modified from http://personal.bgsu.edu/~mrizzo/SCR/SCRch9.R) is then

```r
f906 <- function( th,x ) {
  if (th<0 || th>1 ) return (0)
}  #end function

xdata = c( 125, 18, 20, 34 )      #observed multinom. data
m = 10000
set.seed(906)

th = numeric(m)
th[1] = runif(1)        #initialize: sample from prior on theta
k = 0
u = runif(m)

##employ skewed beta proposal density
for (t in 2:m) {
  xt = th[t-1]
  alph = xt/(1-xt)
  y <- rbeta(1, shape1=alph, shape2=1  )
  numer = f906( y,xdata ) * dbeta( xt, y/(1-y), 1)
  denom = f906( xt,xdata ) * dbeta( y, alph, 1)
  if ( u[t] <= numer/denom )
    th[t] = y else {
    th[t] = th[t-1]
    k = k + 1
  }  #end if/else
}                  #end for loop
```
Using this code, the posterior sample of 10,000 points resides in the vector `th`; its raw trace plot is found via

```r
plot( th, type="l", ylim=range(th),
     xlab=bquote(theta), ylab="posterior" )
```

This produces

![Trace plot](image)

which appears somewhat variable, but not otherwise unacceptable. A burn-in of perhaps $b_0 = 2000$ will likely suffice. The consequent histogram, found via

```r
hist( th[2001:m], prob=T, breaks="Scott",
     xlab=bquote(theta), ylab='posterior', main="" )
```

shows a generally unimodal, slightly left-skewed sample.

(histogram follows →)
9.6. (cont’d)

The associated posterior mean (a Bayesian estimate) of $\theta$, and the corresponding posterior ‘cell’ probabilities are found via

$$\hat{\theta} = \text{mean}(\ \text{th}[2001:m])$$

$$[1] 0.623192$$

and

$$c(0.5 + \hat{\theta}/4, (1 - \hat{\theta})/4, (1 - \hat{\theta})/4, \hat{\theta}/4) \quad \# \text{p.hat vector}$$

$$[1] 0.655798 \ 0.094202 \ 0.094202 \ 0.155798$$

respectively.

9.7. Sample R code for a Gibbs sampler for bivariate normal with zero means, unit variances, and $\rho = 0.9$:

```r
m = 10000
b0 = 1000
set.seed(907)
```

(code continues →)
9.7. (Gibbs sampler R code, cont’d)

\[ \rho = 0.9 \]
\[ \mu_1 = \mu_2 = 0 \]
\[ \sigma_1 = \sigma_2 = 1 \]
\[ s_1 = \sqrt{1 - \rho^2} \cdot \sigma_1 \]
\[ s_2 = \sqrt{1 - \rho^2} \cdot \sigma_2 \]

\[ X = \text{matrix}(0, \text{nrow}=m, \text{ncol}=2) \]


#### generate the chain

\[ X[1, ] \leftarrow c(\mu_1, \mu_2) \quad \# \text{initialize} \]
\[ \text{for (i in 2:m)} \{ \]
\[ \quad x2 \leftarrow X[i-1, 2] \]
\[ \quad m1 \leftarrow \mu_1 + \rho \times (x2 - \mu_2) \times \sigma_1/\sigma_2 \]
\[ \quad X[i, 1] \leftarrow \text{rnorm}(1, m1, s1) \]
\[ \quad x1 \leftarrow X[i, 1] \]
\[ \quad m2 \leftarrow \mu_2 + \rho \times (x1 - \mu_1) \times \sigma_2/\sigma_1 \]
\[ \quad X[i, 2] \leftarrow \text{rnorm}(1, m2, s2) \]
\[ \} \]
\[ x \leftarrow X[(b0+1):m, 1] \]
\[ Y \leftarrow X[(b0+1):m, 2] \]

With this, plot \( Y \) against \( x \):

\[ \text{plot( Y~x ); abline( h=0, v=0 )} \]

producing a typical elliptical scatter with high positive correlation, as expected:
9.7. (cont’d)

Residuals from the fit can be plotted against fitted values and checked for normality via a quantile-quantile plot:

```r
resid = resid( lm(Y~x) )
par(mfrow=c(1,2))
plot( resid~fitted(lm(Y~x)), xlab='fitted value' ); abline( h=0 )
qqnorm( resid ); qqline( resid )
```

producing reasonable graphics for comparison to a normal distribution.

9.8. Sample R code for a Gibbs sampler for the given binomial/beta construction with \( a = 2 \), \( b = 3 \), and at \( n = 10 \):

```r
Gibbs908 = function(m, a, b, n){
  X = matrix(0, nrow=m, ncol=2)
  y = (0.5*n + a)/(n + a + b)    #initialize y with E[Y | n/2]
  x = floor( n*y )               #initialize x with E[X | Y]
  X[1,] = c( x,y )               #initialize matrix
  for (t in 2:m) {
    y = X[t-1, 2]
    X[t, 1] = rbinom( 1, size=n, y)   #x|y~bin(n,y)
    x = X[t, 1]
    X[t, 2] = rbeta( 1, x+a, n-x+b )  #y|x~beta(x+a,n-x+b)
  }
  return( X )
}
```

(code continues →)
9.8. (Gibbs sampler R code, cont’d)

```r
set.seed(908)
m = 10000 # size of chain
b0 = 1000 # burn-in
a = 2
b = 3
n = 10

XYGibbs = Gibbs908(m, a, b, n)
aftburn = b0+1

To study the sampled chain, we can:
(a) plot the sampled joint distribution (notice the discrete nature of x)

plot( XYGibbs[aftburn:m,], xlab='x', ylab='y' )

(b) plot the sampled marginal posterior distribution of y

hist( XYGibbs[aftburn:m,2], prob=T, breaks='scott', main='', xlab='y' )
```

(plot follows →)
9.8. (cont’d)

Sampled posterior for $y$ is unimodal with a right skew:

(c) find the sampled (discrete) marginal distribution of $x$ and also plot the marginal posterior distribution

```r
xGibbs = XYGibbs[aftburn:m,1]
fx.hat = table(xGibbs)/length(xGibbs)
round(fx.hat,3)
```

```
xGibbs
  0  1  2  3  4  5  6  7  8  9 10
0.066 0.108 0.140 0.135 0.143 0.134 0.107 0.077 0.048 0.031 0.012
```

```r
barplot(fx.hat, space=0, ylim=c(0, 0.15), xlab="x",
        ylab='marginal posterior', main='')
abline(h=0)
```

(plot follows →)
9.8. (cont’d)

Sampled posterior for $x$ is unimodal and also skews slightly to the right:

![Histogram of sampled posterior for $x$.](image)


```r
set.seed(909)
sigma = 0.2
k <- 4       # number of chains to generate
n <- 15000   # length of chains
b <- 1000    # burn-in length
x0 <- c(-10, -5, 5, 10)

# generate the chains
X <- matrix(0, nrow=k, ncol=n)
for (i in 1:k)
  X[i, ] <- normal.chain(sigma, n, x0[i])
```

(code continues →)
9.9. (Gelman-Rubin convergence R code, cont’d)

```r
# compute diagnostic statistics
psi <- t(apply(X, 1, cumsum))
for (i in 1:nrow(psi))
    psi[i,] <- psi[i,] / (1:ncol(psi))

print(Gelman.Rubin(psi))
```

We find $\hat{R}$ (for our choice of seed) is

```
[1] 1.109622
```