

Geometric MCMC sampling with `geomc`

Vivekananda Roy

2026-05-08

Contents

What is Geometric approach to MCMC?	1
Basic Usage	2
Example 1: Sampling from a Multivariate Normal	2
Example 2: Bayesian Inference for Normal Data	7
Understanding the Output	12
Example 3: Univariate Mixture of Normals	14
Univariate Mixture of Normals: Run <code>geomc</code> with Default Settings	14
Univariate Mixture of Normals: Run <code>geomc</code> with Custom Random Walk Base Density	16
Univariate Mixture of Normals: Run <code>geomc</code> with Informed Approximate Targets	18
Univariate Mixture of Normals: Run <code>geomc</code> with Informed Approximate Targets along with a Custom Random Walk Base Density	20
Univariate Mixture of Normals: Run <code>geomc</code> with other choices of <code>eps</code>	22
Univariate Mixture of Normals: Run <code>geomc</code> with another (non-Gaussian) Informed Approximate Target	27
Univariate Mixture of Normals: Importance Sampling to estimate inner products for non-Gaussian densities	28
Example 4: Bivariate Mixture of Normals	30
Bivariate Mixture of Normals: Run <code>geomc</code> with Informed Approximate Targets along with a Custom Random Walk Base Density	31
Bivariate Mixture of Normals: Run <code>geomc</code> with a diffuse <code>g</code>	34
Bivariate Mixture of Normals: Comparison with Random Walk Metropolis	36
Example 2 Continued: Bayesian Inference for Normal Data with MALA Base Density	38
Example 5: Discrete Distribution (Binomial)	41
Discrete Distribution (Binomial): Run <code>geomc</code> with Random Walk Base (and user defined <code>bhat.coef</code> function)	41
Discrete Distribution (Binomial): Run <code>geomc</code> with Reflecting Random Walk Base (and user defined <code>bhat.coef</code> function)	44
Discrete Distribution (Binomial): Run <code>geomc</code> with Uniform Base (an example with <code>ind</code> set as TRUE)	48
Summary of Non-Default Settings	50
References	50

What is Geometric approach to MCMC?

Geometric approach to MCMC (`geomc`) is a Riemannian manifold based method proposed by Roy (2024) for constructing informative proposal distributions for Metropolis-Hastings (MH) algorithms. Unlike traditional random walk Metropolis algorithms, geometric MH sampling adapts to the Riemannian geometry of the target space. The geometric MH algorithm transports a base uninformed kernel (e.g., a random walk proposal) along geodesic paths informed by global and local approximations of the target density. `geomc` can be used to sample from both discrete and continuous distributions and this vignette illustrates its use for sampling from a target distribution of interest.

The method can be effective for:

- Multimodal distributions (can move between modes efficiently)
- High-dimensional problems (adapts to local geometry)
- Complex posterior distributions in Bayesian inference

```
# Install using install.packages('geommc')
library(geommc)
```

Basic Usage

The main function is `geomc()`, which requires:

- `logp`: A function that evaluates the log-target density (or a list with additional specifications)
- `initial`: Starting values for the MCMC chain
- `n.iter`: Number of iterations to run

We denote the target density by ψ . Following Roy (2024), `geomc` constructs informative geometric MH proposals by perturbing a ‘baseline’ proposal density $f(y|x)$ along directions specified by $\mathcal{G} = \{g_1, g_2, \dots, g_k\}$, a set of k pdfs representing suitable (local or global) approximations of the target density ψ . Roy (2024) showed that the geometric MH chain dominates the MH chain corresponding the ‘base’ proposal with respect to different Markov chain orderings by at least a strictly positive factor. Let’s start with two simple examples.

Example 1: Sampling from a Multivariate Normal

We’ll sample from a bivariate normal distribution to demonstrate the basic workflow. Thus, the target density is

$$\psi(y) = \phi_2(y; \mu, \Sigma),$$

where $\phi_d(y; \mu, \Sigma)$ denotes the pdf of the d –dimensional normal distribution with mean vector μ , covariance matrix Σ , and evaluated at y .

```
# Define the log-target density
log_target_mvnorm <- function(x, target.mean, target.Sigma) {
  d <- length(x)
  xc <- x - target.mean
  Q <- solve(target.Sigma)
  -0.5 * drop(t(xc) %*% Q %*% xc)
}

# Target distribution parameters
target_mean <- c(1, -2)
target_Sigma <- matrix(c(1.5, 0.7, 0.7, 2.0), 2, 2)

# Run geomc with default settings
set.seed(3)
result <- geomc(
  logp = log_target_mvnorm,
  initial = c(0, 0),
  n.iter = 1000,
  target.mean = target_mean,
  target.Sigma = target_Sigma
)
```

Note how we pass additional arguments (`target.mean` and `target.Sigma`) through `...` to the `logp` (log-target) function.

Examining the Results

The `geomc()` function returns a list with several components:

```
names(result)
```

```
#> [1] "var.base"      "mean.ap.tar"    "var.ap.tar"
#> [4] "samples"       "log.p"          "acceptance.rate"
#> [7] "model.case"    "gaus"           "ind"
```

The most important components are:

- `samples`: The MCMC samples (matrix with `n.iter` rows)
- `acceptance.rate`: The proportion of accepted proposals
- `log.p`: Log-density values at each sample

```
# Sample means (should be close to target_mean)
cat("Sample means:\n")
```

```
#> Sample means:
```

```
print(colMeans(result$samples))
```

```
#> [1]  0.959010 -2.029356
```

```
cat("\nTarget means:\n")
```

```
#>
```

```
#> Target means:
```

```
print(target_mean)
```

```
#> [1]  1 -2
```

```
# Sample covariance (should be close to target_Sigma)
cat("\nSample covariance:\n")
```

```
#>
```

```
#> Sample covariance:
```

```
print(cov(result$samples))
```

```
#>      [,1]      [,2]
```

```
#> [1,] 1.4164482 0.7027092
```

```
#> [2,] 0.7027092 2.1983867
```

```
cat("\nTarget covariance:\n")
```

```
#>
```

```
#> Target covariance:
```

```
print(target_Sigma)
```

```
#>      [,1] [,2]
```

```
#> [1,]  1.5  0.7
```

```
#> [2,]  0.7  2.0
```

Visualizing the Results

Let's create trace, autocorrelation, density, and scatter plots to assess convergence and mixing. See Roy (2020) for a review of MCMC convergence assessment methods.

```

par(mfrow = c(3, 2))

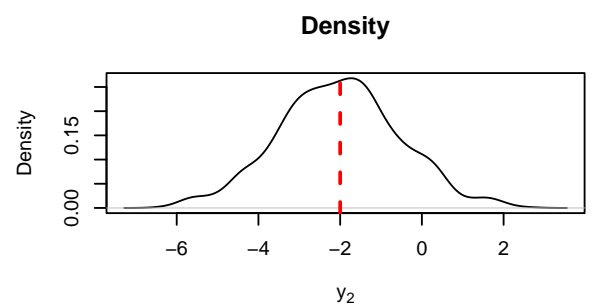
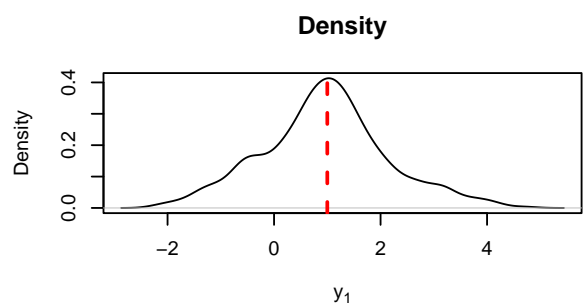
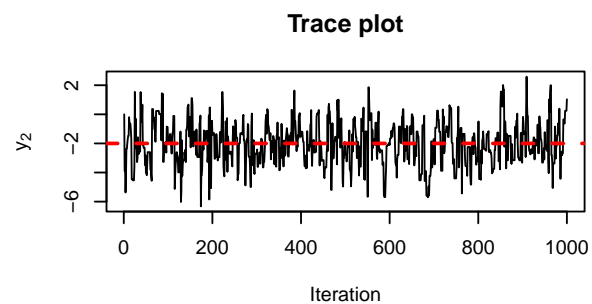
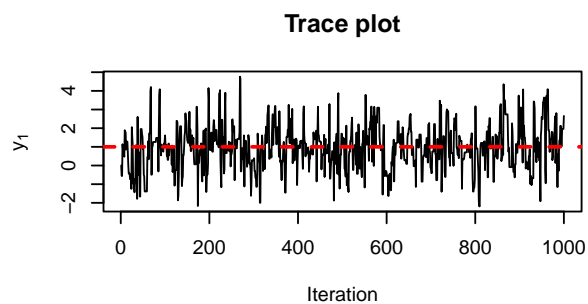
# Trace plots
plot(result$samples[, 1], type = "l",
     main = "Trace plot",
     ylab = expression(y[1]), xlab = "Iteration")
abline(h = target_mean[1], col = "red", lty = 2, lwd = 2)

plot(result$samples[, 2], type = "l",
     main = "Trace plot",
     ylab = expression(y[2]), xlab = "Iteration")
abline(h = target_mean[2], col = "red", lty = 2, lwd = 2)

# Density plots
plot(density(result$samples[, 1]),
     main = "Density",
     xlab = expression(y[1]))
abline(v = target_mean[1], col = "red", lty = 2, lwd = 2)

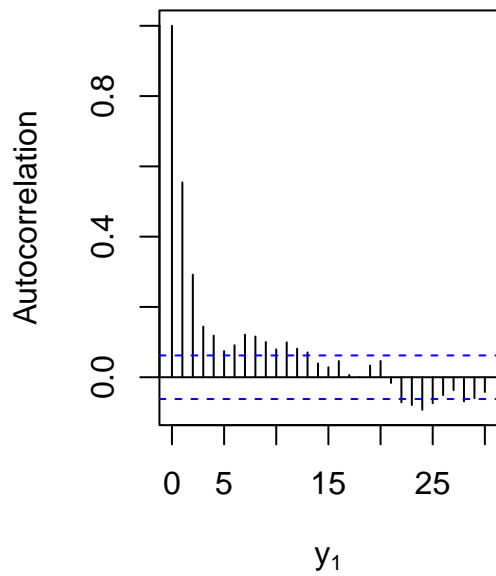
plot(density(result$samples[, 2]),
     main = "Density",
     xlab = expression(y[2]))
abline(v = target_mean[2], col = "red", lty = 2, lwd = 2)

```

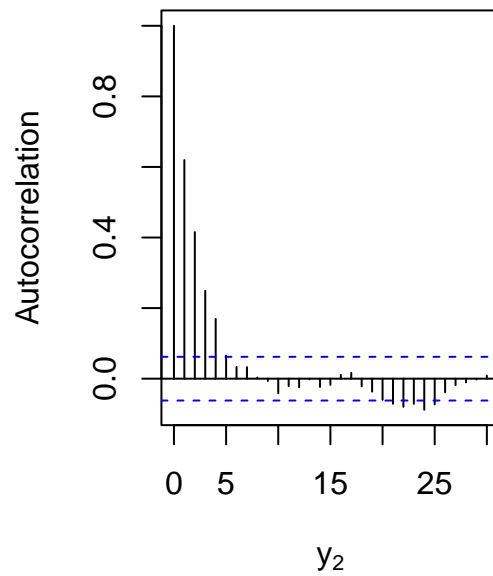


```
# Autocorrelation plot of samples
par(mfrow = c(1, 2))
acf(result$samples[, 1], main = "Autocorrelation plot",
     xlab = expression(y[1]), ylab = "Autocorrelation")
acf(result$samples[, 2], main = "Autocorrelation plot",
     xlab = expression(y[2]), ylab = "Autocorrelation")
```

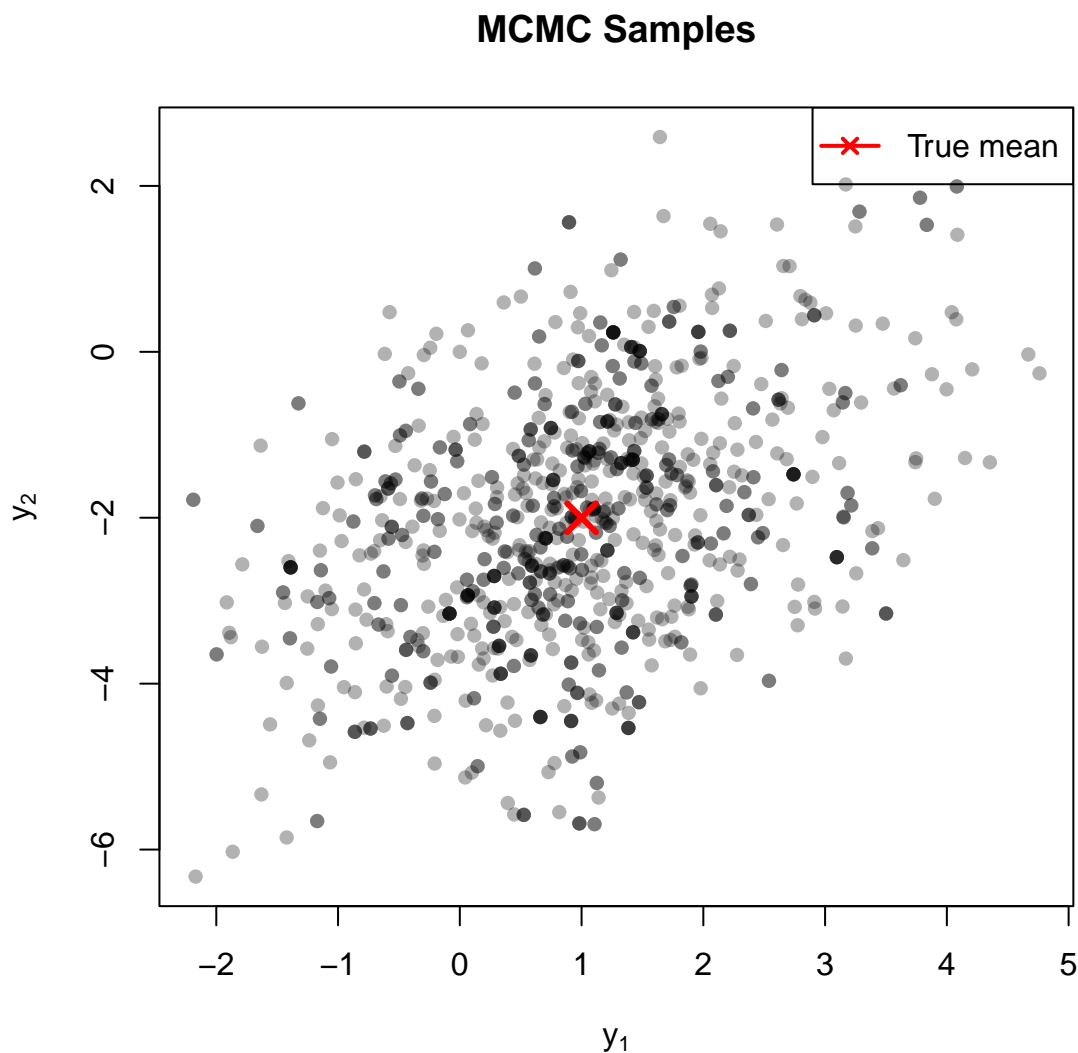
Autocorrelation plot



Autocorrelation plot



```
# Scatter plot of samples
plot(result$samples[, 1], result$samples[, 2],
     pch = 16, col = rgb(0, 0, 0, 0.3),
     xlab = expression(y[1]), ylab = expression(y[2]),
     main = "MCMC Samples")
points(target_mean[1], target_mean[2], col = "red", pch = 4, cex = 2, lwd = 3)
legend("topright", legend = "True mean", col = "red", pch = 4, lwd = 2)
```



Example 2: Bayesian Inference for Normal Data

Next, we consider an example involving Bayesian inference for the mean and variance of normal data.

Model Setup

Suppose we observe iid data $w_1, \dots, w_n \sim N(\mu, \sigma^2)$ and want to infer μ and σ^2 . Let us denote (w_1, \dots, w_n) by w . We use:

- Prior on μ : $\mu \sim N(\mu_0, \tau_0^2)$
- Prior on σ^2 : $\sigma^2 \sim \text{Inverse-Gamma}(\alpha_0, \beta_0)$

The log-posterior (up to a constant) is:

$$\log p(\mu, \sigma^2 | w) \propto -\frac{n}{2} \log(\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^n (w_i - \mu)^2 - \frac{(\mu - \mu_0)^2}{2\tau_0^2} - (\alpha_0 + 1) \log(\sigma^2) - \frac{\beta_0}{\sigma^2},$$

and the target density to sample from is $\psi(\mu, \sigma^2) = p(\mu, \sigma^2 | w)$.

```

# Generate data
set.seed(42)
true_mu <- 5
true_sigma <- 2
n <- 100
w <- rnorm(n, mean = true_mu, sd = true_sigma)

cat("True parameters:\n")

#> True parameters:
cat("  mu =", true_mu, "\n")

#>  mu = 5
cat("  sigma =", true_sigma, "\n")

#>  sigma = 2
cat("  sigma^2 =", true_sigma^2, "\n\n")

#>  sigma^2 = 4
cat("Sample statistics:\n")

#> Sample statistics:
cat("  mean =", round(mean(w), 3), "\n")

#>  mean = 5.065
cat("  sd =", round(sd(w), 3), "\n")

#>  sd = 2.083

# Define log-posterior
log_target <- function(par, x, mu0, tau0, alpha0, beta0) {
  mu <- par[1]
  sigma2 <- par[2]

  # Check constraint
  if (sigma2 <= 0) return(-Inf)

  n <- length(x)
  SSE <- sum((x - mu)^2)
  val <- -(n/2) * log(sigma2) - SSE / (2 * sigma2)
  val <- val - (mu - mu0)^2 / (2 * tau0^2)
  val <- val - (alpha0 + 1) * log(sigma2) - beta0 / sigma2

  return(val)
}

# Hyperparameters (weakly informative priors)
mu0 <- 0      # prior mean for mu
tau0 <- 10    # prior sd for mu
alpha0 <- 2.01 # shape parameter for inverse-gamma
beta0 <- 1.01 # scale parameter for inverse-gamma

# Run geomc

```



```

set.seed(3)
result <- geomc(
  logp = log_target,
  initial = c(0, 1), # Starting at  $\mu=0$ ,  $\sigma^2=1$ 
  n.iter = 1000,
  x = w,
  mu0 = mu0,
  tau0 = tau0,
  alpha0 = alpha0,
  beta0 = beta0
)
burnin<-50

```

Trace plots

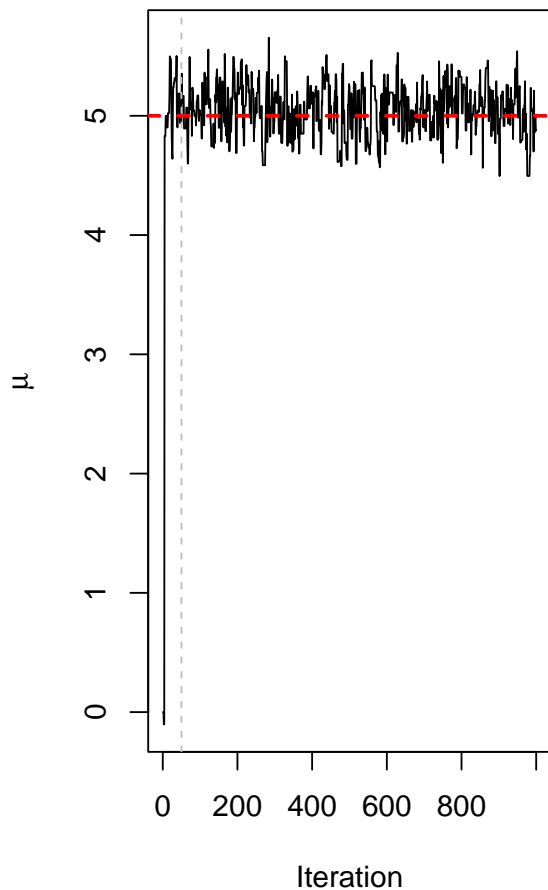
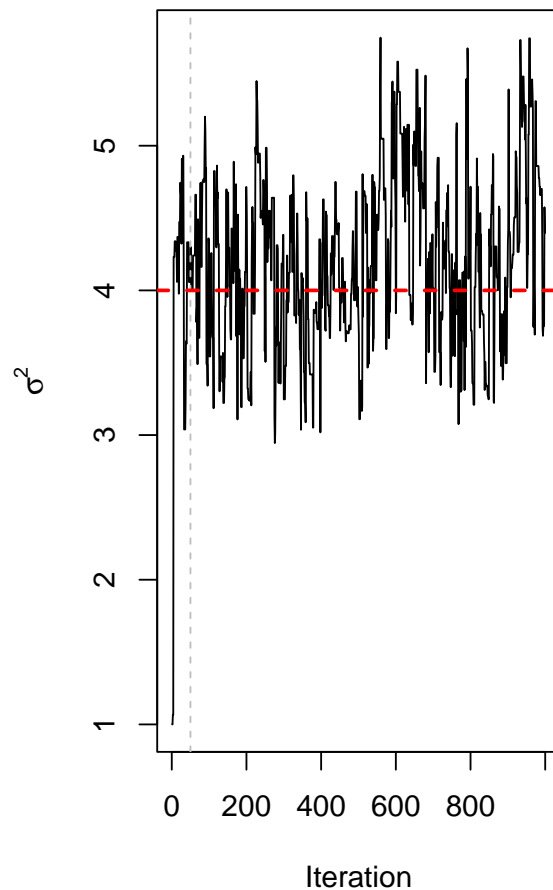
```

par(mfrow = c(1, 2))

# Trace plots
plot(result$samples[, 1], type = "l",
      main = expression(paste("Trace plot: ", mu)),
      ylab = expression(mu), xlab = "Iteration")
abline(v = burnin, col = "gray", lty = 2)
abline(h = true_mu, col = "red", lty = 2, lwd = 2)

plot(result$samples[, 2], type = "l",
      main = expression(paste("Trace plot: ", sigma^2)),
      ylab = expression(sigma^2), xlab = "Iteration")
abline(v = burnin, col = "gray", lty = 2)
abline(h = true_sigma^2, col = "red", lty = 2, lwd = 2)

```

Trace plot: μ Trace plot: σ^2 

Posterior Summaries

```
burnin <- 50
# Remove burn-in (first 500 samples)
samples_post_burnin <- result$samples[!(1:burnin), ]

# Posterior mean estimates
mu_post_mean <- mean(samples_post_burnin[, 1])
sigma2_post_mean <- mean(samples_post_burnin[, 2])
sigma_post_mean <- mean(sqrt(samples_post_burnin[, 2]))

cat("Posterior mean estimates:\n")

#> Posterior mean estimates:
cat("  mu:", round(mu_post_mean, 3), "\n")

#>  mu: 5.046
cat("  sigma^2:", round(sigma2_post_mean, 3), "\n")
```

```

#>   sigma^2: 4.22
cat("   sigma:", round(sigma_post_mean, 3), "\n\n")

#>   sigma: 2.049
# 95% credible intervals
mu_ci <- quantile(samples_post_burnin[, 1], c(0.025, 0.975))
sigma2_ci <- quantile(samples_post_burnin[, 2], c(0.025, 0.975))

cat("95% Credible Intervals:\n")

#> 95% Credible Intervals:
cat("   mu: [", round(mu_ci[1], 3), ",", round(mu_ci[2], 3), "]\n")

#>   mu: [ 4.599 , 5.451 ]
cat("   sigma^2: [", round(sigma2_ci[1], 3), ",", round(sigma2_ci[2], 3), "]\n")

#>   sigma^2: [ 3.222 , 5.459 ]

```

Visualizing the Posterior

```

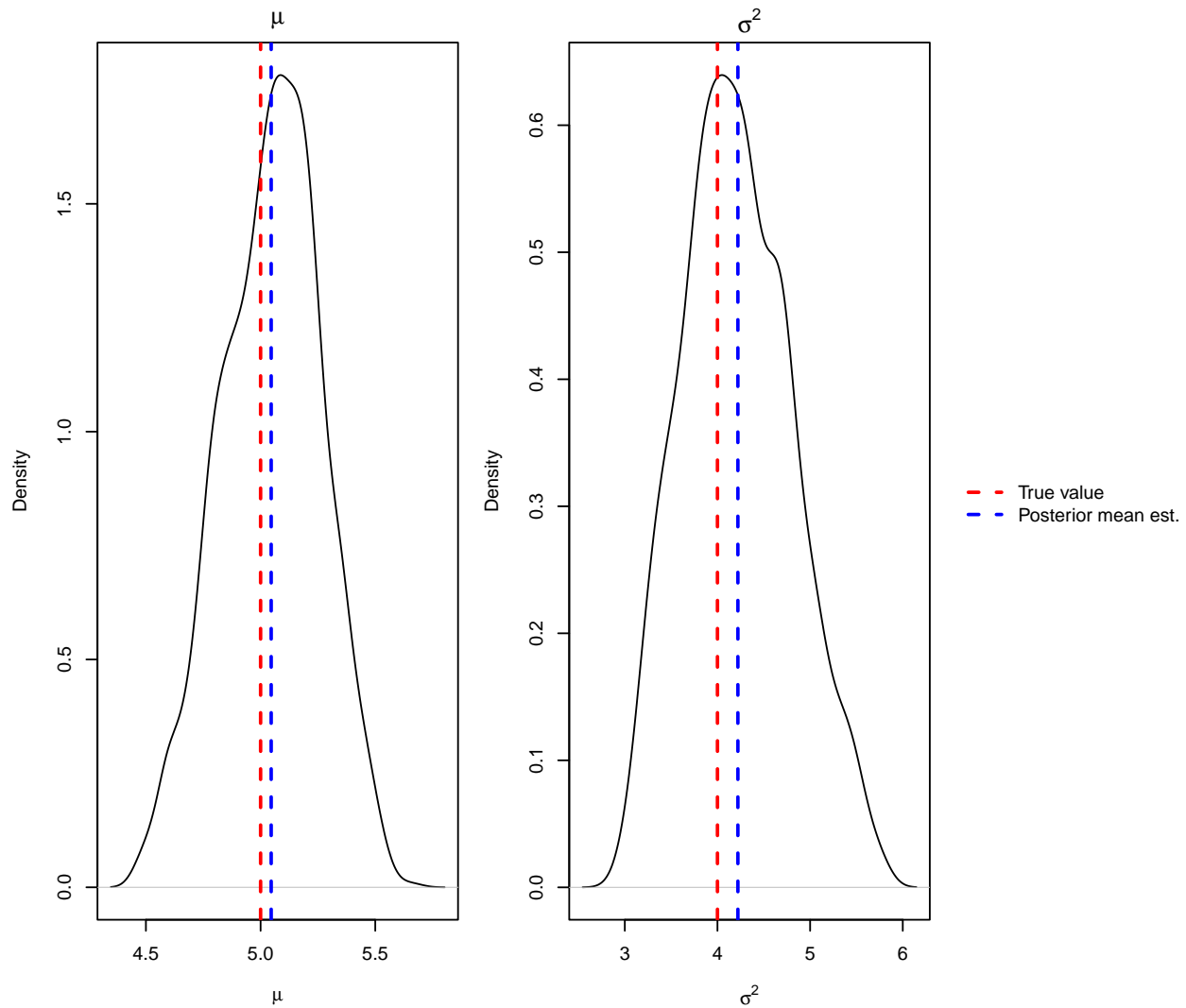
layout(matrix(c(1,2,3), nrow = 1), widths = c(4, 4, 2))
par(mar = c(4, 4, 2, 1))

plot(density(samples_post_burnin[,1]), main=expression(mu), xlab=expression(mu))
abline(v = true_mu, col="red", lty=2, lwd=2)
abline(v = mu_post_mean, col="blue", lty=2, lwd=2)

plot(density(samples_post_burnin[,2]), main=expression(sigma^2), xlab=expression(sigma^2))
abline(v = true_sigma^2, col="red", lty=2, lwd=2)
abline(v = sigma2_post_mean, col="blue", lty=2, lwd=2)

par(mar = c(0,0,0,0))
plot.new()
legend("center", legend=c("True value","Posterior mean est."),
      col=c("red","blue"), lty=2, lwd=2, bty="n")

```



Understanding the Output

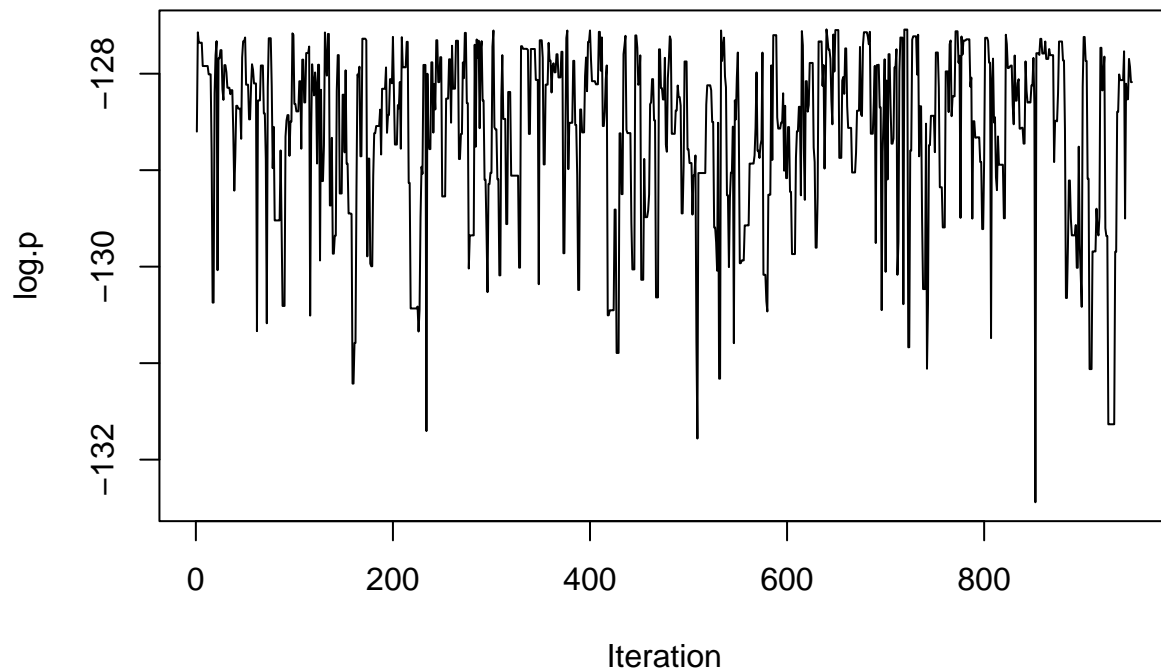
The `geomc()` function returns a list with these components:

```
#- **samples**: Matrix of MCMC samples (rows = iterations, columns = parameters)
head(result$samples)
```

```
#>           [,1]      [,2]
#> [1,]  0.0000000  1.000000
#> [2,]  0.0000000  1.000000
#> [3,] -0.1030793  1.066395
#> [4,] -0.1030793  1.066395
#> [5,]  4.8300905  4.232190
#> [6,]  4.8300905  4.232190
```

```
#- **log.p**: Log-density values at each sample
# Trace plot of log-density values
plot(result$log.p[-(1:burnin)], type = "l",
      main = "Trace plot of log-density values",
      ylab = "log.p", xlab = "Iteration")
```

Trace plot of log-density values



```
#- **acceptance.rate**: Proportion of accepted proposals  
cat("\nAcceptance rate:", round(result$acceptance.rate, 3), "\n")
```

```
#>  
#> Acceptance rate: 0.577
```

```
#- **var.base**: Variance of the random walk base density used  
cat("Variance of the random walk base=", result$var.base, "\n")
```

```
#> Variance of the random walk base= 0.08850625
```

```
#- **mean.ap.tar**: Mean of the approximate target density  
cat("Mean of the approximate target:\n")
```

```
#> Mean of the approximate target:
```

```
print(result$mean.ap.tar)
```

```
#> [1] 5.062969 4.069538
```

```
#- **var.ap.tar**: Variance of the approximate target density  
cat("\nVariance of the approximate target:\n")
```

```
#>
```

```
#> Variance of the approximate target:
```

```
print(result$var.ap.tar)
```

```
#>           [,1]      [,2]  
#> [1,] 0.040678910 -0.000158106
```

```
#> [2,] -0.000158106  0.312415961
#- **model.case**: Which model configuration was used
cat("model configuration:\n")

#> model configuration:
print(result$model.case)

#> [1] "No user-specified values detected; using default settings for base and ap.tar."
#- **gaus**: Whether Gaussian densities for both base and ap.tar were assumed
cat("Whether Gaussian densities for both base and ap.tar were assumed=", result$gaus, "\n")

#> Whether Gaussian densities for both base and ap.tar were assumed= TRUE
#- **ind**: Whether independence was assumed for both base and ap.tar
cat("Whether independence was assumed for both base and ap.tar=", result$ind, "\n")

#> Whether independence was assumed for both base and ap.tar= FALSE
```

Next, we consider more examples to demonstrate different non-default settings and features of the `geomc` function, including:

1. **Custom base densities** - Specifying your own base density
2. **Custom approximate target densities** - Guiding proposals toward specific regions
3. **Bhattacharyya coefficient** - Providing a function to compute the Bhattacharyya coefficient when `gaus` is `FALSE` (for example, when the target is a discrete distribution)
4. **Importance sampling** - Alternative to **3.** and numerical integration to compute the Bhattacharyya coefficient when `gaus` is `FALSE`
5. **Mixture models** - Sampling from multimodal target distributions
6. **Discrete distributions** - Sampling from non-continuous targets
7. **Comparison with other methods** - Understanding `geomc`'s advantages over random walk Metropolis

We will revisit the **Example 2** later in this document to discuss some of these advanced features of the `geomc` function. Now, we consider some multi-modal target examples.

Example 3: Univariate Mixture of Normals

Let's start with a bimodal distribution: a 50-50 mixture of two normal distributions. In particular, the target density is

$$\psi(y) = 0.5 \cdot \phi_1(y; 0, 1) + 0.5 \cdot \phi_1(y; 10, 1.4^2).$$

This is a challenging target to sample from because the modes are well-separated.

Univariate Mixture of Normals: Run `geomc` with Default Settings

```
# Define the log-target
log_target_mixture <- function(y) {
  log(0.5 * dnorm(y) + 0.5 * dnorm(y, mean = 10, sd = 1.4))
}

# Run geomc with default settings
set.seed(3)
result <- geomc(
  logp = list(log.target = log_target_mixture),
  initial = 0,
  n.iter = 1000
)
```

Visualizing the Results

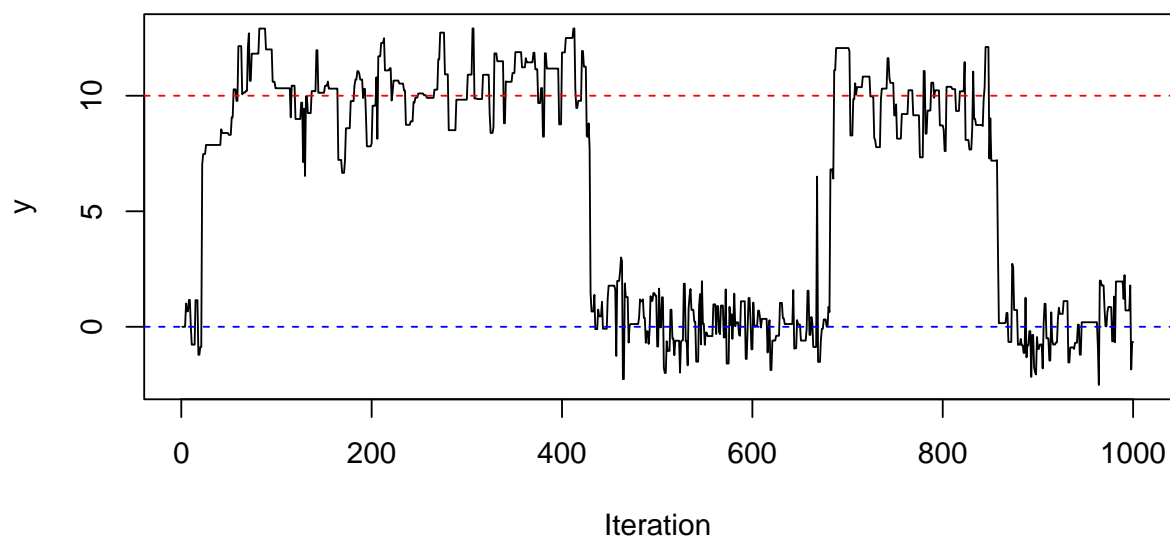
```
par(mfrow = c(2, 1))

# Trace plot
plot(result$samples, type = "l",
     main = "Trace plot: Mixture of Normals",
     ylab = "y", xlab = "Iteration")
abline(h = c(0, 10), col = c("blue", "red"), lty = 2)

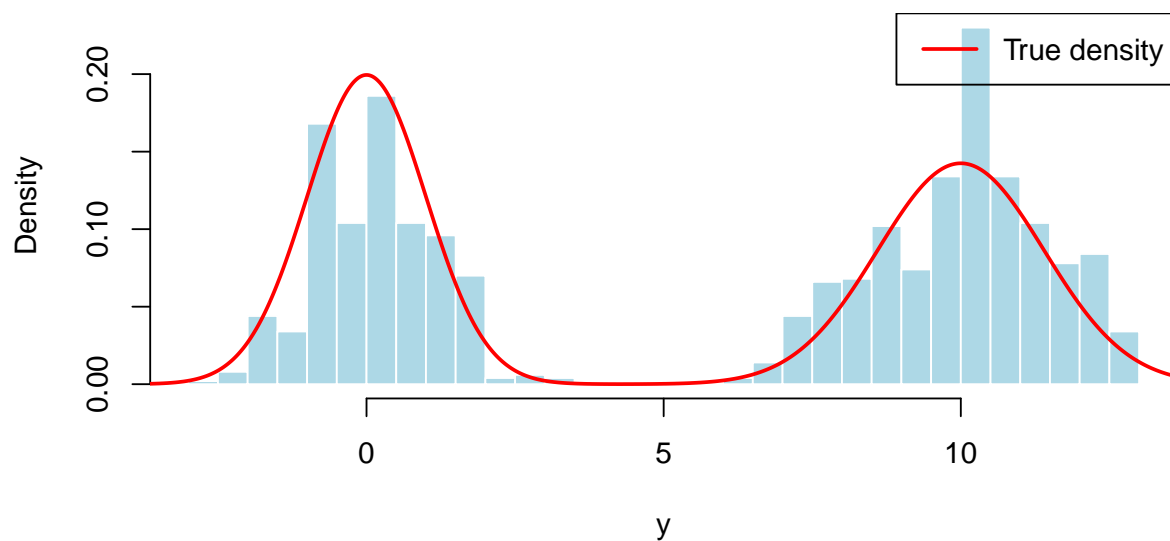
# Histogram with true density overlay
hist(result$samples, breaks = 50, freq = FALSE,
     main = "Posterior samples vs True density",
     xlab = "y", col = "lightblue", border = "white")

# Add true density
y_grid <- seq(-5, 15, length.out = 1000)
true_density <- 0.5 * dnorm(y_grid) + 0.5 * dnorm(y_grid, mean = 10, sd = 1.4)
lines(y_grid, true_density, col = "red", lwd = 2)
legend("topright", legend = "True density", col = "red", lwd = 2)
```

Trace plot: Mixture of Normals



Posterior samples vs True density



Observation: Even with default settings, `geomc` successfully explores both modes!

Univariate Mixture of Normals: Run `geomc` with Custom Random Walk Base Density

Now let's explicitly specify a random walk base density. This gives us more control over the proposal mechanism. In particular, we use a random walk proposal density $\phi_1(y; x, 4)$ as the base density $f(y|x)$.


```

set.seed(123)
result_custom <- geomc(
  logp = list(
    log.target = log_target_mixture,
    mean.base = function(x) x,          # Centered at current state
    var.base = function(x) 4           # Variance = 4
  ),
  initial = 0,
  n.iter = 1000
)
#Note the variance of the random walk base density of the default setting
cat("Variance of the default geomc =", result$var.base, "\n")

```

```
#> Variance of the default geomc = 5.6644
```

```
cat("The acceptance rate of the default geomc:\n")
```

```
#> The acceptance rate of the default geomc:
```

```
result$acceptance.rate
```

```
#> [1] 0.37
```

```
cat("The acceptance rate of geomc with the custom base density:\n")
```

```
#> The acceptance rate of geomc with the custom base density:
```

```
result_custom$acceptance.rate
```

```
#> [1] 0.47
```

Note that the scale of the default random walk base is higher than that of the custom random walk base resulting in lower acceptance rates.

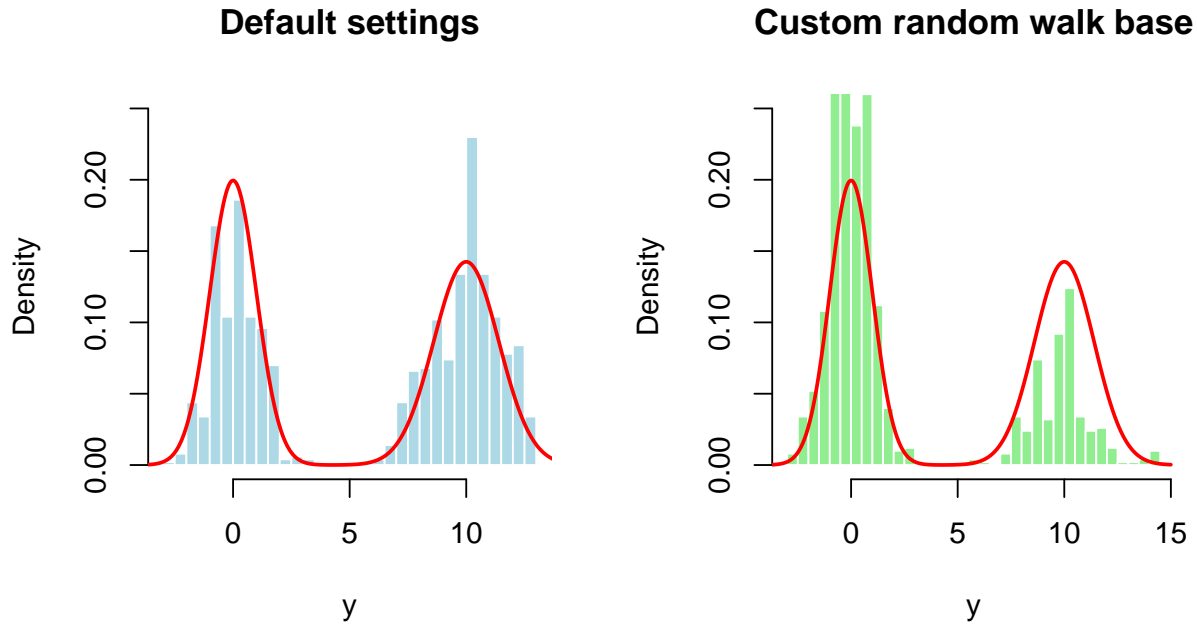
```

par(mfrow = c(1, 2))

hist(result$samples, breaks = 50, freq = FALSE,
     main = "Default settings",
     xlab = "y", col = "lightblue", border = "white", ylim = c(0, 0.25))
lines(y_grid, true_density, col = "red", lwd = 2)

hist(result_custom$samples, breaks = 50, freq = FALSE,
     main = "Custom random walk base",
     xlab = "y", col = "lightgreen", border = "white", ylim = c(0, 0.25))
lines(y_grid, true_density, col = "red", lwd = 2)

```



Univariate Mixture of Normals: Run geomc with Informed Approximate Targets

We can dramatically improve performance by providing approximate target densities that match the true target structure. We will use two approximate densities $\mathcal{G} = \{g_1, g_2\}$, where $g_1(y) = \phi(y; 0, 1)$ and $g_2(y) = \phi(y; 10, 1.4^2)$. Thus, the set of approximate densities, $\mathcal{G} = \{g_1, g_2\}$ is a set of two normal densities centered at the different local modes of the target.

```
set.seed(123)
result_informed <- geomc(
  logp = list(
    log.target = log_target_mixture,
    # Specify the two densities g_1 and g_2
    mean.ap.tar = function(x) c(0, 10), # Means of the two densities
    var.ap.tar = function(x) c(1, 1.4^2) # Variances of the two densities
  ),
  initial = 0,
  n.iter = 1000
)
```

Note that, in contrast to all earlier examples in which a single approximate target density is used by default, this case enforces two approximate Gaussian target densities.

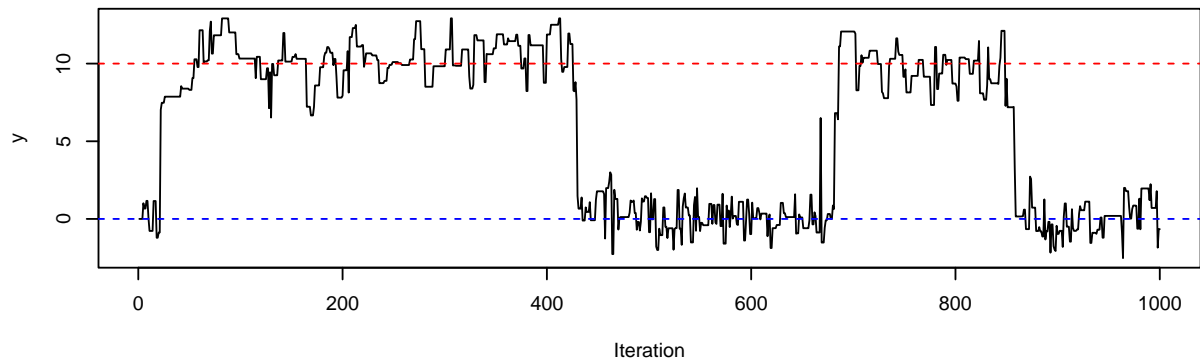
```
par(mfrow = c(3, 1))

plot(result$samples, type = "l", main = "Default settings", ylab = "y",
      xlab = "Iteration")
abline(h = c(0, 10), col = c("blue", "red"), lty = 2)

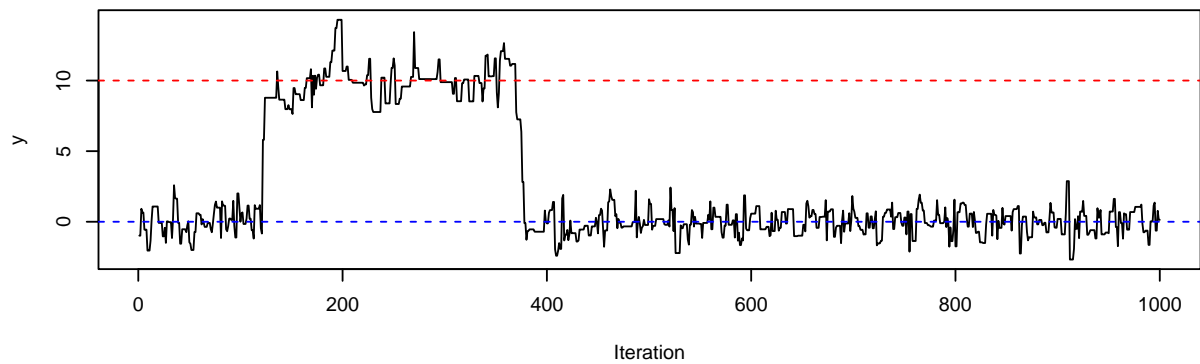
plot(result_custom$samples, type = "l", main = "Custom random walk",
      ylab = "y", xlab = "Iteration")
abline(h = c(0, 10), col = c("blue", "red"), lty = 2)
```

```
plot(result_informed$samples, type = "l", main = "Informed approximate targets",
     ylab = "y", xlab = "Iteration")
abline(h = c(0, 10), col = c("blue", "red"), lty = 2)
```

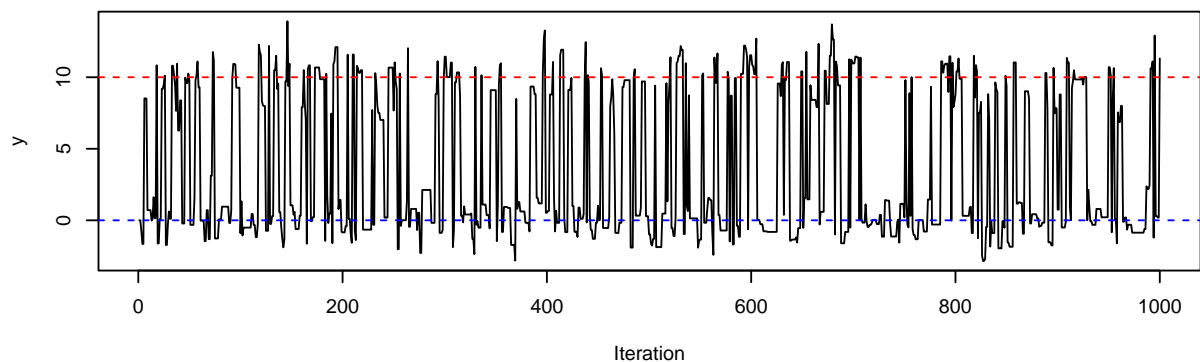
Default settings



Custom random walk



Informed approximate targets



Key insight: With informed approximate targets, the sampler moves between modes very efficiently!

Univariate Mixture of Normals: Run geomc with Informed Approximate Targets along with a Custom Random Walk Base Density

Users can provide both one or more approximate target densities, as well as a custom base density in `geomc`. Since the base and approximate target densities are normal, we can simply specify these by their means and variances.

```
set.seed(123)
result_informed_apbase <- geomc(
  logp = list(
    log.target = log_target_mixture,
    mean.base = function(x) x, # specify the base mean
    var.base = function(x) 4, # specify the base variance
    # Specify the two densities g_1 and g_2
    mean.ap.tar = function(x) c(0, 10), # Means of the two densities
    var.ap.tar = function(x) c(1, 1.4^2) # Variances of the two densities
  ),
  initial = 0,
  n.iter = 1000
)
# Compare the values of model.case
cat("The model configuration under default setting=\n")

#> The model configuration under default setting=
print(result$model.case)

#> [1] "No user-specified values detected; using default settings for base and ap.tar."
cat("The model configuration under result_custom\n")

#> The model configuration under result_custom
print(result_custom$model.case)

#> [1] "No user-specified value detected; using the default setting for ap.tar."
cat("The model configuration under result_informed\n")

#> The model configuration under result_informed
print(result_informed$model.case)

#> [1] "No user-specified value detected; using the default setting for base."
cat("The model configuration under result_informed_apbase\n")

#> The model configuration under result_informed_apbase
print(result_informed_apbase$model.case)

#> [1] "User-specified settings for both base and ap.tar are being used."
par(mfrow = c(3, 1))

plot(result$samples, type = "l", main = "Default settings", ylab = "y",
      xlab = "Iteration")
abline(h = c(0, 10), col = c("blue", "red"), lty = 2)

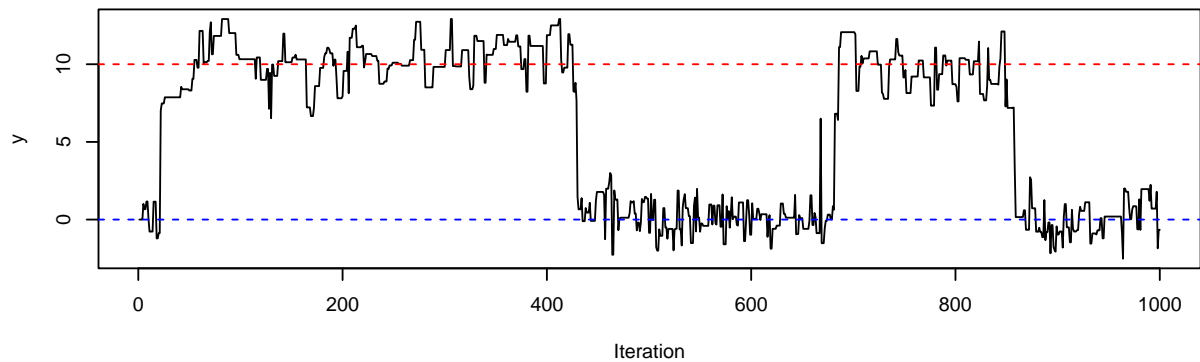
plot(result_informed$samples, type = "l", main = "Informed approximate targets",
```

```

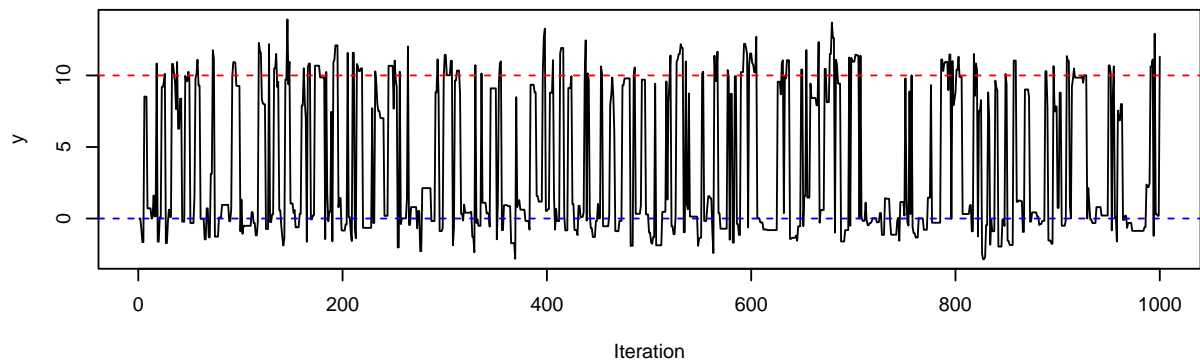
ylab = "y", xlab = "Iteration")
abline(h = c(0, 10), col = c("blue", "red"), lty = 2)
plot(result_informed_apbase$samples, type = "l",
     main = "Informed approximate targets and custom base",
     ylab = "y", xlab = "Iteration")
abline(h = c(0, 10), col = c("blue", "red"), lty = 2)

```

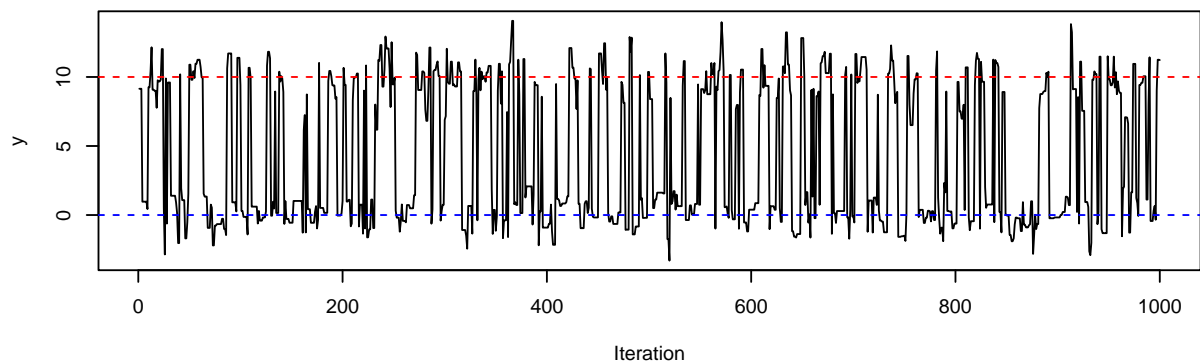
Default settings



Informed approximate targets



Informed approximate targets and custom base



Univariate Mixture of Normals: Run geomc with other choices of eps

geomc also allows tuning the parameter `eps`. As mentioned in Roy (2024), `eps` controls the degree of perturbation of the base density along the directions defined by the approximate targets. In particular, larger `eps` puts less weight on the base density. On the other hand, as $\text{eps} \downarrow 0$, the transition density of the geometric MH chain converges to that of the MH chain with base proposal density. By default, `eps` is fixed at the moderate value of 0.5, but it can be set at any other proper fraction.

```
set.seed(123)
result_informed_apbase_eps_large <- geomc(
  logp = list(
    log.target = log_target_mixture,
    mean.base = function(x) x,
    var.base = function(x) 4,
    # Specify the two densities g_1 and g_2
    mean.ap.tar = function(x) c(0, 10), # Means of the two densities
    var.ap.tar = function(x) c(1, 1.4^2) # Variances of the two densities
  ),
  initial = 0,
  n.iter = 1000,
  eps = 0.9
)

set.seed(123)
result_informed_apbase_eps_small <- geomc(
  logp = list(
    log.target = log_target_mixture,
    mean.base = function(x) x,
    var.base = function(x) 4,
    # Specify the two densities g_1 and g_2
    mean.ap.tar = function(x) c(0, 10), # Means of the two densities
    var.ap.tar = function(x) c(1, 1.4^2) # Variances of the two densities
  ),
  initial = 0,
  n.iter = 1000,
  eps = 0.01
)

set.seed(123)
result_rwm_mix <- geommc:::rwm(
  log_target_mixture,
  initial = 0,
  n_iter = 1000,
  sig = 4,
  return_sample = TRUE
)
cat("Geometric MCMC:\n")

#> Geometric MCMC:
cat("With default eps:\n")

#> With default eps:
cat("  Acceptance rate:", round(result_informed_apbase$acceptance.rate, 3), "\n")

#>   Acceptance rate: 0.58
```

```

cat("With eps =0.9:\n")

#> With eps =0.9:
cat("  Acceptance rate:", round(result_informed_apbase_eps_large$acceptance.rate, 3), "\n")

#>   Acceptance rate: 0.656
cat("With eps=0.01:\n")

#> With eps=0.01:
cat("  Acceptance rate:", round(result_informed_apbase_eps_small$acceptance.rate, 3), "\n")

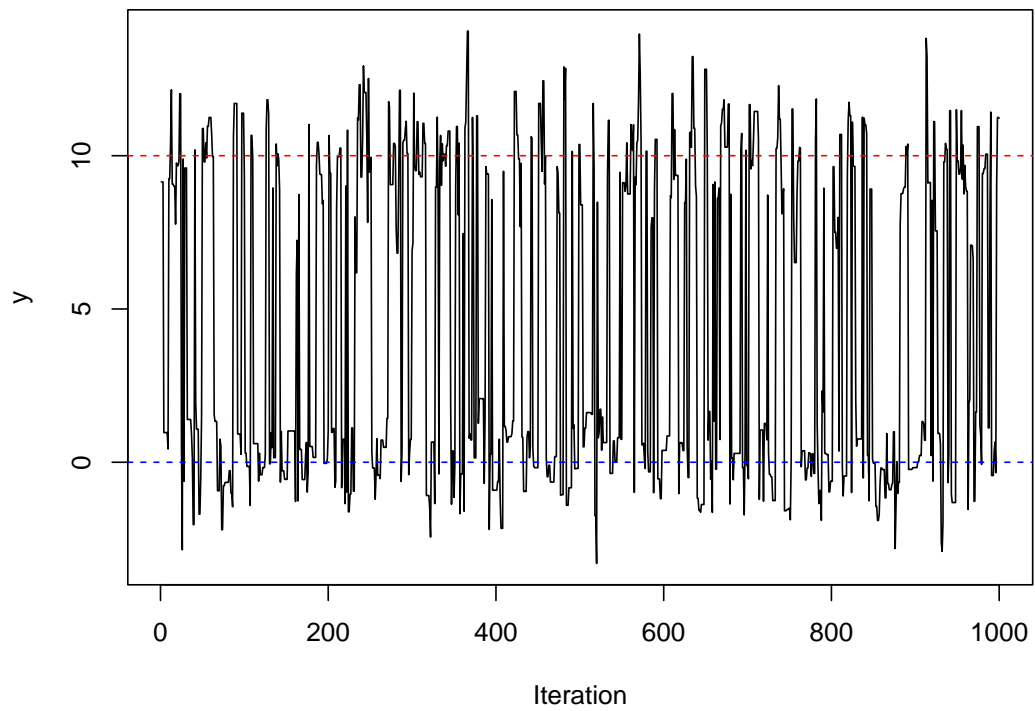
#>   Acceptance rate: 0.524
cat("Random Walk Metropolis:\n")

#> Random Walk Metropolis:
cat("  Acceptance rate:", round(result_rwm_mix$acceptance_rate, 3), "\n")

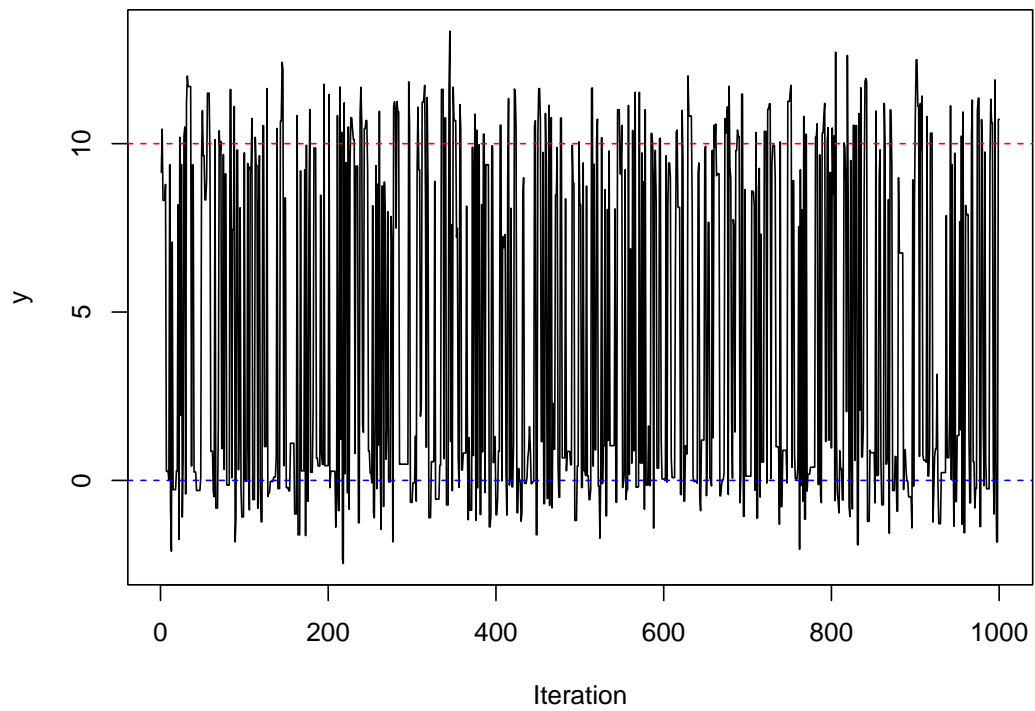
#>   Acceptance rate: 0.538
par(mfrow = c(2, 1))
plot(result_informed_apbase$samples, type = "l",
     main = "Informed approximate targets and custom base (Default eps)",
     ylab = "y", xlab = "Iteration")
abline(h = c(0, 10), col = c("blue", "red"), lty = 2)
plot(result_informed_apbase_eps_large$samples, type = "l",
     main = "Informed approximate targets and custom base (eps=0.9)",
     ylab = "y", xlab = "Iteration")
abline(h = c(0, 10), col = c("blue", "red"), lty = 2)

```

Informed approximate targets and custom base (Default eps)

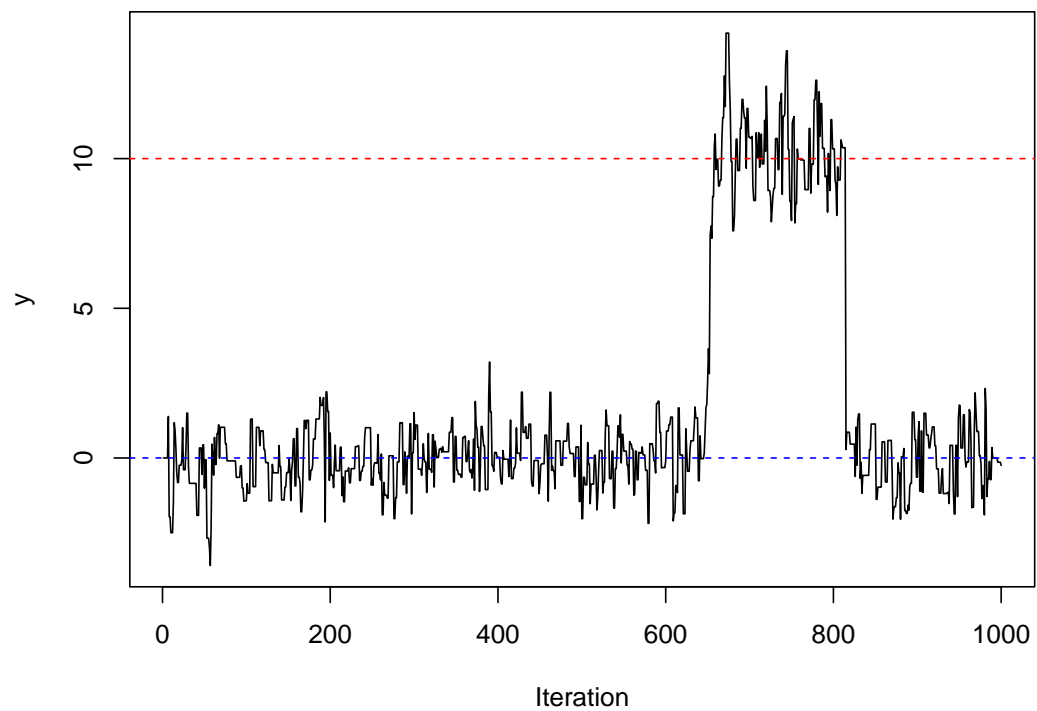


Informed approximate targets and custom base (eps=0.9)

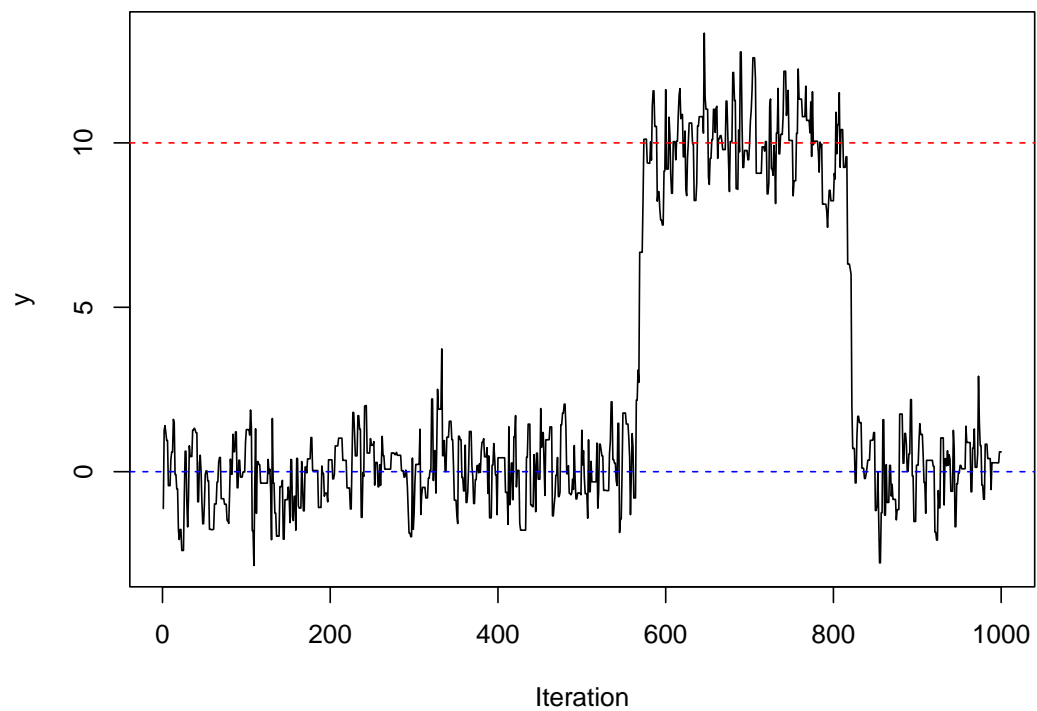



```
plot(result_informed_apbase_eps_small$samples, type = "l",
     main = "Informed approximate targets and custom base (eps=0.01)",
     ylab = "y", xlab = "Iteration")
abline(h = c(0, 10), col = c("blue", "red"), lty = 2)
plot(result_rwm_mix$samples, type = "l",
     main = "Random walk Matropolis",
     ylab = "y", xlab = "Iteration")
abline(h = c(0, 10), col = c("blue", "red"), lty = 2)
```

Informed approximate targets and custom base (eps=0.01)



Random walk Matropolis



We observe that `geomc` with a small `eps` value has similar performance as the random walk Metropolis (the MH chain with the base proposal density).

Univariate Mixture of Normals: Run `geomc` with another (non-Gaussian) Informed Approximate Target

So far we have used one or two Gaussian approximate target densities. Next, we consider a non-normal approximate target $g = \psi$, the target density itself. Since $\langle \sqrt{f}, \sqrt{g} \rangle$ is no more available in closed form, it is estimated by numerical integration.

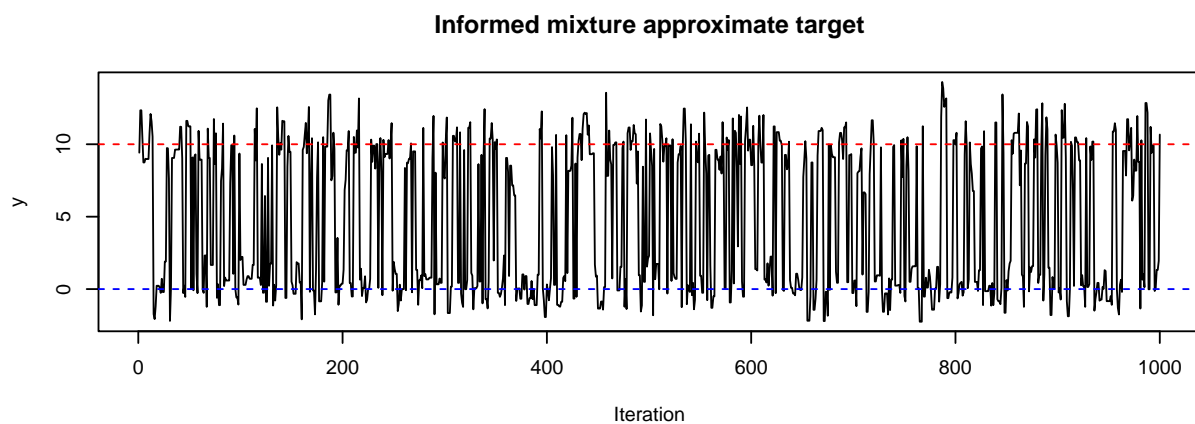
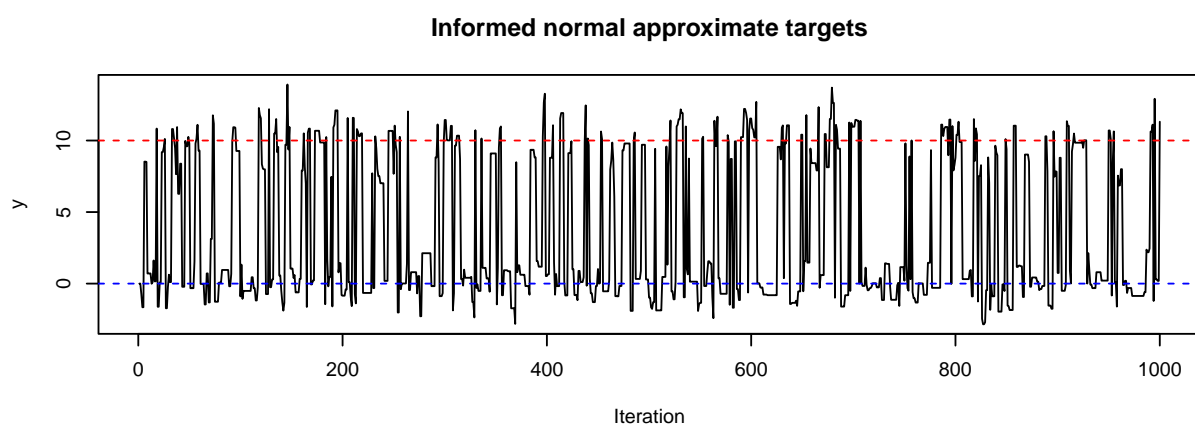
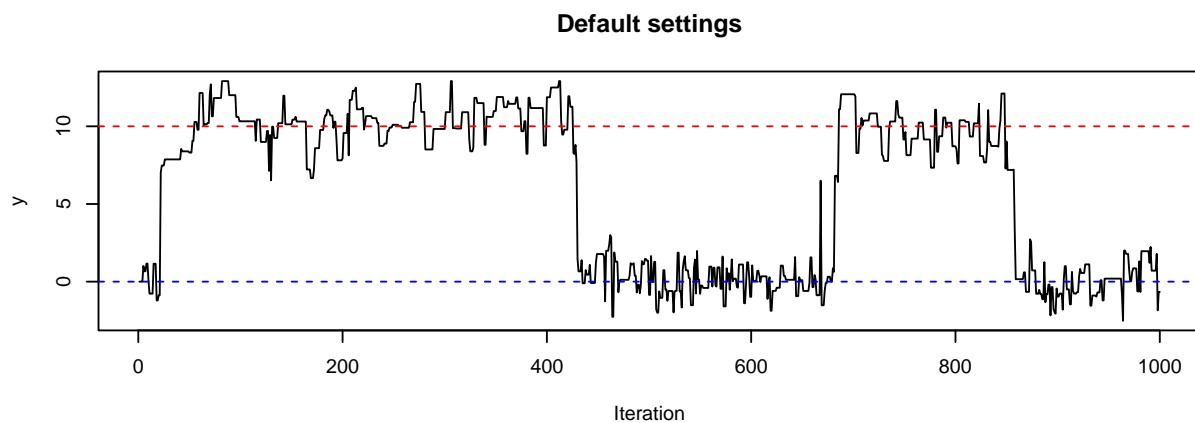
```
# Sampler that draws from the mixture density g
samp.ap.tar_mixture <- function(x, kk = 1) {
  component <- sample.int(2, 1, prob = c(0.5, 0.5))
  if (component == 1) {
    return(rnorm(1))
  } else {
    return(10 + 1.4 * rnorm(1))
  }
}
set.seed(123)
result_informed_another <- geomc(
  logp = list(
    log.target = log_target_mixture,
    dens.ap.tar=function(y,x) 0.5*dnorm(y)+0.5*dnorm(y,mean=10,sd=1.4),
    samp.ap.tar=samp.ap.tar_mixture
  ),
  initial = 0,
  n.iter = 1000,
  gaus = FALSE # Not using Gaussian assumption
)
```

Note that, the argument `gaus` is required to be set as `FALSE` (not the default value).

```
par(mfrow = c(3, 1))

plot(result$samples, type = "l", main = "Default settings",
     ylab = "y", xlab = "Iteration")
abline(h = c(0, 10), col = c("blue", "red"), lty = 2)

plot(result_informed$samples, type = "l",
     main = "Informed normal approximate targets",
     ylab = "y", xlab = "Iteration")
abline(h = c(0, 10), col = c("blue", "red"), lty = 2)
plot(result_informed_another$samples, type = "l",
     main = "Informed mixture approximate target",
     ylab = "y", xlab = "Iteration")
abline(h = c(0, 10), col = c("blue", "red"), lty = 2)
```



Univariate Mixture of Normals: Importance Sampling to estimate inner products for non-Gaussian densities

Instead of numerical integration to compute $\langle \sqrt{f}, \sqrt{g} \rangle$, we can use importance sampling. We can also specify a custom base density.

```
set.seed(123)
result_imp <- geomc(
```

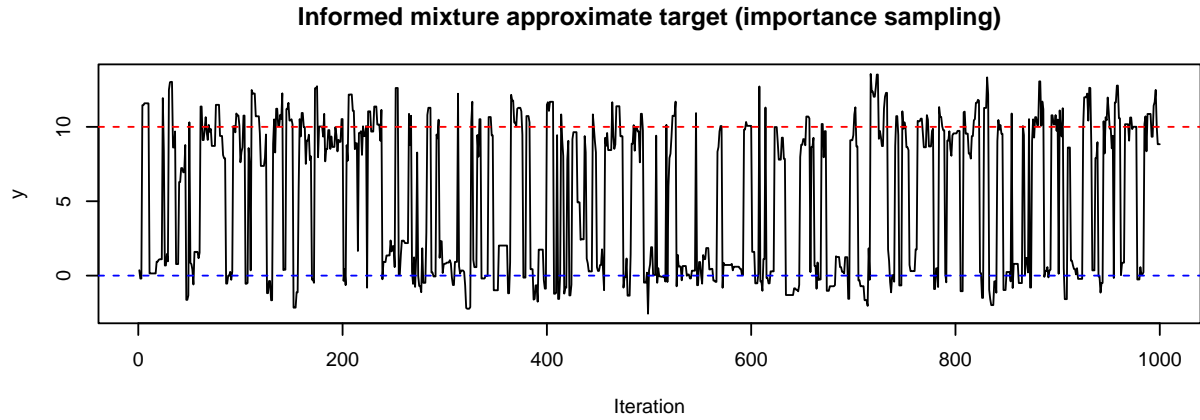
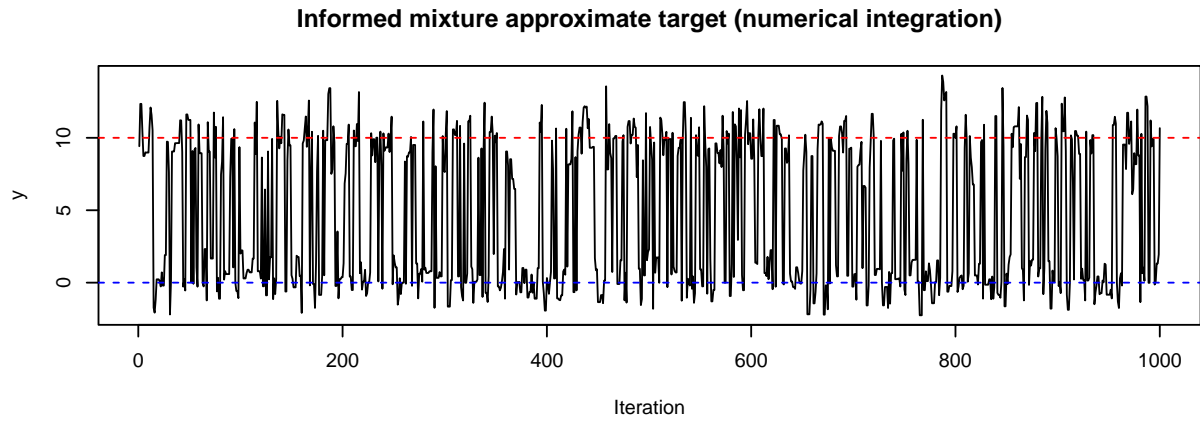
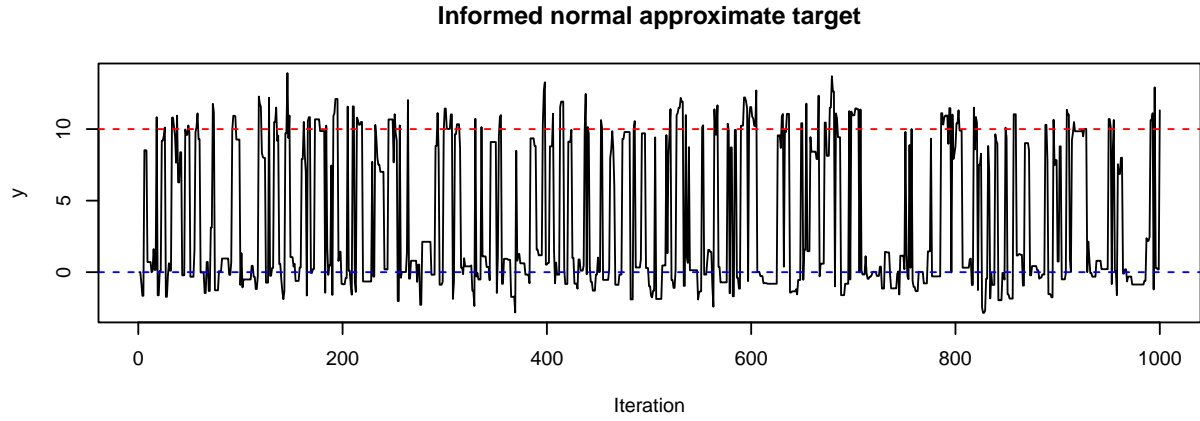
```

logp = list(
  log.target = log_target_mixture,
  dens.base = function(y, x) dnorm(y, mean = x, sd = 2),
  samp.base = function(x) x + 2 * rnorm(1),
  dens.ap.tar = function(y, x) 0.5 * dnorm(y) + 0.5 * dnorm(y, mean = 10, sd = 1.4),
  samp.ap.tar = samp.ap.tar_mixture
),
initial = 0,
n.iter = 1000,
gaus = FALSE, # Not using Gaussian assumption
imp = list(
  enabled = TRUE,
  n.samp = 50,
  samp.base = FALSE # Draw samples from approximate target density
),
show.progress = FALSE
)

par(mfrow = c(3, 1))

plot(result_informed$samples, type = "l",
     main = "Informed normal approximate target",
     ylab = "y", xlab = "Iteration")
abline(h = c(0, 10), col = c("blue", "red"), lty = 2)
plot(result_informed_another$samples, type = "l",
     main = "Informed mixture approximate target (numerical integration)",
     ylab = "y", xlab = "Iteration")
abline(h = c(0, 10), col = c("blue", "red"), lty = 2)
plot(result_imp$samples, type = "l",
     main = "Informed mixture approximate target (importance sampling)",
     ylab = "y", xlab = "Iteration")
abline(h = c(0, 10), col = c("blue", "red"), lty = 2)

```



As discussed in Roy (2024), use of a non-normal approximate target density results in extra run time due to computation of $\langle \sqrt{f}, \sqrt{g} \rangle$ by numerical integration or importance sampling in every iteration of the Markov chain, unless the Bhattacharyya coefficient is provided via the argument `bhat.coef`.

Example 4: Bivariate Mixture of Normals

Next, we consider a bivariate multi-modal target density. In particular, we consider the following bivariate mixture of normals:

$$\psi(y) = 0.5 \cdot \phi_2\left(y; \begin{pmatrix} 0 \\ 0 \end{pmatrix}, I\right) + 0.5 \cdot \phi_2\left(y; \begin{pmatrix} 10 \\ 10 \end{pmatrix}, 2I\right).$$

Bivariate Mixture of Normals: Run geomc with Informed Approximate Targets along with a Custom Random Walk Base Density

We first run `geomc` with the base density $f(y|x) = \phi_2(y; x, 2I)$ and two approximate denisites: $g_1(y) = \phi_2(y; \mu_1, \Sigma_1)$ and $g_2(y) = \phi_2(y; \mu_2, \Sigma_2)$. Thus, we use a random walk proposal as the base density and the set of approximate denisites, $\mathcal{G} = \{g_1, g_2\}$ is a set of two Gaussian densities centered at the different local modes of the target.

```
# Define log-target for bivariate mixture
log_target_mvnorm_mix <- function(x, mean1, Sigma1, mean2, Sigma2) {
  ldens1 <- geommc::ldens_mvnorm(x, mean1, Sigma1)
  ldens2 <- geommc::ldens_mvnorm(x, mean2, Sigma2)
  return(log(0.5 * exp(ldens1) + 0.5 * exp(ldens2)))
}

# Parameters for the two components
mean1 <- c(0, 0)
Sigma1 <- diag(2)
mean2 <- c(10, 10)
Sigma2 <- 2 * diag(2)

# Run geomc with informed approximate targets
set.seed(123)
result_biv <- geomc(
  logp = list(
    log.target = log_target_mvnorm_mix,
    mean.base = function(x) x,
    var.base = function(x) 2 * diag(2),
    mean.ap.tar = function(x) c(0, 0, 10, 10), # Means of both densities; can also be list(c(0, 0), c(10, 10))
    var.ap.tar = function(x) cbind(diag(2), 2 * diag(2)) # Covariances specified by cbind; can also be list(diag(2), 2 * diag(2))
  ),
  initial = c(5, 5), # Start between the modes
  n.iter = 1000,
  show.progress = FALSE,
  mean1 = mean1,
  Sigma1 = Sigma1,
  mean2 = mean2,
  Sigma2 = Sigma2
)

cat("Acceptance rate:", round(result_biv$acceptance.rate, 3), "\n")

#> Acceptance rate: 0.531

cat("\nSample means:\n")

#>
#> Sample means:
print(colMeans(result_biv$samples))

#> [1] 6.463765 6.352963
```

```
cat("\nExpected mean: (5, 5)\n")
```

```
#>
```

```
#> Expected mean: (5, 5)
```

Visualizing Bivariate Mixture Results

```
par(mfrow = c(2, 2))
```

```
# Trace plots
```

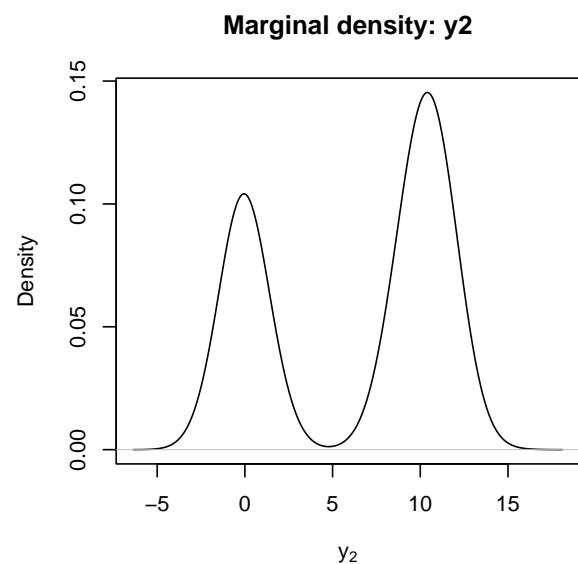
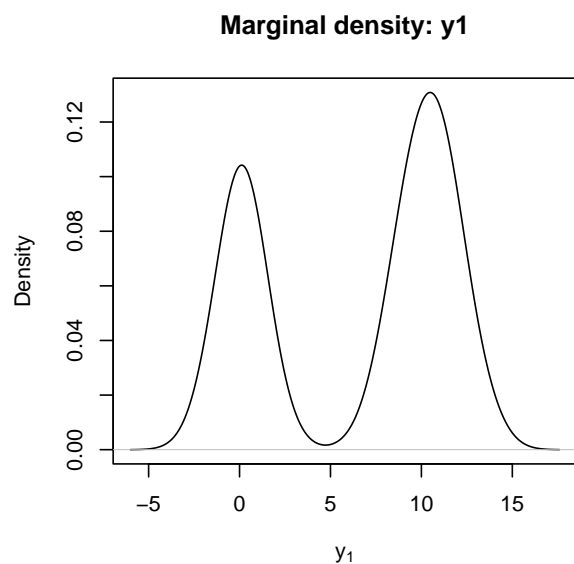
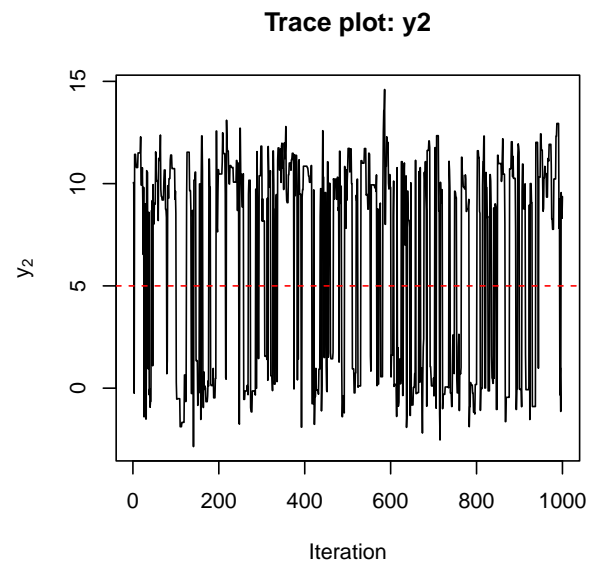
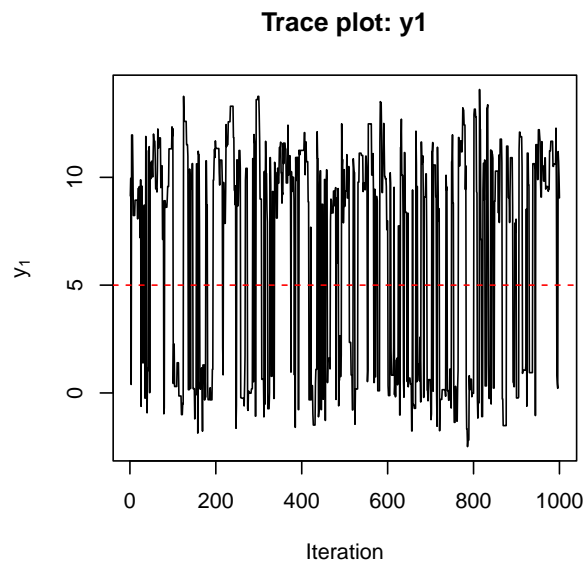
```
plot(result_biv$samples[, 1], type = "l",  
      main = "Trace plot: y1",  
      ylab = expression(y[1]), xlab = "Iteration")  
abline(h = 5, col = "red", lty = 2)
```

```
plot(result_biv$samples[, 2], type = "l",  
      main = "Trace plot: y2",  
      ylab = expression(y[2]), xlab = "Iteration")  
abline(h = 5, col = "red", lty = 2)
```

```
# Marginal densities
```

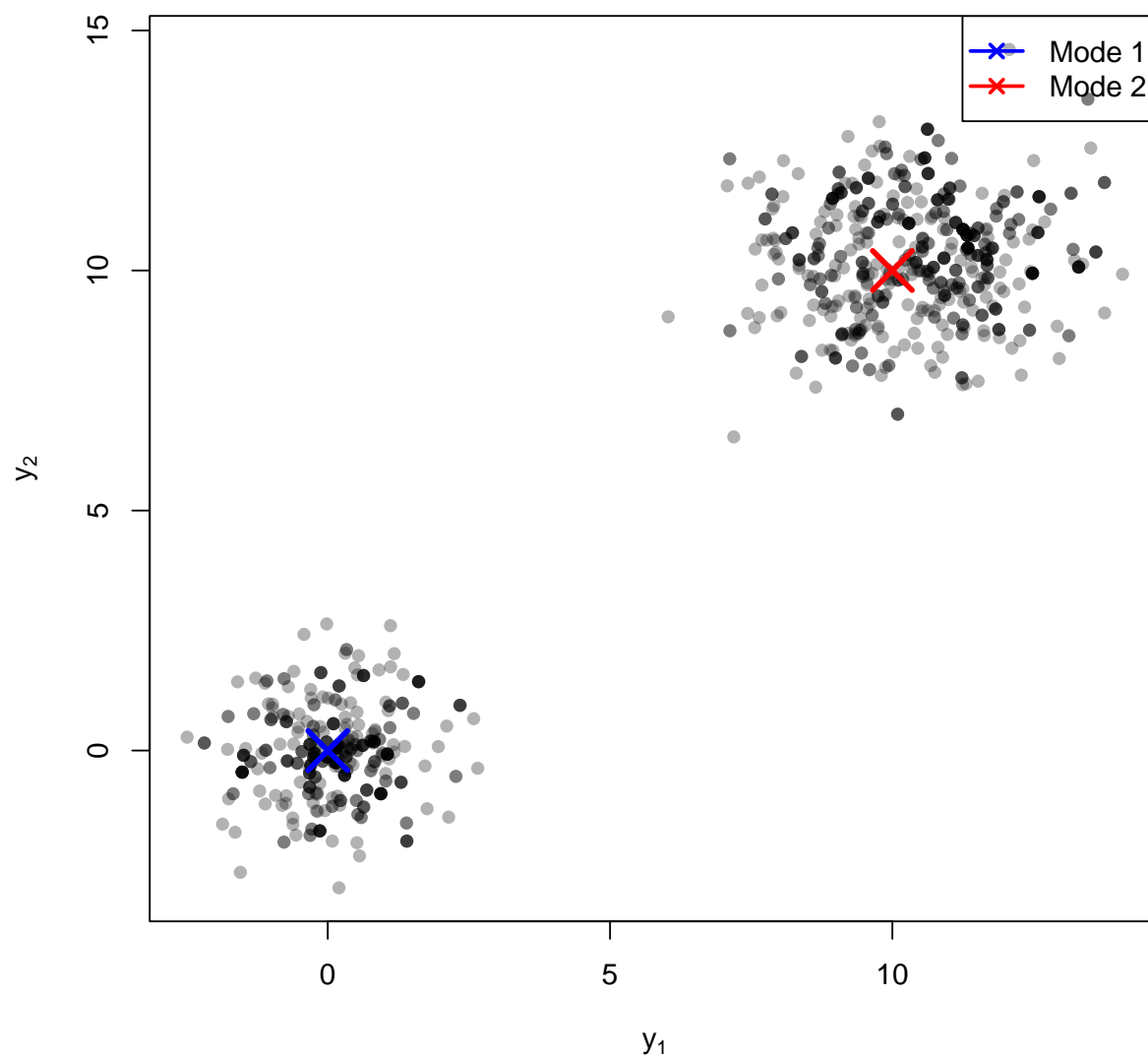
```
plot(density(result_biv$samples[, 1]),  
      main = "Marginal density: y1",  
      xlab = expression(y[1]))
```

```
plot(density(result_biv$samples[, 2]),  
      main = "Marginal density: y2",  
      xlab = expression(y[2]))
```

```
# Scatter plot
plot(result_biv$samples[, 1], result_biv$samples[, 2],
     pch = 16, col = rgb(0, 0, 0, 0.3),
     xlab = expression(y[1]), ylab = expression(y[2]),
     main = "Bivariate Mixture Samples")
points(mean1[1], mean1[2], col = "blue", pch = 4, cex = 3, lwd = 3)
points(mean2[1], mean2[2], col = "red", pch = 4, cex = 3, lwd = 3)
legend("topright",
     legend = c("Mode 1", "Mode 2"),
     col = c("blue", "red"), pch = 4, lwd = 2)
```

Bivariate Mixture Samples



Bivariate Mixture of Normals: Run geomc with a diffuse g

Next, we run `geomc` with the default base density and a diffuse choice of g , namely

$$g(y) = \phi_2\left(y; \begin{pmatrix} 0 \\ 0 \end{pmatrix}, 400I\right).$$

```
# Run geomc with diffuse g
set.seed(123)
result_biv_diffuse <- geomc(
  logp = list(
    log.target = log_target_mvnorm_mix,
    mean.ap.tar = function(x) c(0, 0),
    var.ap.tar = function(x) 400*diag(2)
```

```

),
  initial = c(5, 5), # Start between the modes
  n.iter = 1000,
  mean1 = mean1,
  Sigma1 = Sigma1,
  mean2 = mean2,
  Sigma2 = Sigma2
)

cat("Acceptance rate:", round(result_biv_diffuse$acceptance.rate, 3), "\n")

#> Acceptance rate: 0.502

cat("\nSample means:\n")

#>
#> Sample means:
print(colMeans(result_biv_diffuse$samples))

#> [1] 5.551886 3.507097

cat("\nExpected mean: (5, 5)\n")

#>
#> Expected mean: (5, 5)

```

Visualizing Bivariate Mixture Results

```

par(mfrow = c(2, 2))

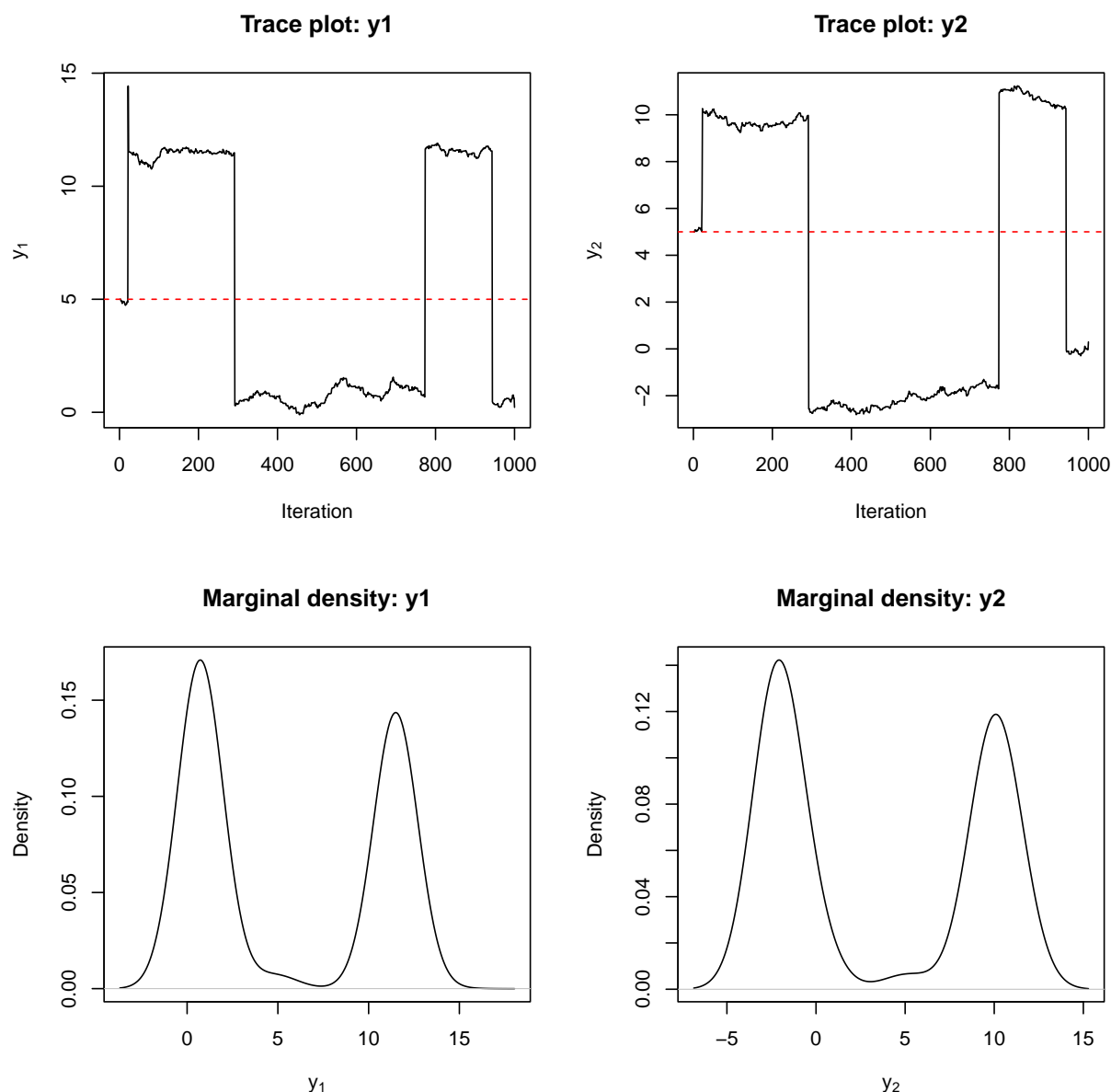
# Trace plots
plot(result_biv_diffuse$samples[, 1], type = "l",
     main = "Trace plot: y1",
     ylab = expression(y[1]), xlab = "Iteration")
abline(h = 5, col = "red", lty = 2)

plot(result_biv_diffuse$samples[, 2], type = "l",
     main = "Trace plot: y2",
     ylab = expression(y[2]), xlab = "Iteration")
abline(h = 5, col = "red", lty = 2)

# Marginal densities
plot(density(result_biv_diffuse$samples[, 1]),
     main = "Marginal density: y1",
     xlab = expression(y[1]))

plot(density(result_biv_diffuse$samples[, 2]),
     main = "Marginal density: y2",
     xlab = expression(y[2]))

```



We see that even with a diffuse density for g , `geomc` successfully moves between the two local modes.

Bivariate Mixture of Normals: Comparison with Random Walk Metropolis

Let's compare the performance of `geomc` with that of a standard random walk Metropolis algorithm on this bivariate mixture target. For this comparison, we use the same scale for the random walk Metropolis as the base random walk density used while running `geomc` with a custom base.

```
# Run random walk Metropolis
set.seed(123)
result_rwm <- geommc:::rwm(
  log_target = function(x) {
    log_target_mvnorm_mix(x, mean1, Sigma1, mean2, Sigma2)
  },
  initial = c(5, 5),
```

```

n_iter = 1000,
sig = 2,
return_sample = TRUE
)

cat("Random Walk Metropolis:\n")

#> Random Walk Metropolis:
cat("  Acceptance rate:", round(result_rwm$acceptance_rate, 3), "\n")

#>   Acceptance rate: 0.007
cat("  Sample means:", round(colMeans(result_rwm$samples), 3), "\n\n")

#>   Sample means: 9.058 12.727
cat("Geometric MCMC:\n")

#> Geometric MCMC:
cat("  Acceptance rate:", round(result_biv$acceptance.rate, 3), "\n")

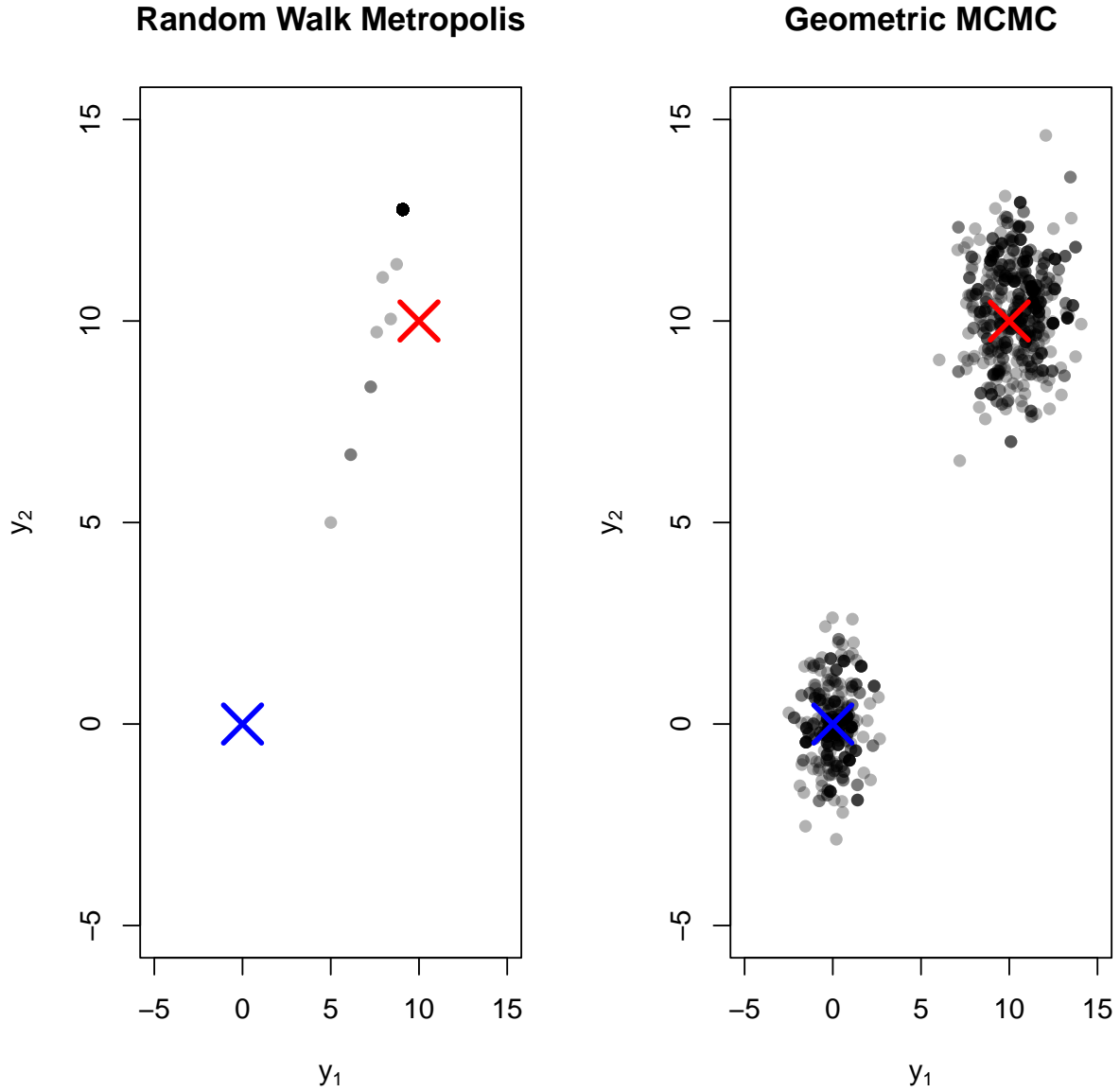
#>   Acceptance rate: 0.531
cat("  Sample means:", round(colMeans(result_biv$samples), 3), "\n")

#>   Sample means: 6.464 6.353
par(mfrow = c(1, 2))

# Random Walk Metropolis
plot(result_rwm$samples[, 1], result_rwm$samples[, 2],
     pch = 16, col = rgb(0, 0, 0, 0.3),
     xlab = expression(y[1]), ylab = expression(y[2]),
     main = "Random Walk Metropolis",
     xlim = c(-5, 15), ylim = c(-5, 15))
points(mean1[1], mean1[2], col = "blue", pch = 4, cex = 3, lwd = 3)
points(mean2[1], mean2[2], col = "red", pch = 4, cex = 3, lwd = 3)

# Geometric MCMC
plot(result_biv$samples[, 1], result_biv$samples[, 2],
     pch = 16, col = rgb(0, 0, 0, 0.3),
     xlab = expression(y[1]), ylab = expression(y[2]),
     main = "Geometric MCMC",
     xlim = c(-5, 15), ylim = c(-5, 15))
points(mean1[1], mean1[2], col = "blue", pch = 4, cex = 3, lwd = 3)
points(mean2[1], mean2[2], col = "red", pch = 4, cex = 3, lwd = 3)

```



Key observation: Random walk Metropolis often gets trapped in one mode, while geomc successfully explores both modes, resulting in estimates of the means close to their true values.

Example 2 Continued: Bayesian Inference for Normal Data with MALA Base Density

So far, we have used only random-walk base densities for both the default and custom settings. We now revisit Example 2, which involves Bayesian inference for the mean and variance of normal data, to demonstrate the use of a custom Metropolis-Adjusted Langevin Algorithm (MALA) base density that leverages gradients of the target posterior density. We reparameterize: $(\mu, \sigma^2) \rightarrow \theta \equiv (\mu, \eta)$ with $\eta = \log \sigma^2$ and use the base density

$$f(\theta'|\theta) = \phi_2(\theta'; \theta + \delta \nabla \log \psi(\theta)/2, \delta I),$$

where $\psi(\theta)$ is the target posterior density of θ .

Model Setup

```
# Generate data
set.seed(42)
true_mu <- 5
true_sigma <- 2
n <- 100
w <- rnorm(n, mean = true_mu, sd = true_sigma)

# Define log-posterior
# Log posterior for theta = (mu, eta = log(sigma^2))
log_target_normal <- function(theta, x, mu0, tau0, alpha0, beta0) {
  mu <- theta[1]
  eta <- theta[2]
  sigma2 <- exp(eta)

  n <- length(x)
  SSE <- sum((x - mu)^2)

  val <- -(n/2)*eta - SSE/(2*sigma2)
  val <- val - (mu - mu0)^2/(2*tau0^2)
  val <- val - (alpha0 + 1)*eta - beta0/sigma2

  return(val)
}

# Gradient of log posterior wrt (mu, eta)
grad_log_target_normal <- function(theta, x, mu0, tau0, alpha0, beta0) {
  mu <- theta[1]
  eta <- theta[2]
  sigma2 <- exp(eta)

  n <- length(x)
  SSE <- sum((x - mu)^2)

  # d/dmu
  grad_mu <- (sum(x) - n*mu)/sigma2 - (mu - mu0)/tau0^2

  # d/deta
  grad_eta <- -(n/2) - (alpha0 + 1)
  grad_eta <- grad_eta + SSE/(2*sigma2) + beta0/sigma2
  return(c(grad_mu, grad_eta))
}

# Hyperparameters set as before (weakly informative priors)
mu0 <- 0      # prior mean for mu
tau0 <- 10    # prior sd for mu
alpha0 <- 2.01 # shape parameter for inverse-gamma
beta0 <- 1.01 # scale parameter for inverse-gamma

#step-size for MALA proposal
step <- 0.001
mala_mean<- function(theta)
  theta+(step/2)*grad_log_target_normal(theta, w, mu0, tau0, alpha0, beta0)
```

```

# Run geomc
set.seed(123)
result <- geomc(
  logp = list(log.target=log_target_normal,
              mean.base= function(theta)
                mala_mean(theta),
              var.base=function(theta) step*diag(2)),
  initial = c(0, log(1)), # Starting at mu=0, eta= log(1)
  n.iter = 1000,
  x = w,
  mu0 = mu0,
  tau0 = tau0,
  alpha0 = alpha0,
  beta0 = beta0
)

```

Trace plots

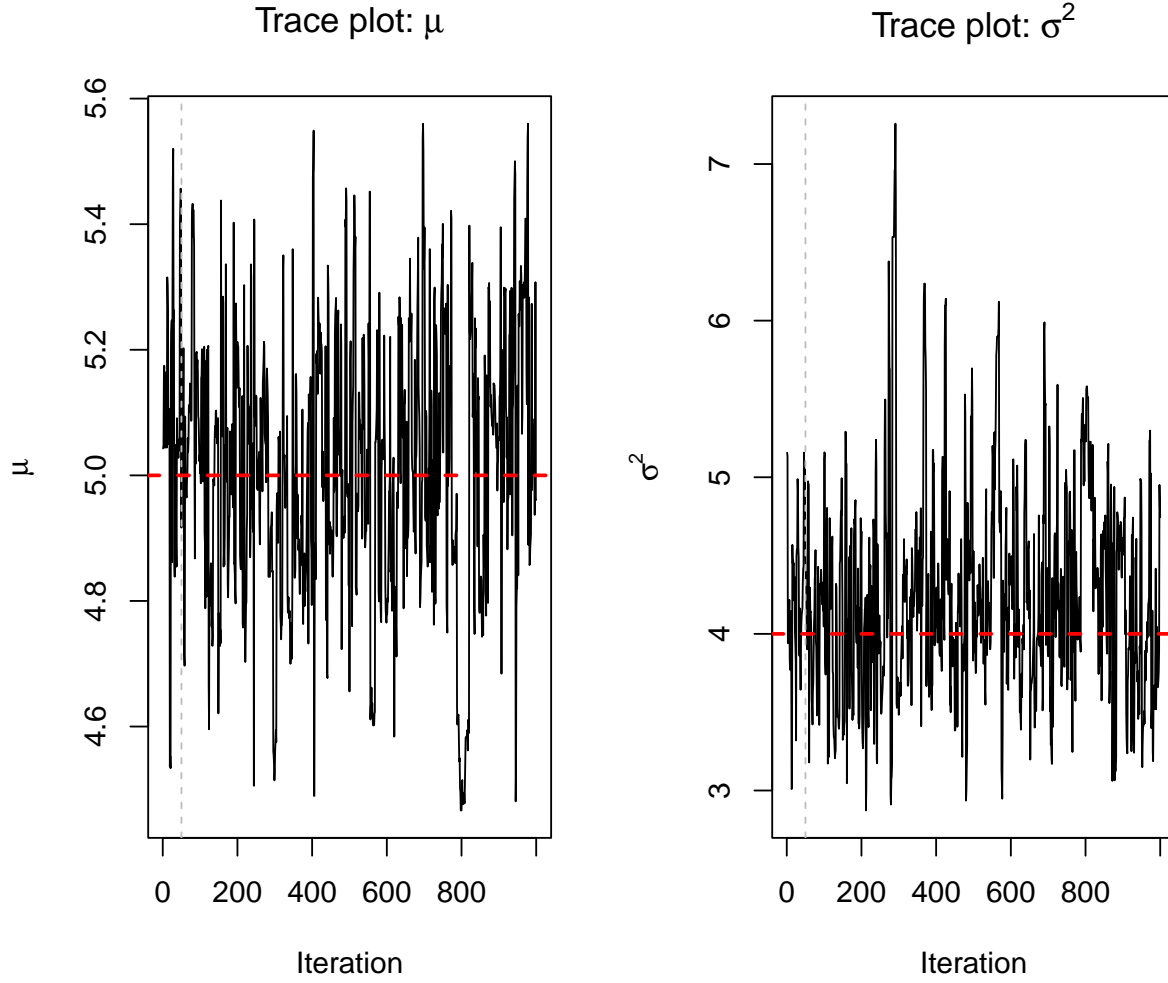
```

par(mfrow = c(1, 2))

# Trace plots
plot(result$samples[, 1], type = "l",
     main = expression(paste("Trace plot: ", mu)),
     ylab = expression(mu), xlab = "Iteration")
abline(v = burnin, col = "gray", lty = 2)
abline(h = true_mu, col = "red", lty = 2, lwd = 2)

plot(exp(result$samples[, 2]), type = "l",
     main = expression(paste("Trace plot: ", sigma^2)),
     ylab = expression(sigma^2), xlab = "Iteration")
abline(v = burnin, col = "gray", lty = 2)
abline(h = true_sigma^2, col = "red", lty = 2, lwd = 2)

```

Example 5: Discrete Distribution (Binomial)

`geomc` can also sample from discrete target distributions. We consider an example with a Binomial target. Thus, the target pmf is

$$\psi(y) = \binom{m}{y} p^y (1-p)^{m-y}, \quad y = 0, 1, 2, \dots, m.$$

Discrete Distribution (Binomial): Run `geomc` with Random Walk Base (and user defined `bhat.coef` function)

For sampling from a discrete target distribution, we must provide a base pmf f and one or more g_i 's. First, we consider a random walk base and a non-informative uniform approximate target pmf with

$$f(y|x) = \begin{cases} 0.5 & \text{if } |y-x| = 1 \\ 0.5 & \text{if } x \in \{0, m\} \text{ and } y = x \\ 0 & \text{otherwise} \end{cases} \quad \text{for } x, y \in \{0, 1, \dots, m\},$$

and

$$g(y) = \begin{cases} \frac{1}{m+1} & \text{if } y \in \{0, 1, \dots, m\} \\ 0 & \text{otherwise.} \end{cases}$$

```
# Binomial parameters
size <- 5 #m=size
true_prob <- 0.3 #p=0.3

# Define discrete random walk base density
dens.base <- function(y, x){
  if (x > 0 && x < size) {
    if (y == x - 1 || y == x + 1) {
      return(0.5)
    } else {
      return(0)
    }
  }
  if (x == 0) {
    if (y == 0 || y == 1) {
      return(0.5)
    } else {
      return(0)
    }
  }
  if (x == size) {
    if (y == size - 1 || y == size) {
      return(0.5)
    } else {
      return(0)
    }
  }
  return(0)
}

samp.base<- function(x) {
  if (x > 0 && x < size) {
    return(x + sample(c(-1, 1), 1))
  }
  if (x == 0) {
    return(sample(c(0, 1), 1))
  }
  if (x == size) {
    return(sample(c(size - 1, size), 1))
  }
}

# Define approximate target as the discrete uniform distribution
dens.ap.tar <- function(y, x) 1 / (size + 1)
samp.ap.tar <- function(x, kk = 1) sample(0:size, 1)

# Define the Bhattacharyya coefficients \langle \sqrt{f(\cdot/x)}, \sqrt{g(\cdot/x)} \rangle
bhat.coef=function(x) sqrt(2 / (size + 1))

set.seed(123)
```

```

result_binom_rw <- geomc(
  logp = list(
    log.target = function(y) dbinom(y, size, true_prob, log = TRUE),
    dens.base = dens.base,
    samp.base = samp.base,
    dens.ap.tar = dens.ap.tar,
    samp.ap.tar = samp.ap.tar,
    bhat.coef=bhat.coef
  ),
  initial = 2,
  n.iter = 1000,
  gaus = FALSE, # Not Gaussian
)

cat("Acceptance rate:", round(result_binom_rw$acceptance.rate, 3), "\n")

```

```
#> Acceptance rate: 0.699
```

Note that for sampling from discrete target distributions, user must set `gaus` as `FALSE` and either provide the `bhat.coef` function or set `imp$enabled` as `TRUE` (these are not the default values of `gaus` and `imp$enabled`). This ensures that the algorithm uses importance sampling rather than numerical integration or closed-form Gaussian expressions for computing $\langle \sqrt{f}, \sqrt{g} \rangle$ for discrete targets. In this example, the Bhattacharyya coefficient is available in closed form, and is provided to the call to `geomc` through the `bhat.coef` function.

Comparing with True Distribution

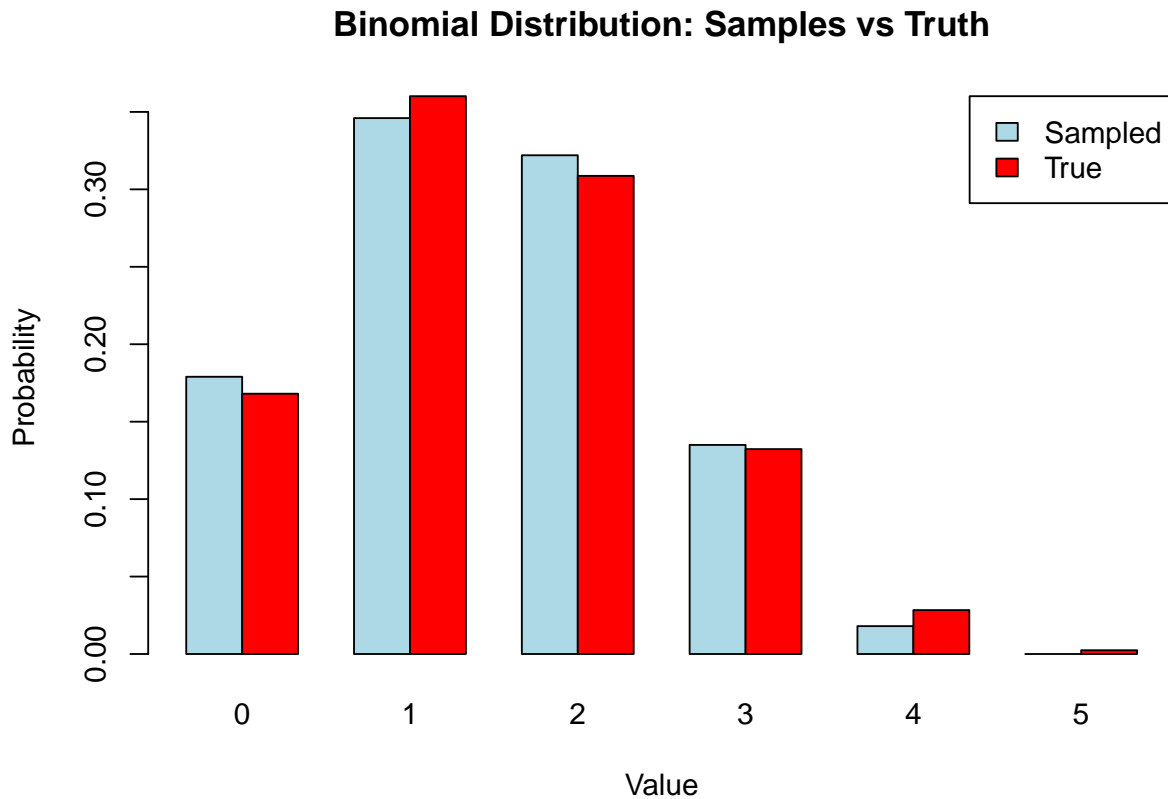
```

# Observed frequencies
obs_freq_rw <- table(factor(result_binom_rw$samples, levels = 0:size))
obs_prop_rw <- obs_freq_rw / sum(obs_freq_rw)

# True probabilities
true_prob_vals <- dbinom(0:size, size, true_prob)

# Plot comparison
barplot(rbind(obs_prop_rw, true_prob_vals),
  beside = TRUE,
  names.arg = 0:size,
  col = c("lightblue", "red"),
  main = "Binomial Distribution: Samples vs Truth",
  xlab = "Value",
  ylab = "Probability",
  legend.text = c("Sampled", "True"),
  args.legend = list(x = "topright"))

```



```
# Detailed comparison
comparison <- data.frame(
  Value = 0:size,
  Sampled = as.numeric(obs_prop_rw),
  True = true_prob_vals,
  Difference = as.numeric(obs_prop_rw) - true_prob_vals
)
print(round(comparison, 4))
```

```
#>   Value Sampled   True Difference
#> 1     0  0.179 0.1681    0.0109
#> 2     1  0.346 0.3601   -0.0141
#> 3     2  0.322 0.3087    0.0133
#> 4     3  0.135 0.1323    0.0027
#> 5     4  0.018 0.0284   -0.0104
#> 6     5  0.000 0.0024   -0.0024
```

Discrete Distribution (Binomial): Run geomc with Reflecting Random Walk Base (and user defined `bhat.coef` function)

Next, we consider a reflecting random walk base pmf with

$$f(y|x) = \begin{cases} 0.5 & \text{if } x \in \{1, 2, \dots, m-1\} \text{ and } |y-x|=1 \\ 1 & \text{if } (x=0 \text{ and } y=1) \text{ or } (x=m \text{ and } y=m-1) \\ 0 & \text{otherwise} \end{cases}$$

As before, we use the non-informative uniform approximate target pmf. As in the previous case, the Bhattacharyya coefficient is available in closed form, and is provided to the call to `geomc` through the `bhat.coef` function.

```
# Binomial parameters
size <- 5
true_prob <- 0.3

# Define discrete reflecting random walk base density
dens.base <- function(y, x){
  if (x > 0 && x < size) {
    if (y == x - 1 || y == x + 1) {
      return(0.5)
    } else {
      return(0)
    }
  }
  if (x == 0) {
    if (y == 1) {
      return(1)
    } else {
      return(0)
    }
  }
  if (x == size) {
    if (y == size - 1) {
      return(1)
    } else {
      return(0)
    }
  }
  return(0)
}

samp.base<- function(x) {
  if (x == 0) {
    return(1)
  }
  if (x == size) {
    return(size - 1)
  }
  return(x + sample(c(-1, 1), 1))
}

# Define approximate target as the discrete uniform distribution
dens.ap.tar <- function(y, x) 1 / (size + 1)
samp.ap.tar <- function(x, kk = 1) sample(0:size, 1)

# Define the Bhattacharyya coefficients  $\langle \sqrt{f(\cdot/x)}, \sqrt{g(\cdot/x)} \rangle$ 
bhat.coef<- function(x) {
  ifelse(x == 0 || x == size, 1, sqrt(2)) / sqrt(size + 1)
}
```

```

set.seed(123)
result_binom_rrw <- geomc(
  logp = list(
    log.target = function(y) dbinom(y, size, true_prob, log = TRUE),
    dens.base = dens.base,
    samp.base = samp.base,
    dens.ap.tar = dens.ap.tar,
    samp.ap.tar = samp.ap.tar,
    bhat.coef = bhat.coef
  ),
  initial = 2,
  n.iter = 1000,
  gaus = FALSE, # Not Gaussian
)

cat("Acceptance rate:", round(result_binom_rrw$acceptance.rate, 3), "\n")

```

```
#> Acceptance rate: 0.741
```

Comparing with True Distribution

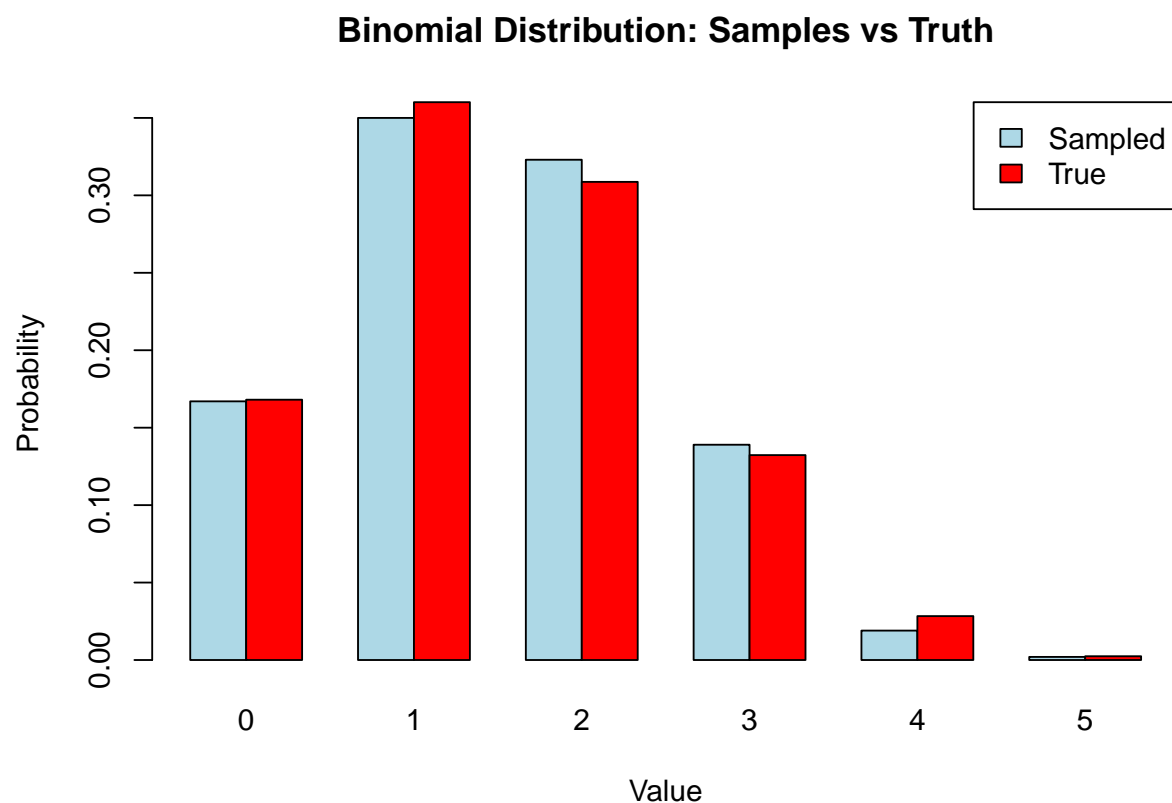
```

# Observed frequencies
obs_freq_rrw <- table(factor(result_binom_rrw$samples, levels = 0:size))
obs_prop_rrw <- obs_freq_rrw / sum(obs_freq_rrw)

# True probabilities
true_prob_vals <- dbinom(0:size, size, true_prob)

# Plot comparison
barplot(rbind(obs_prop_rrw, true_prob_vals),
  beside = TRUE,
  names.arg = 0:size,
  col = c("lightblue", "red"),
  main = "Binomial Distribution: Samples vs Truth",
  xlab = "Value",
  ylab = "Probability",
  legend.text = c("Sampled", "True"),
  args.legend = list(x = "topright"))

```



```
# Detailed comparison
comparison <- data.frame(
  Value = 0:size,
  Sampled = as.numeric(obs_prop_rrw),
  True = true_prob_vals,
  Difference = as.numeric(obs_prop_rrw) - true_prob_vals
)
print(round(comparison, 4))
```

```
#>   Value Sampled   True Difference
#> 1     0   0.167 0.1681   -0.0011
#> 2     1   0.350 0.3601   -0.0101
#> 3     2   0.323 0.3087    0.0143
#> 4     3   0.139 0.1323    0.0067
#> 5     4   0.019 0.0284   -0.0094
#> 6     5   0.002 0.0024   -0.0004
```

We observe that the estimated probabilities obtained by `geomc` with the proposal of either of the two random walks as its base pmf f and the discrete uniform pmf as g are close to the target probabilities. `geomc` with these choices of f and g can also be used for sampling from other discrete target distributions.

Discrete Distribution (Binomial): Run geomc with Uniform Base (an example with ind set as TRUE)

Finally, we consider a uniform base pmf, that is

$$f(y|x) = \begin{cases} \frac{1}{m+1} & \text{if } y \in \{0, 1, \dots, m\} \\ 0 & \text{otherwise,} \end{cases}$$

and the approximate target pmf g is another Binomial pmf. In particular,

$$g(y) = \binom{m}{y} p_1^y (1 - p_1)^{m-y}, \quad y = 0, 1, 2, \dots, m,$$

for some $p_1 \neq p$.

```
# Binomial parameters
size <- 5
true_prob <- 0.3

# Define discrete uniform base density
dens.base <- function(y, x) 1 / (size + 1)
samp.base <- function(x) sample(0:size, 1)

# Define approximate target (another binomial with different probability)
dens.ap.tar <- function(y, x) dbinom(y, size, 0.7) #p_1=0.7
samp.ap.tar <- function(x, kk = 1) rbinom(kk, size, 0.7)
set.seed(123)
result_binom <- geomc(
  logp = list(
    log.target = function(y) dbinom(y, size, true_prob, log = TRUE),
    dens.base = dens.base,
    samp.base = samp.base,
    dens.ap.tar = dens.ap.tar,
    samp.ap.tar = samp.ap.tar
  ),
  initial = 2,
  n.iter = 1000,
  ind = TRUE, # Base and approximate target don't depend on current state
  gaus = FALSE, # Not Gaussian
  imp = list(
    enabled = TRUE,
    n.samp = 1000,
    samp.base = TRUE #samples from the base density is used in the importance sampling
  )
)

cat("Acceptance rate:", round(result_binom$acceptance.rate, 3), "\n")
```

```
#> Acceptance rate: 0.582
```

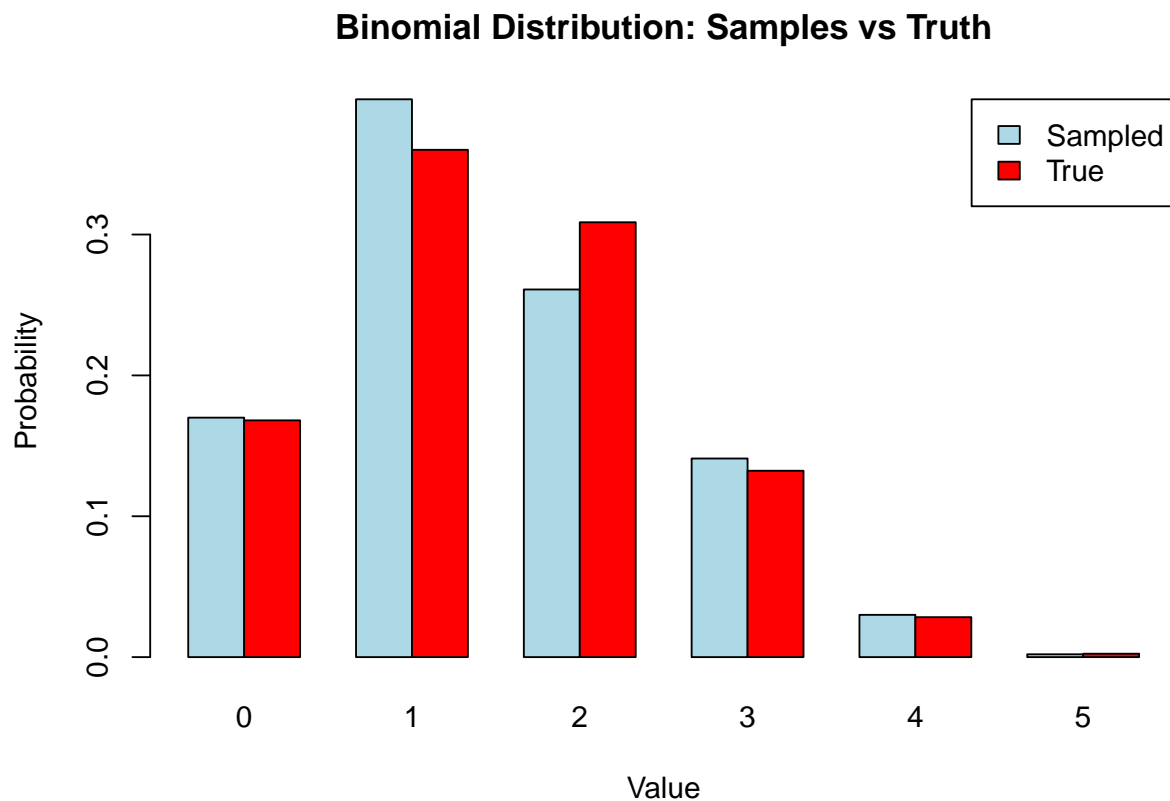
Note that since the base and approximate target don't depend on current state, `ind` is set as `TRUE` implying $\langle f, g \rangle$ is computed only once instead of in every MCMC iteration. Also since the `bhat.coef` function is not provided (although the Bhattacharyya coefficient is available in closed form), we set `imp$enabled` as `TRUE` (not the default value). This ensures that the algorithm uses importance sampling rather than numerical integration or closed-form Gaussian expressions for computing $\langle \sqrt{f}, \sqrt{g} \rangle$ for discrete targets.

Comparing with True Distribution

```
# Observed frequencies
obs_freq<- table(factor(result_binom$samples, levels = 0:size))
obs_prop <- obs_freq / sum(obs_freq)

# True probabilities
true_prob_vals <- dbinom(0:size, size, true_prob)

# Plot comparison
barplot(rbind(obs_prop, true_prob_vals),
        beside = TRUE,
        names.arg = 0:size,
        col = c("lightblue", "red"),
        main = "Binomial Distribution: Samples vs Truth",
        xlab = "Value",
        ylab = "Probability",
        legend.text = c("Sampled", "True"),
        args.legend = list(x = "topright"))
```



```
# Detailed comparison
comparison <- data.frame(
  Value = 0:size,
  Sampled = as.numeric(obs_prop),
  True = true_prob_vals,
  Difference = as.numeric(obs_prop) - true_prob_vals)
```

```
)
print(round(comparison, 4))
```

```
#>   Value Sampled   True Difference
#> 1     0    0.170 0.1681      0.0019
#> 2     1    0.396 0.3601      0.0359
#> 3     2    0.261 0.3087     -0.0477
#> 4     3    0.141 0.1323      0.0087
#> 5     4    0.030 0.0284      0.0016
#> 6     5    0.002 0.0024     -0.0004
```

Summary of Non-Default Settings

When to Use Custom Specifications

1. **Custom base density:** When you have an idea of appropriate scale for random walk, or want to use a heavy-tailed or a non-random walk (e.g. MALA) proposal
2. **Custom approximate target:** When you know the structure of your target (e.g., mixture components, multimodality)
3. **Bhattacharyya coefficient:** When `gaus = FALSE`, the Bhattacharyya coefficient should be provided via the function `bhat.coef`, if it is available in closed form or if computing it without numerical integration or importance sampling is preferred.
4. **Importance sampling:** When `gaus = FALSE`, `bhat.coef` is not provided, and dimension is not small so that numerical integration is expensive or target is a discrete distribution
5. **Discrete distributions:** Set `gaus = FALSE` and provide appropriate base and `ap.tar` discrete densities.
6. **Base and approximate targets don't depend on the current state:** If both base and approximate targets don't depend on the current state, `ind` should be set `TRUE` to avoid unnecessary inner product computation in every Markov chain iteration.

References

- Roy, Vivekananda. 2020. "Convergence Diagnostics for Markov Chain Monte Carlo." *Annual Review of Statistics and Its Application* 7: 387–412.
- . 2024. "A Geometric Approach to Informed MCMC Sampling." *arXiv Preprint arXiv:2406.09010*.