Applying the common random number technique as a Markov chain convergence diagnostic

Sabrina Sixta

2023-06-07

The following code generated the graphs and calculations found in "Applying the common random number technique as a Markov chain convergence diagnostic."

Bayesian Regression Gibbs Sampler

Numerical example

```
# Data is obtained from the general linear models textbook by Dobson
#Table 6.3 Carbohydrate, age, relative weight and protein for twenty male insulin dependent diabetics;

df <- c(33, 33, 100, 14,
40, 47, 92, 15,
37, 49, 135, 18,
27, 35, 144, 12,
30, 46, 140, 15,
43, 52, 101, 15,
34, 62, 95, 14,
48, 23, 101, 17,
30, 32, 98, 15,
38, 42, 105, 14,
50, 31, 108, 17,
51, 61, 85, 19,
30, 63, 130, 19,
36, 40, 127, 20,
41, 50, 109, 15,
42, 64, 107, 16,
46, 56, 117, 18,
24, 61, 100, 13,
35, 48, 118, 18,
37, 28, 102, 14)

df <- matrix(df, nrow = 20, byrow=TRUE)

Y <- df[,1]
X <- cbind(1,df[,-1])
n <- length(Y)
p <- dim(X)[2]

df <- data.frame(df)
colnames(df) <- c("carbs", "age", "weight", "protein")
```

```r
# setting priors: beta ~ N(b_0,E_0) sigma^2 ~ Inv-Chi^2(v_0, c_0)
b_0 <- rep(0,4)
E_0 <- diag(4)
v_0 <- 1
c_0 <- 6
c_0 <- 10
```

Consistent with equation (18) we define

$$g(\beta, \sigma^2) = \frac{1}{(\sigma^2)^{(n+\upsilon_0)/2+1}} \exp\left(-\frac{1}{2\sigma^2}(y - X\beta)^T(y - X\beta) - \frac{1}{2}(\beta - \beta_0)^T \Sigma_\beta^{-1}(\beta - \beta_0) - \frac{\upsilon_0 c_0^2}{\sigma^2}\right)$$

We further define

$$f(\beta, \sigma^2) = \log(g(\beta, \sigma^2))$$

```r
# g(x,y)= g(beta,sigma^2) as defined in equation 6
g <- function(x){
  b <- c(x[1], x[2], x[3], x[4])
  o <- x[5]
  z = exp(-((n+v_0)/2+1)*log(o) - 1/(2*o)*t(Y-(X %*% b)) %*% (Y-(X %*% b)) - 0.5*t(b-b_0) %*% inv(E_0) %
  return(z)}

f <- function(x){
  b <- c(x[1], x[2], x[3], x[4])
  o <- x[5]
  z = -((n+v_0)/2+1)*log(o) - 1/(2*o)*t(Y-(X %*% b)) %*% (Y-(X %*% b)) - 0.5*t(b-b_0) %*% inv(E_0) %*%
  return(-z)
  }
```

We want to find a lower bound (L) on $\int_{\mathbb{R}^4\times\mathbb{R}} g(\beta, \sigma^2)d(\beta, \sigma^2)$. To do so, we apply the following,

$\int_{\mathbb{R}^4\times\mathbb{R}} g(\beta, \sigma^2)d(\beta, \sigma^2) \geq \int_C g(\beta, \sigma^2)d(\beta, \sigma^2) \geq e^{\int_C f(\beta,\sigma^2)d(\beta,\sigma^2)}$

```r
x <- c(0.1,0.1,0.1,0.1,0.1)
x <- optim(x, f)$par
#c(0.1,0.1,0.1,0.1,0.1)
#c(0.2,0.2,0.2,0.2,0.2)
int_f <- adaptIntegrate(f, lowerLimit = x-c(0.1,0.1,0.1,0.1,0.1), upperLimit = x+c(0.1,0.1,0.1,0.1,0.1)
int_f
```

```
## $integral
## [1] 0.03176281
##
## $error
## [1] 1.376038e-12
##
## $functionEvaluations
## [1] 93
##
## $returnCode
## [1] 0
```

```r
L <- exp(-int_f$integral)
L
```

```
## [1] 0.9687363
```

```
alpha <- (n+v_0)/2
beta <- v_0*c_0/2
K <- 1/L*gamma(alpha)/beta^alpha*(2*pi)^(p/2)
K
```

```
## [1] 2.114977
```

```
tvbound <- (n+v_0)^2/(2*v_0*c_0)
tvbound
```

```
## [1] 22.05
```

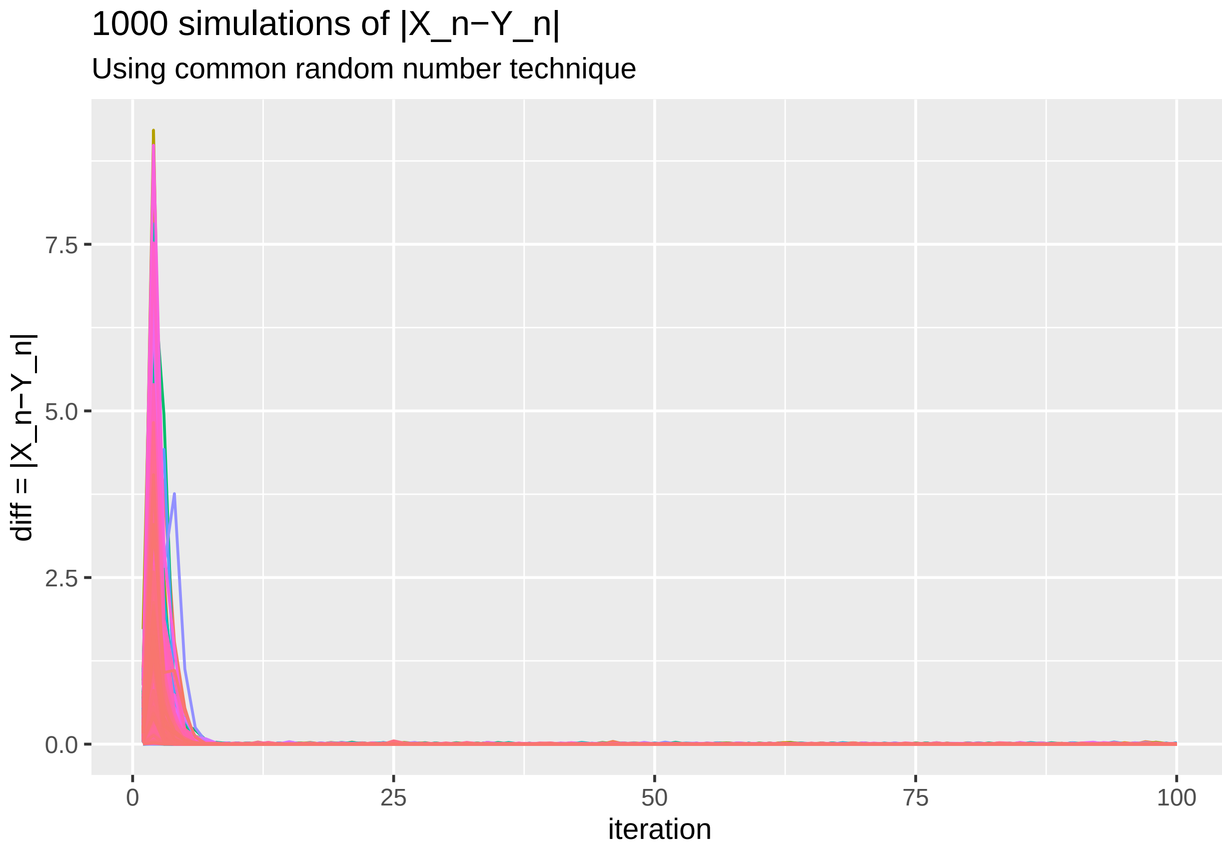
The following is function that generates σ^2_{n+1} given σ^2_n, equation 19.

```
b_hat <- inv(t(X) %*% X) %*% t(X) %*%  Y
invE_0 <- inv(E_0)
nextIt <- function(o, Z, G){
  eigenV <- eigen(t(X) %*% X/o + inv(E_0))
  Q <- eigenV$vectors
  L <- diag(eigenV$values)
  Vinv12 <-  Q %*% inv(sqrt(L)) %*% inv(Q)
  Vinv <-  Q %*% inv(L) %*% inv(Q)
  b_tilde <- Vinv %*% (t(X) %*% X %*% b_hat/o + inv(E_0) %*% b_0)
  W <- X %*% b_tilde - Y + X %*% Vinv12 %*% Z
  o1 <- (v_0*c_0/2 +(t(W) %*% W)/2)/G
  return(o1)
}
```

Now we apply the common random number technique to generate an estimate of $E[X_k - Y_k]$, $N = 100$ $I = 1000$ and $X_0, Y_0 \sim \Gamma^{-1}(\alpha', \beta') = \Gamma^{-1}(10.5, 2)$

```
I = 1000
#I = 100
J = 100
diff <- matrix(0, I, J)
for(i in 1:I){
  it <- matrix(0, ncol=2, nrow=J)

  it[1,] <- 1/rgamma(2, shape = alpha, rate = beta)
  for(j in 2:J){
    Z <- rnorm(p, 0, 1)
    G <- rgamma(1, shape = alpha, rate =1)
    it[j,1] <- nextIt(it[j-1,1], Z, G)
    it[j,2] <- nextIt(it[j-1,2], Z, G)
  }

  diff[i,] <- abs(it[,1]-it[,2])
}
```

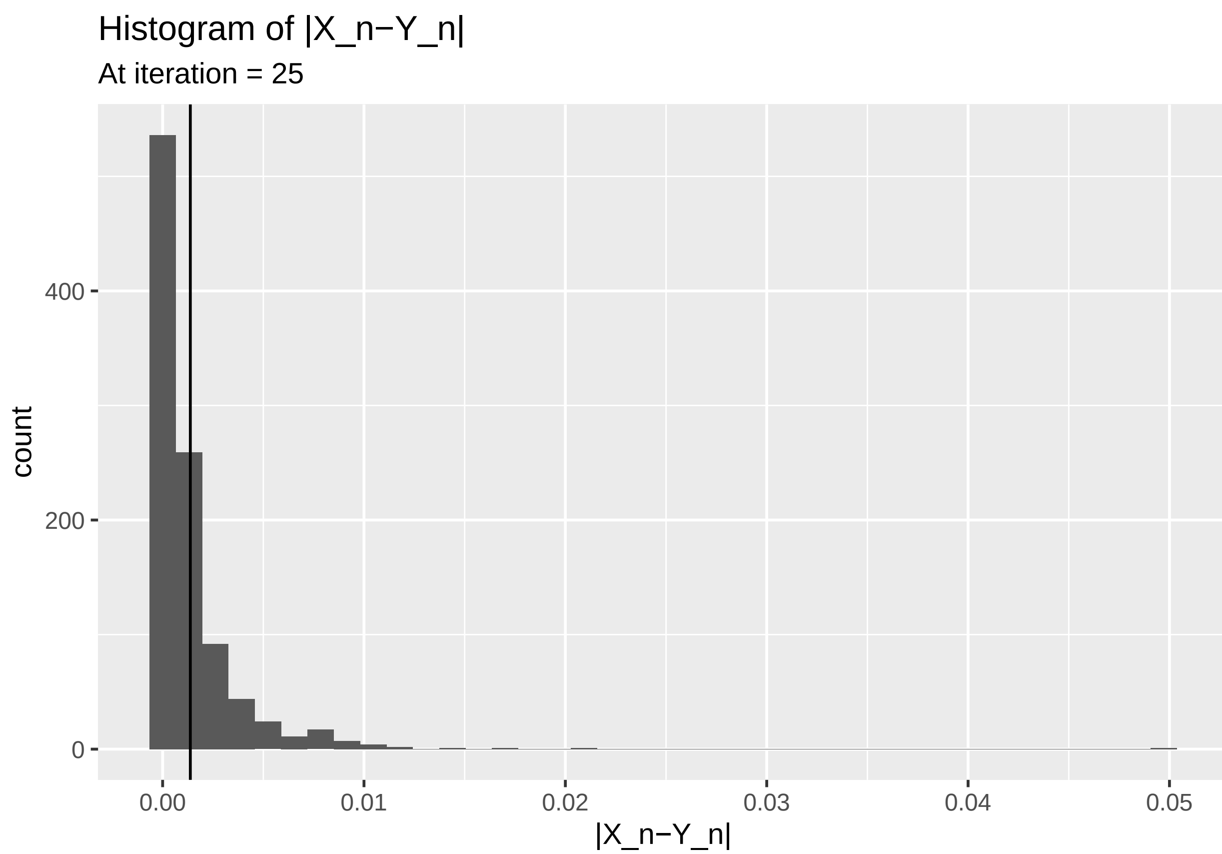
```
diff_df <- data.frame(t(diff), iter_no = 1:J)
diff_df <- diff_df %>%
  pivot_longer(cols = starts_with("X"), names_to = "sim_no", values_to = "val")
diff_df %>%
  ggplot(aes(x = iter_no, y = val)) +
  geom_line(aes(color = sim_no)) + theme(legend.position = "none") +
  labs(title = "1000 simulations of |X_n-Y_n|", subtitle = "Using common random number technique") +
  xlab("iteration") + ylab("diff = |X_n-Y_n|")
```




```
expdiff <- diff_df %>%
  filter(iter_no==25) %>%
  summarise(expdiff = mean(val))

expdiff
```

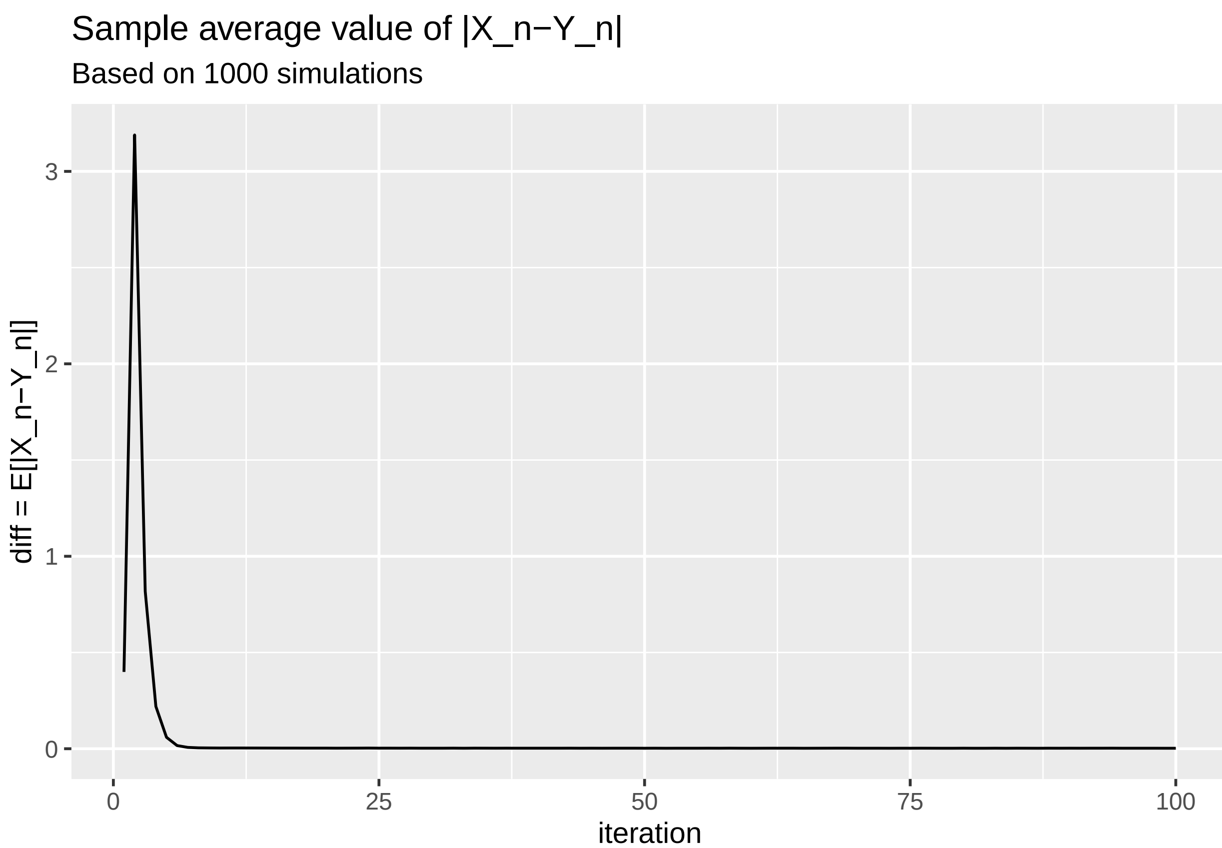
```
## # A tibble: 1 x 1
##   expdiff
##     <dbl>
## 1 0.00138
```

```
diff_df %>%
  filter(iter_no==25) %>%
  ggplot(aes(x=val)) +
  geom_histogram(bins = "39") + labs(title = "Histogram of |X_n-Y_n|", subtitle = "At iteration = 25") +
  xlab("|X_n-Y_n|") +
  geom_vline(aes(xintercept = expdiff$expdiff), colour="black")
```




```
  #geom_vline(xintercept = expdiff)
```

```
diff_df %>%
  group_by(iter_no) %>%
  summarise(mean_val = mean(val)*K) %>%
  ggplot(aes(x = iter_no, y = mean_val)) +
  geom_line() + theme(legend.position = "none") +
  labs(title = "Sample average value of |X_n-Y_n|", subtitle = "Based on 1000 simulations") +
  xlab("iteration") + ylab("diff = E[|X_n-Y_n|]")
```




```
expdiff*K
```

```
##       expdiff
## 1 0.002909563
```

```
expdiff*K*tvbound
```

```
##      expdiff
## 1 0.06415586
```

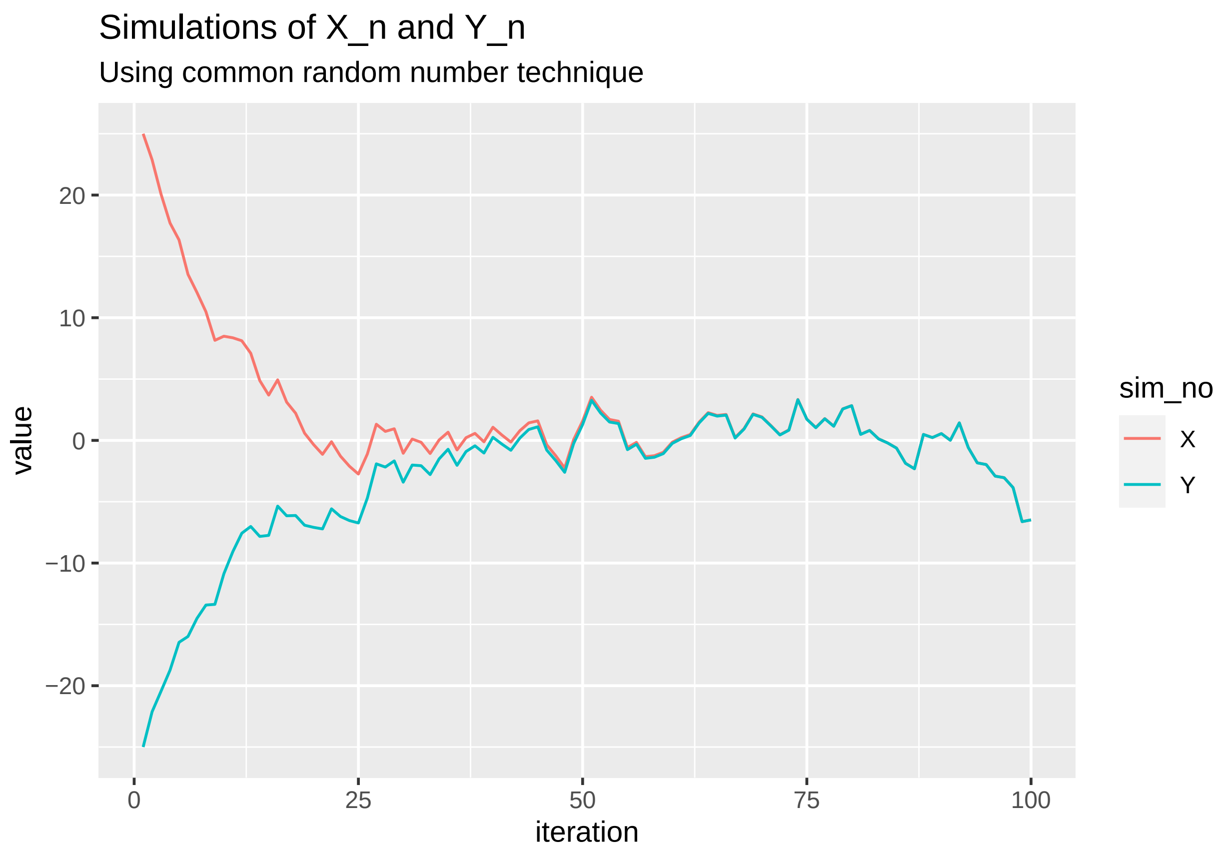
Autoregressive process

Define the autorregressive process to be

$$X_n = 0.9X_{n-1} + Z_n, Z_n \sim N(0,1)$$

```
X <- 1:100
Y <- X
Z <- rnorm(100, 0,1)

X[1] <- 25
Y[1] <- -25
for(i in 2:100){
  X[i] <- 0.9*X[i-1]+Z[i]
  Y[i] <- 0.9*Y[i-1]+Z[i]
}
x_forwards <- X
```

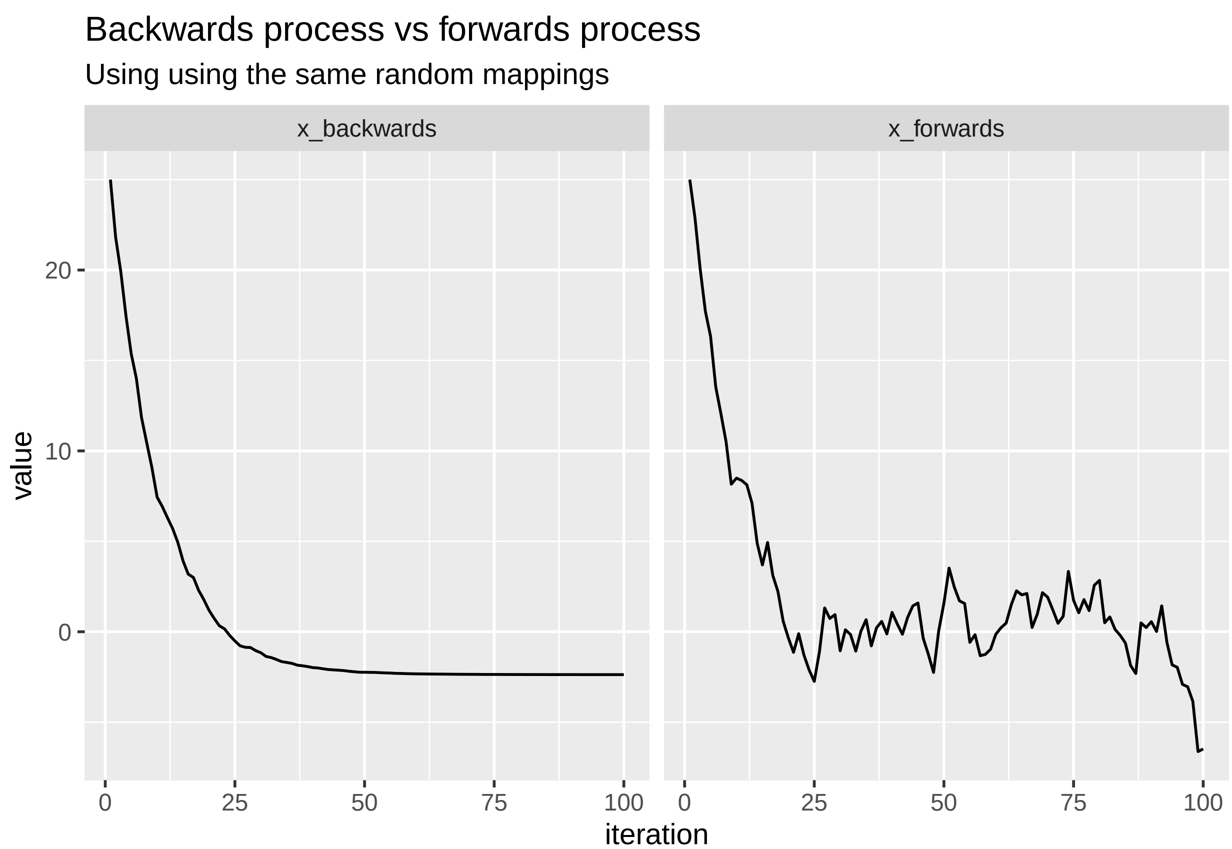
```r
df <- data.frame(X,Y, iter=1:100)
df %>%
  pivot_longer(X:Y, names_to = "sim_no", values_to = "val") %>%
  ggplot(aes(x = iter, y = val)) +
  geom_line(aes(color = sim_no)) + labs(title = "Simulations of X_n and Y_n", subtitle = "Using common :
  xlab("iteration") + ylab("value")
```




```r
n <- 100
X <- 1:n
Y <- 1:n
#Z <- rnorm(n, 0,1)

X[1] <- 25
Y[1] <- 25
for(i in 2:n){
  for(j in 2:i){
    Y[j] <- 0.9*Y[j-1]+Z[i+1-j]
  }
  X[i] <- Y[i]
}
x_backwards <- X
```

```r
df <- data.frame(x_forwards,x_backwards, iter=1:100)
df %>%
  pivot_longer(x_forwards:x_backwards, names_to = "sim_no", values_to = "val") %>%
  ggplot(aes(x = iter, y = val)) +
```

```
  geom_line() + facet_wrap(vars(sim_no)) + labs(title = "Backwards process vs forwards process", subtitl
  xlab("iteration") + ylab("value")
```

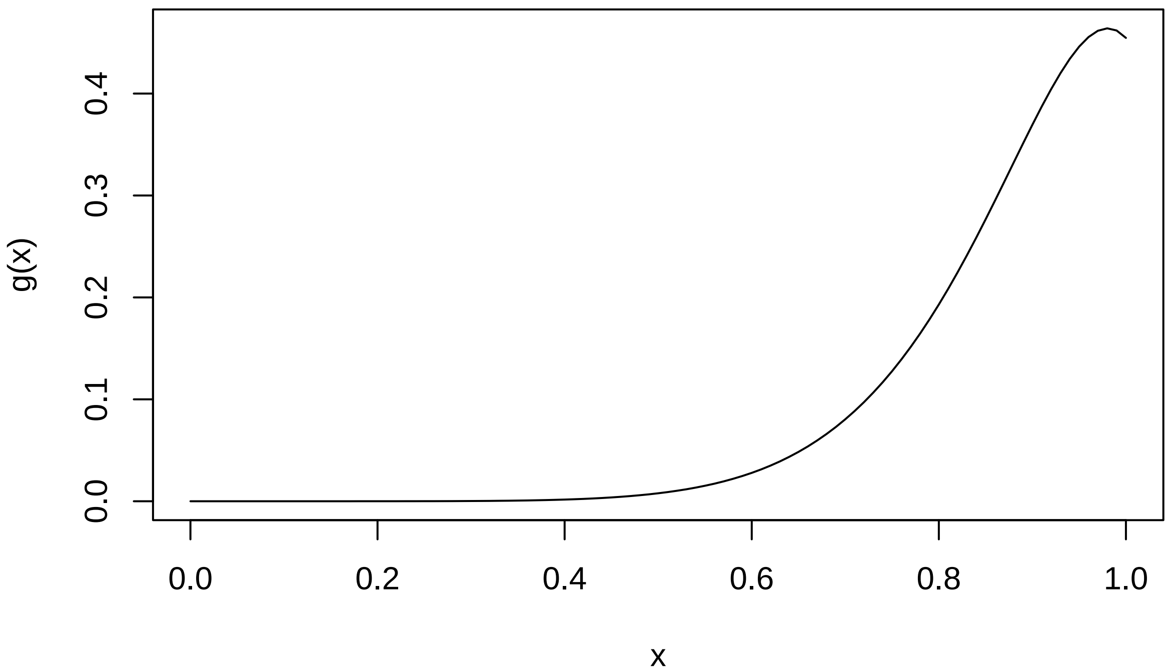


Metropolis Algorithm

The unnormalised target density is defined as follows,

$$g(x) = x^3 \sin(x^4) \cos(x^5) I_{x \in [0,1]}$$

```
g <- function(y){
  if ((0<=y) && (y<=1)){
    h <- y^3*sin(y^4)*cos(y^5)
  }
  else {
    h <- 0
  }
 return(h)
}
g_norestrict <- function(y){
  h <- y^3*sin(y^4)*cos(y^5)
  return(h)
}
curve(g_norestrict, from=0, to=1, xlab="x", ylab="g(x)")
```

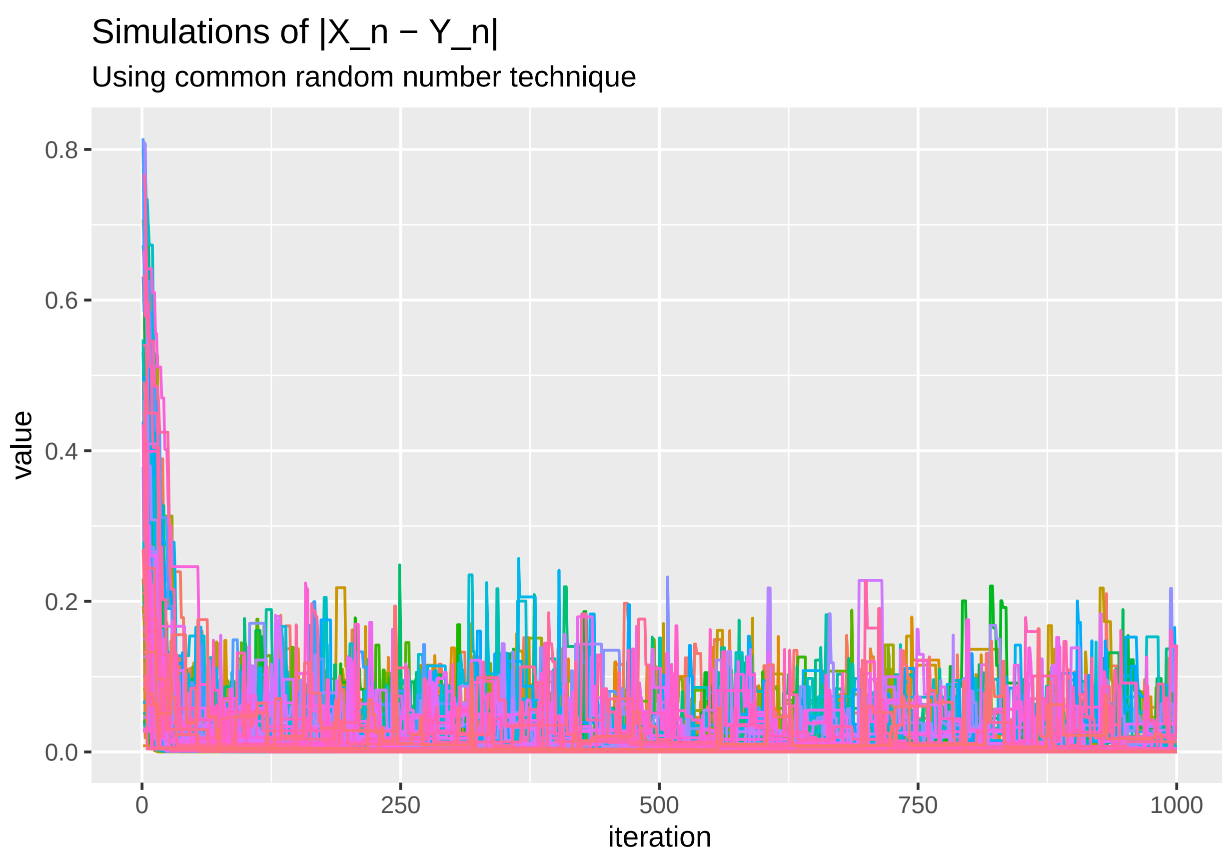



```
n <- 10000
x <- rep(0, n)
y <- x
z <- x
x[1] = 0.7
y[1] = 0.2

acceptreject <- function(u, a, x, x_prop){
  if(u<a){
    x_next <- x_prop
  }
  else {
    x_next <- x
  }
}

getsim_crn <- function(n, i){
  x <- 1:n
  y <- 1:n
  x[1] <- runif(1,0,1)
  y[1] <- runif(1,0,1)
  for (j in 2:n){
    z <- rnorm(1)
    u <- runif(1)
    x_prop = x[j-1] + 0.1*z
    y_prop = y[j-1] + 0.1*z
```

```
    A_x = g(x_prop)/g(x[j-1])
    A_y = g(y_prop)/g(y[j-1])

    x[j] = acceptreject(u, A_x, x[j-1], x_prop)
    y[j] = acceptreject(u, A_y, y[j-1], y_prop)
  }
  df <- data.frame(sim_no = as.character(i), iter = 1:n, x, y, val = abs(x-y))
  return(df)
}
```

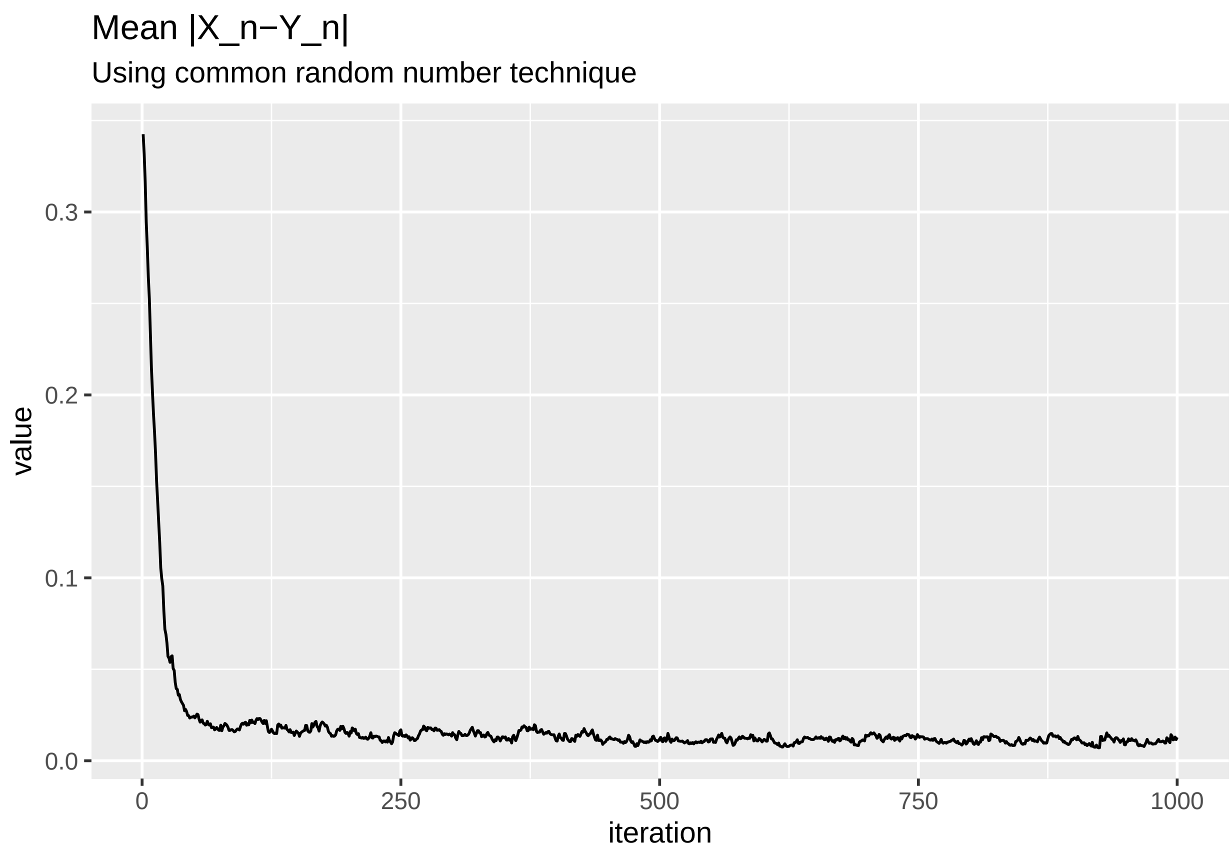
```
I <- 100
N <- 1000

df <- getsim_crn(N, 1)
for(i in 2:I){
  df <- rbind(df, getsim_crn(N, i))
}
```

```
df %>%
  ggplot(aes(x = iter, y = val)) +
  geom_line(aes(color = sim_no)) + labs(title = "Simulations of |X_n - Y_n|", subtitle = "Using common r
  xlab("iteration") + ylab("value") + theme(legend.position = "none")
```

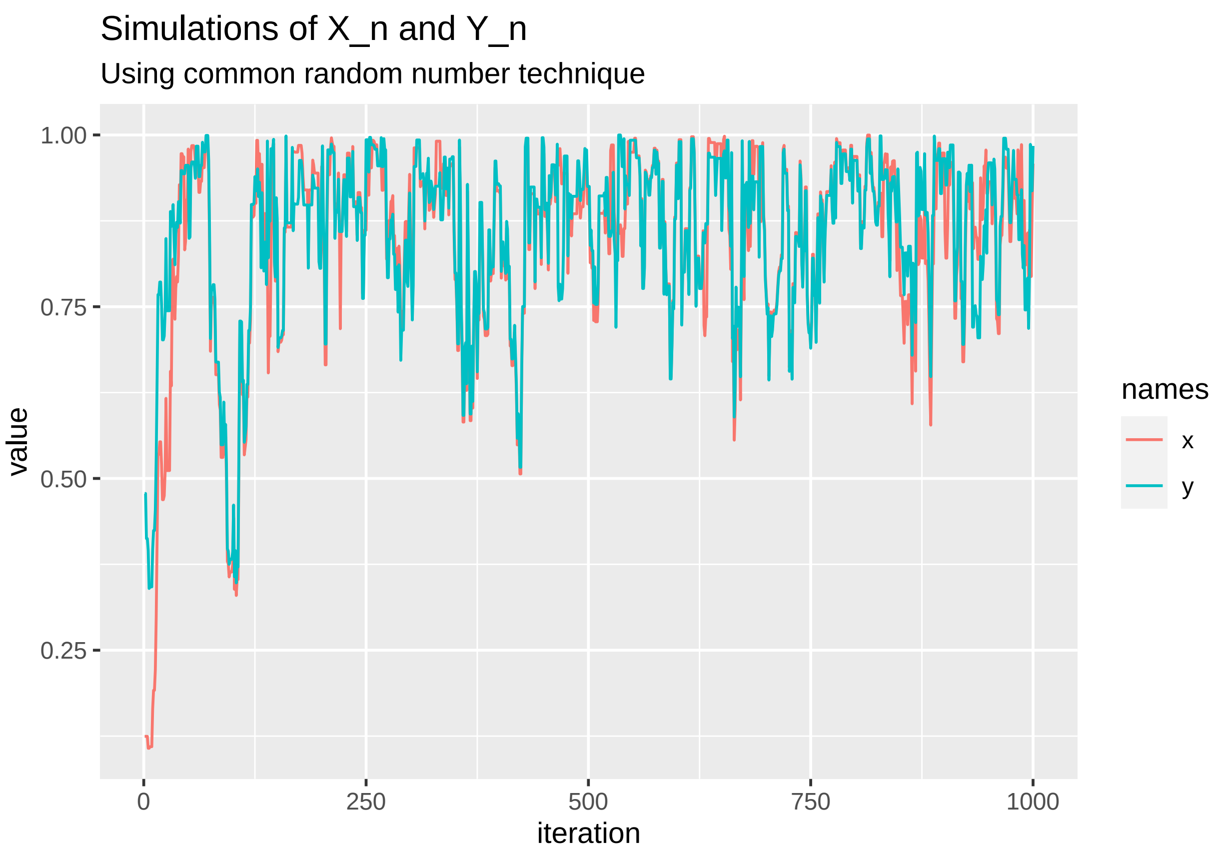



```
df %>%
  group_by(iter) %>%
  summarise(mval = mean(val))%>%
```

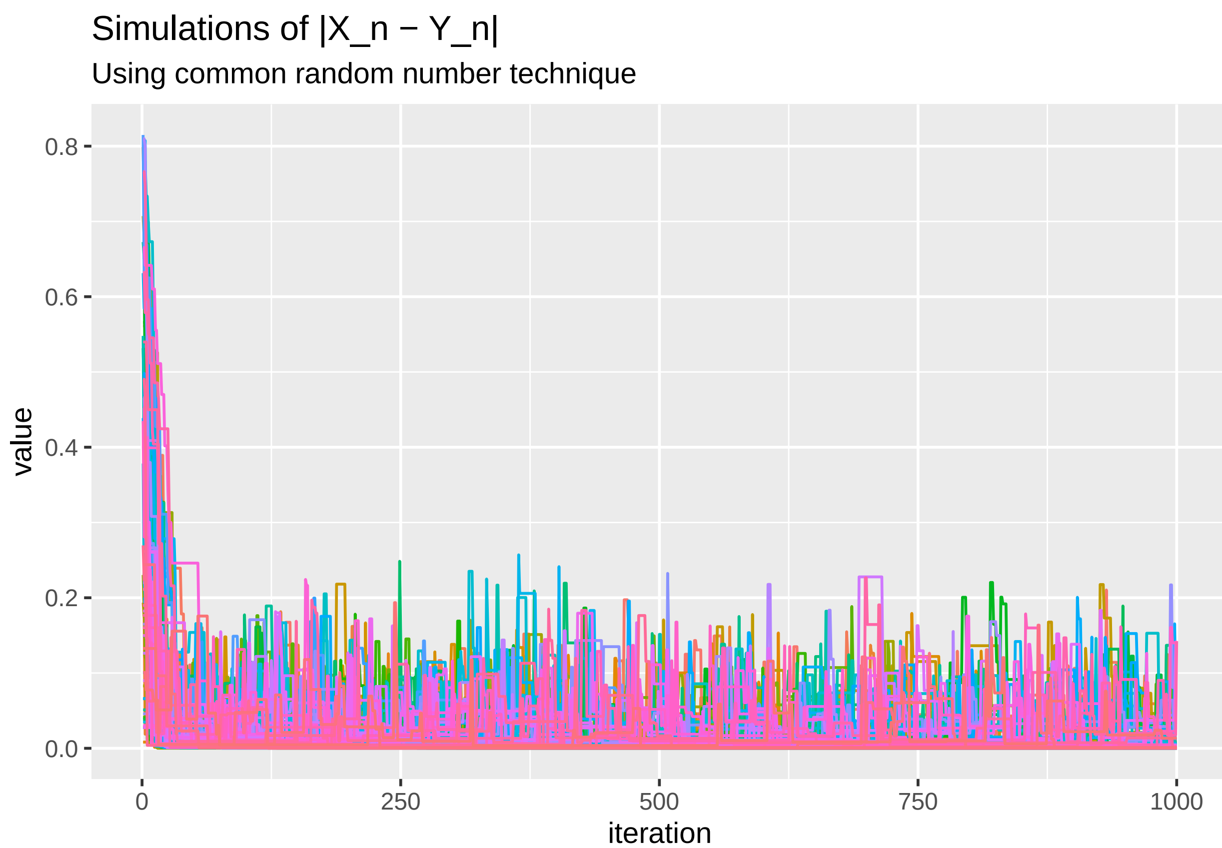
```
ggplot(aes(x = iter, y = mval)) +
geom_line() + labs(title = "Mean |X_n-Y_n|", subtitle = "Using common random number technique") +
xlab("iteration") + ylab("value")
```




```
df %>%
  filter(sim_no == 1) %>%
  pivot_longer(x:y, names_to = "names", values_to = "value") %>%
  ggplot(aes(x = iter, y = value)) +
  geom_line(aes(color = names)) + labs(title = "Simulations of X_n and Y_n", subtitle = "Using common ra
  xlab("iteration") + ylab("value")
```




```
df %>%
  ggplot(aes(x = iter, y = val)) +
  geom_line(aes(color = sim_no)) + labs(title = "Simulations of |X_n - Y_n|", subtitle = "Using common r
  xlab("iteration") + ylab("value") + theme(legend.position = "none")
```

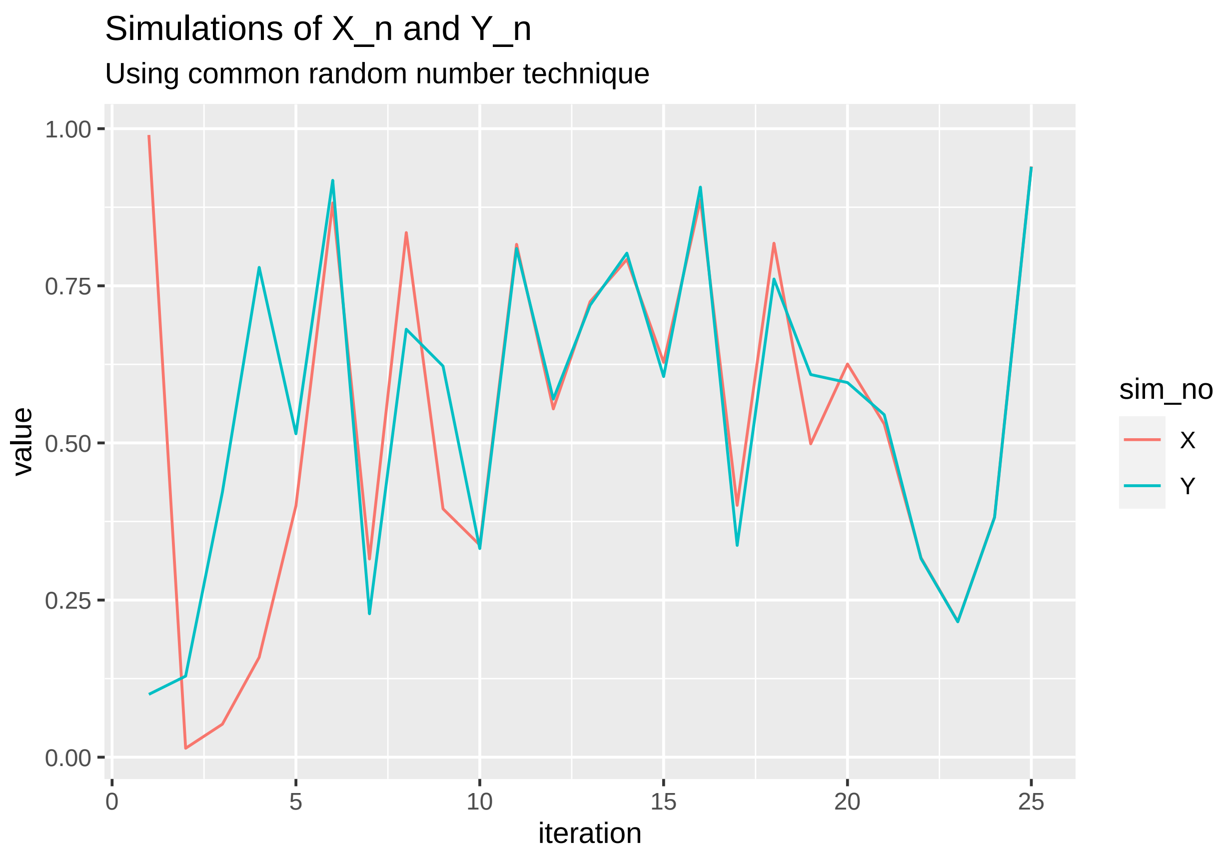


Random logistic Map

Define the logistic regression map to be

$$X_n = 4\theta_n X_{n-1}(1 - X_{n-1}), \theta_n \sim Beta(1.5, 0.5)$$

```
n <- 25
X <- 1:n
Y <- X
B <- rbeta(n, 1.5, 0.5)

X[1] <- 0.99
Y[1] <- 0.1
for(i in 2:n){
  X[i] <- 4*B[i]*X[i-1]*(1-X[i-1])
  Y[i] <- 4*B[i]*Y[i-1]*(1-Y[i-1])
}

df <- data.frame(X,Y, iter=1:n)
df %>%
  pivot_longer(X:Y, names_to = "sim_no", values_to = "val") %>%
  ggplot(aes(x = iter, y = val)) +
  geom_line(aes(color = sim_no)) + labs(title = "Simulations of X_n and Y_n", subtitle = "Using common
  xlab("iteration") + ylab("value")
```



Dirichlet process

Define the Dirichlet process is

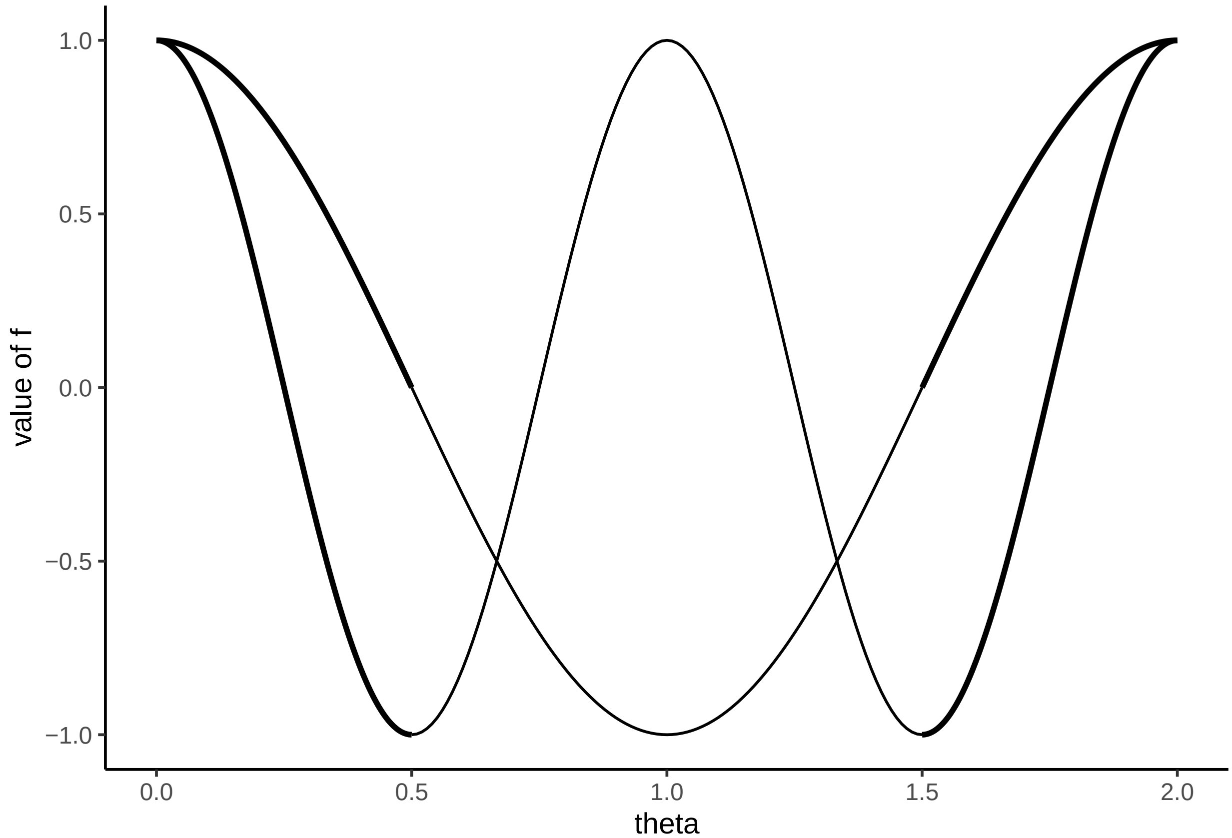
$$X_n = (1-\theta_n)Z_n + \theta_n X_{n-1}, \theta_n \sim Beta(1.5, 1), Z_n \sim N(0, 1)$$

```
n <- 50
X <- 1:n
Y <- X
B <- rbeta(n, 1.5, 1)
Z <- rnorm(n, 0, 1)

X[1] <- 10
Y[1] <- -10
for(i in 2:n){
  X[i] <- (1-B[i])*Z[i] + B[i]*X[i-1]
  Y[i] <- (1-B[i])*Z[i] + B[i]*Y[i-1]
}
```

```
df <- data.frame(X,Y, iter=1:n)
df %>%
  pivot_longer(X:Y, names_to = "sim_no", values_to = "val") %>%
  ggplot(aes(x = iter, y = val)) +
  geom_line(aes(color = sim_no)) + labs(title = "Simulations of X_n and Y_n", subtitle = "Using common 
  xlab("iteration") + ylab("value")
```

Example of the set A

```
eq = function(x){cos(2*x*pi)}
eq2 = function(x){cos(x*pi)}
ggplot(data.frame(x=c(0, 2)), aes(x=x)) +
  stat_function(fun=eq, xlim=c(0,0.5), linewidth=1) +
  stat_function(fun=eq2, xlim=c(0,0.5), linewidth=1) +
  stat_function(fun=eq, xlim=c(0.5,1.5), linewidth=0.5) +
  stat_function(fun=eq2, xlim=c(0.5,1.5), linewidth=0.5) +
  stat_function(fun=eq, xlim=c(1.5,2), linewidth=1) +
  stat_function(fun=eq2, xlim=c(1.5,2), linewidth=1) + theme_classic() +
  labs(x="theta", y="value of f")
```

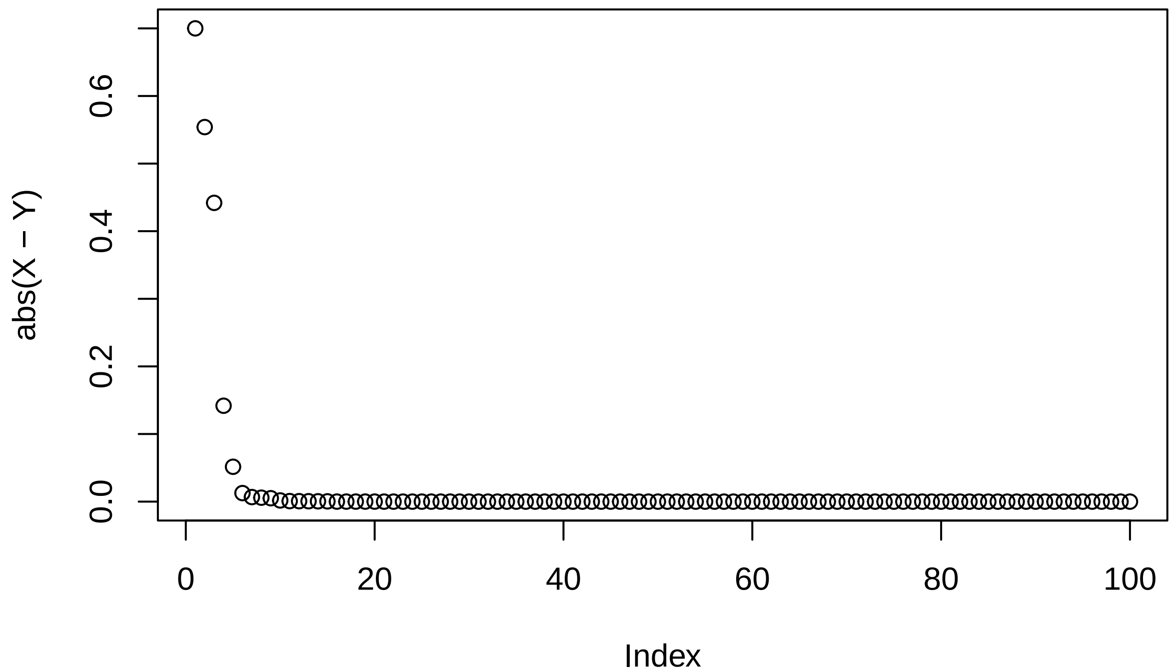


Dirichlet process

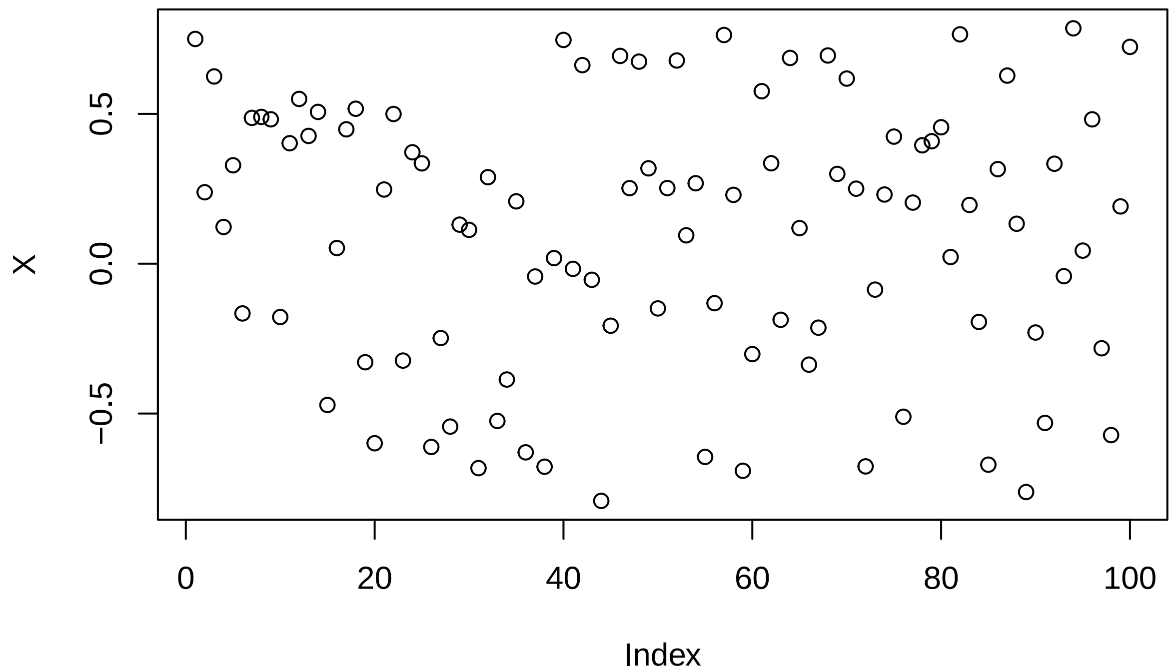
Define the Dirichlet process is

$$X_n = \sin[(1 - X_{n-1})\cos(\theta_n)], \theta_n \sim Unif(-\pi/2, 3\pi/2)$$

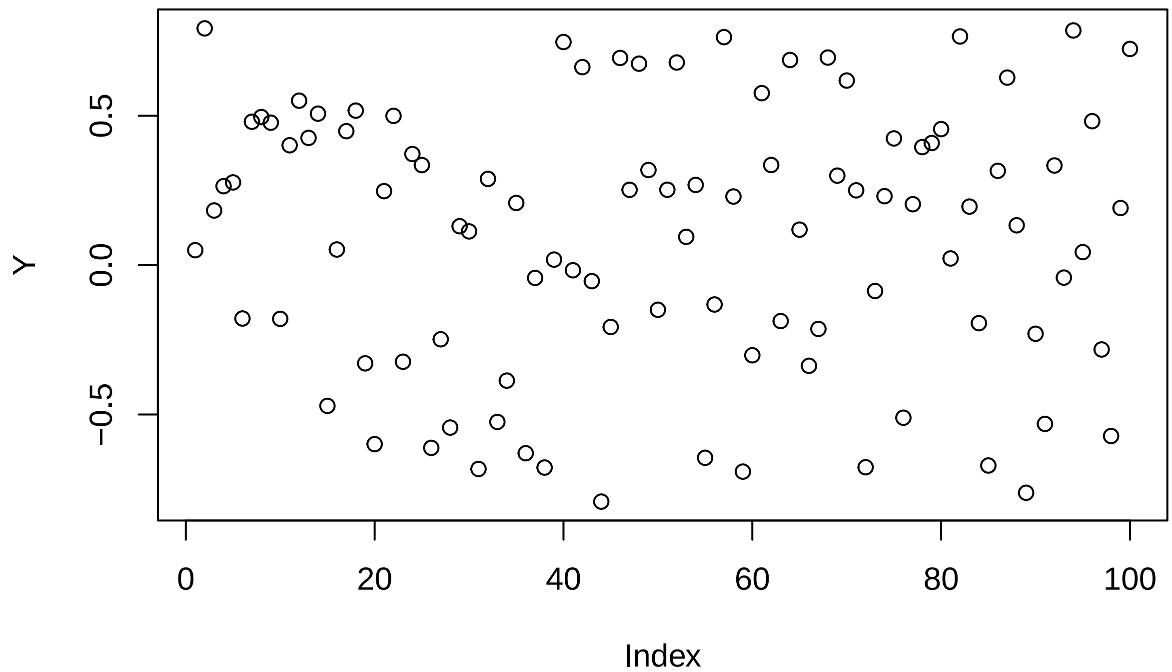
```
n <- 100
X <- 1:n
Y <- X
B <- runif(n, -pi/2, 3*pi/2)

X[1] <- 0.75
Y[1] <- 0.05
for(i in 2:n){
  # X[i] <- X[i-1]*cos(B[i])+sin(B[i])
  # Y[i] <- Y[i-1]*cos(B[i])+sin(B[i])
    X[i] <- sin((1-abs(X[i-1]))*cos(B[i]))
  Y[i] <- sin((1-abs(Y[i-1]))*cos(B[i]))
}

plot(abs(X-Y))
```



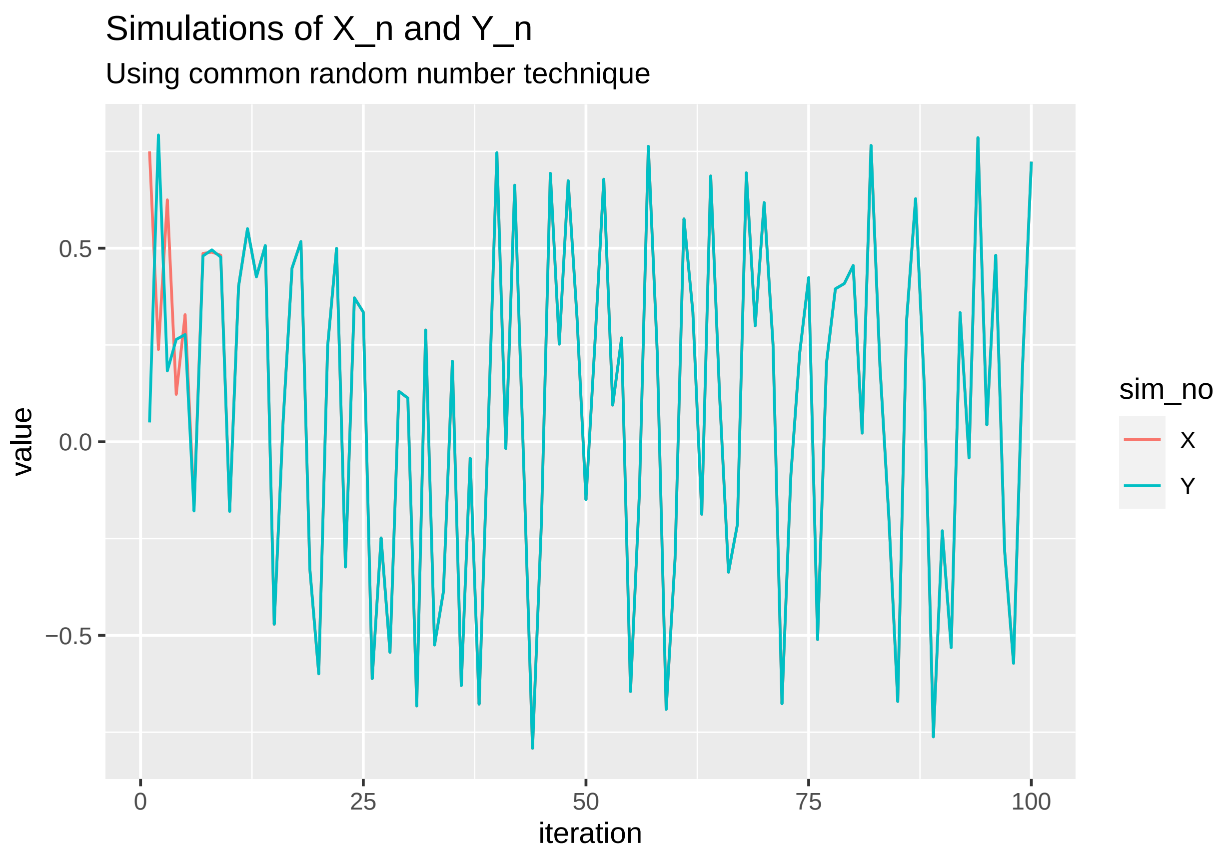

```r
plot(X)
```



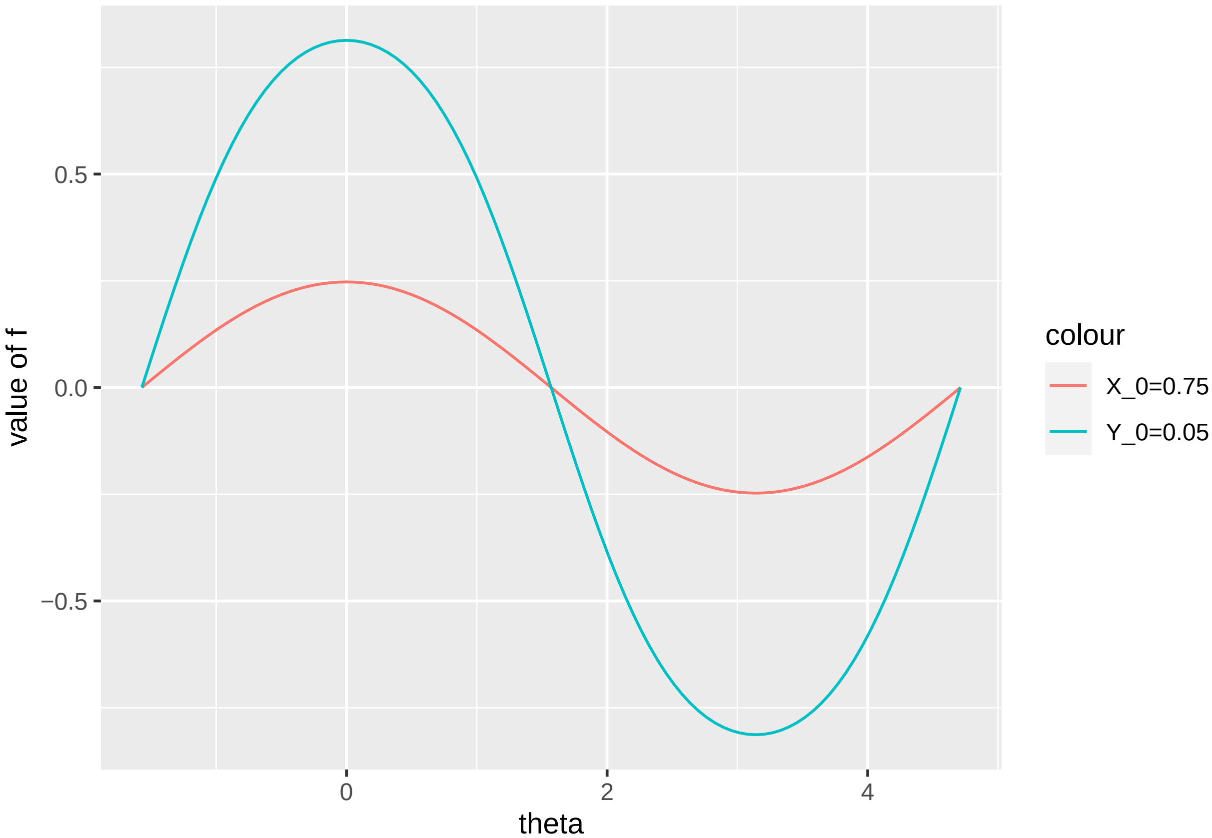

```r
plot(Y)
```




```
df <- data.frame(X,Y, iter=1:n)
df %>%
  pivot_longer(X:Y, names_to = "sim_no", values_to = "val") %>%
  ggplot(aes(x = iter, y = val)) +
  geom_line(aes(color = sim_no)) + labs(title = "Simulations of X_n and Y_n", subtitle = "Using common :
  xlab("iteration") + ylab("value")
```




```r
ggplot() +
  xlim(-pi/2, 3*pi/2) +
  geom_function(aes(colour = "X_0=0.75"), fun = function(x) sin(0.25*cos(x))) +
  geom_function(aes(colour = "Y_0=0.05"), fun = function(x) sin(0.95*cos(x))) +
  xlab("theta") + ylab("value of f")
```


0.5
0.0
−0.5
value of f
0
2
4
theta
colour
X_0=0.75
Y_0=0.05