Informe_Met-Has

La estadística bayesiana se usa para poder dar respuestas a incertidumbres sobre parámetros que afectan a distintas situaciones. Se usan dos fuentes de información para ello. La primera es lo que uno cree conocer sobre los parámetros previamente al realizar un experimento, es decir, la información a *priori*, y la segunda, los resultados que se obtienen al realizarlo, que se lo denomina como la *verosimilitud*. Juntando estas informaciones se obtiene lo que se conoce como la distribución a *posteriori* del o los parámetros. Pero para poder encontrar respuesta al desconocimiento sobre el tema se debe poder sacar muestras de la distribución a *posteriori*. Para realizar esto, una de las técnicas que se pueden utilizar es Metropolis-Hastings.

Metropolis-Hastings

Esta tecnica es un método de Montecarlo basado en cadenas de Markov que consiste, a grandes rasgos, en recorrer los distintos valores de una distribución de probabilidad generando un secuencia de valores posibles x de dicha distribución $x(1), x(2), ..., x(S)$ donde para obtener $x(i+1)$ usamos $x(i)$. De esta manera se quiere que el conjunto de los valores simulados se aproximen lo máximo posible a una muestra real de la distribución.

Esta técnica funciona de la siguiente manera:

1. En la iteración i estamos en el valor $x(i)$
2. En función del valor actual $x(i) = x$, proponemos un nuevo valor x' en función de $q(x' \mid x)$ siendo q la distribución de salto que uno debe proponer, que sera la que presente los puntos de salto posibles.
3. Decidimos si vamos a la nueva ubicación $x(i+1) = x'$ o si nos quedamos en $x(i+1) = x$:
   i. Calcular la probabilidad de salto:
      - $\alpha_{x \to x'} = min(1, f(x')f(x))$
   ii. Saltar a x' con probabilidad $\alpha_{x \to x'}$:

$$x^{(i+1)} = \begin{cases} x' \text{ con probabilidad } \alpha_{x \to x'} \\ x \text{ con probabilidad } \alpha_{x \to x'} \end{cases}$$

La función para utilizar el metodo es la siguiente:

```
# n: Tamaño de la cadena que se quiere generar

# p_inicial: Primer punto con el que se inicial la cadena. Es igual a 0 por defecto.

# v_random: Utilizar si se quiere empezar la cadena con un punto aleatorio. Introducir un ve

# d_objetivo: Función que calcule la densidad de la función que se quiere muestrear.

# r_propuesta: Función que genera un punto de la distribución de salto con la media sin espe

# d_propuesta: Función que devuelve la densidad en un punto de la distribución de salto con

sample_mh_1d = function(n, p_inicial = 0, d_objetivo, r_propuesta = rnorm, d_propuesta = dno
  stopifnot(n > 0)
  muestras = numeric(n)
  #Seleccion del primer valor de la cadena
  if (length(v_random) == 2){
    muestras[1] = runif(1,min(v_random),max(v_random))
  } else {
    muestras[1] = p_inicial
  }
  #Generación de todos los n-1 puntos siguientes de la cadena
  for (i in 2:n) {
    #Proposición del nuevo valor
    puntos = valor_salto_1d(muestras, r_propuesta, i)
    p_actual = puntos[1]
    p_nuevo = puntos[2]

    #Calculo de la probabilidad de salto
    prob = salto_1d(p_actual, p_nuevo, d_objetivo, d_propuesta)

    #Saltamos?
    salto = test_salto_1d(prob)

    #Salto
    muestras[i] = p_gen_1d(salto, p_actual, p_nuevo)
  }
    return(muestras)
}

#Donde las funciones utilizadas son:
```

```
#Proposición del nuevo valor

valor_salto_1d = function(muestras, r_propuesta, i){
  p_actual = muestras[i-1]
  p_nuevo = r_propuesta(p_actual)
  puntos = c(p_actual, p_nuevo)
  return(puntos)
}

#Calculo de la probabilidad de salto

salto_1d = function(p_actual, p_nuevo, d_objetivo, d_propuesta) {
  f_nuevo = d_objetivo(p_nuevo)
  f_actual = d_objetivo(p_actual)
  q_actual = d_propuesta(p_actual, p_nuevo)
  q_nuevo = d_propuesta(p_nuevo, p_actual)
  prob = (f_nuevo*q_actual) / (q_nuevo*f_actual)
  alfa = min(1, prob)
  return(alfa)
}

#Saltamos?

test_salto_1d = function(alfa){
  result = rbinom(1,1,alfa)
  return(result)
}

#Salto

p_gen_1d = function(result, p_actual, p_nuevo) {
  if (result){
    return(p_nuevo)
  } else {
    return(p_actual)
  }
}
```

Aplicación de Metropolis-Hastings en una dimensión

Distribución de Kumaraswamy

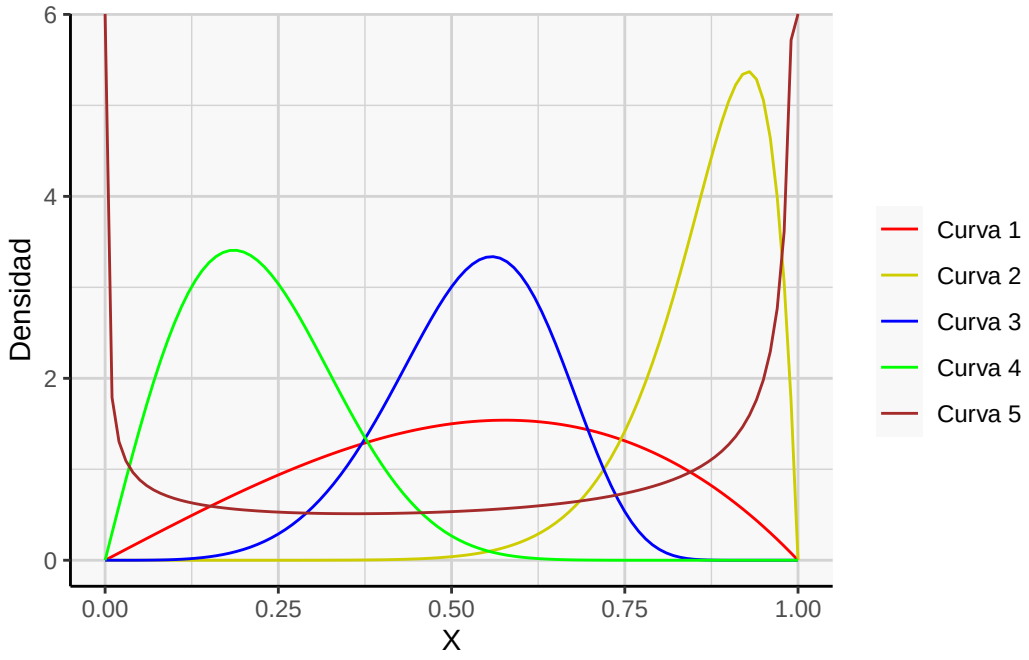
La distribución de Kumaraswamy es una distribución de probabilidad continua que se utiliza para modelar variables aleatorias con soporte en el intervalo $(0, 1)$. Su función de densidad se parece a la beta pero su expresión matematica resulta ser de computo mas sencillo. Dicha función es:

$$p(x \mid a, b) = abx^{a-1}(1 - x^a)^{b-1} \quad \text{con } a, b > 0$$

Para observar que forma tiene, se grafica su funcion de densidad para 5 valores de a y b.

	Valores				
	Curva 1	Curva 2	Curva 3	Curva 4	Curva 5
a	2	10	5	2	0.5
b	2	2	15	15	0.33

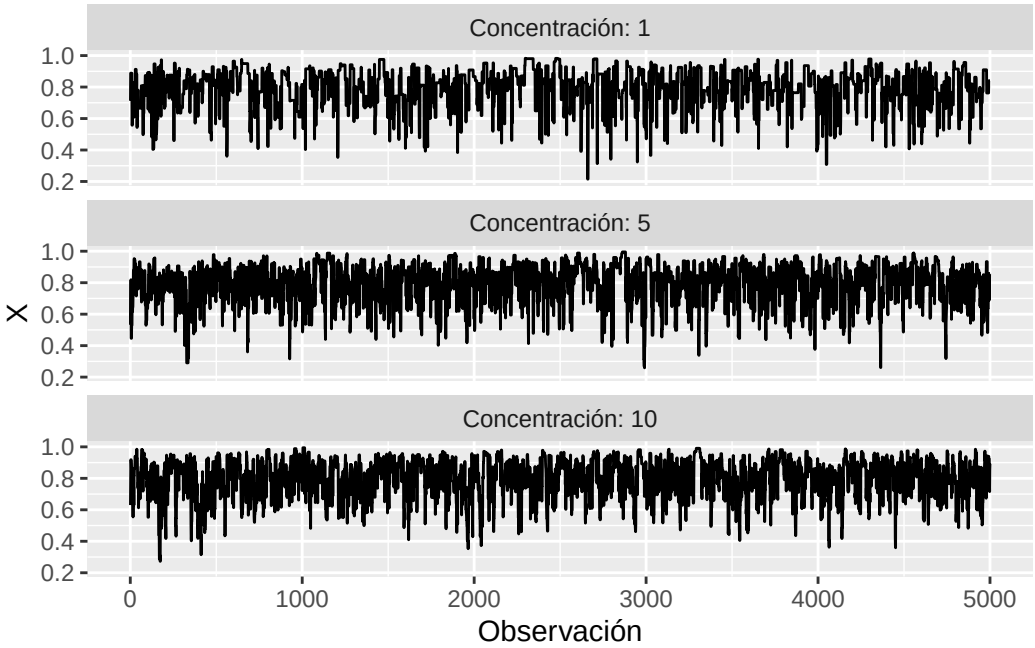
Table 1: Table to test captions and labels.



Es una funcion sencilla donde su forma de densidad resulta muy flexible al variar a y b por lo que se puede usar facilmente para la representacion de la probabilidad a *priori* de parametros que esten entre 0 y 1.

Usando la función de Metropolis-Hastings se obtiene un muestra de 5000 observaciones de la distribución de Kumaraswamy con $a = 6$ y $b = 2$. Se utiliza una distribución beta reparametrizada en su media y su concentración para proponer los nuevos valores y para obtener el primer valor de la cadena se genera aleatoriamente con igual probabilidad un punto entre 0 y 1 .

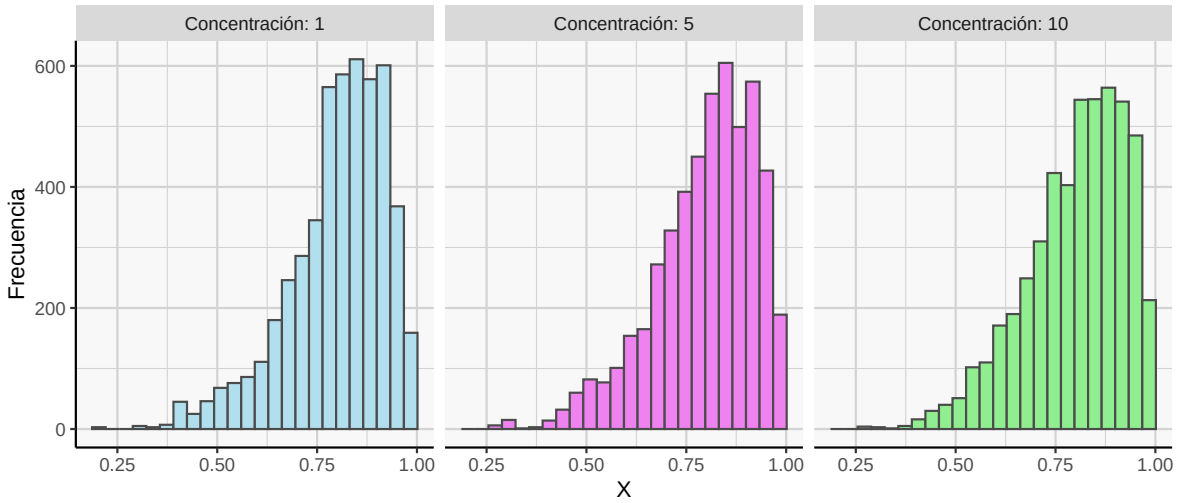
Se usan tres valores de concentración diferentes para evaluar la eficiencia del método.

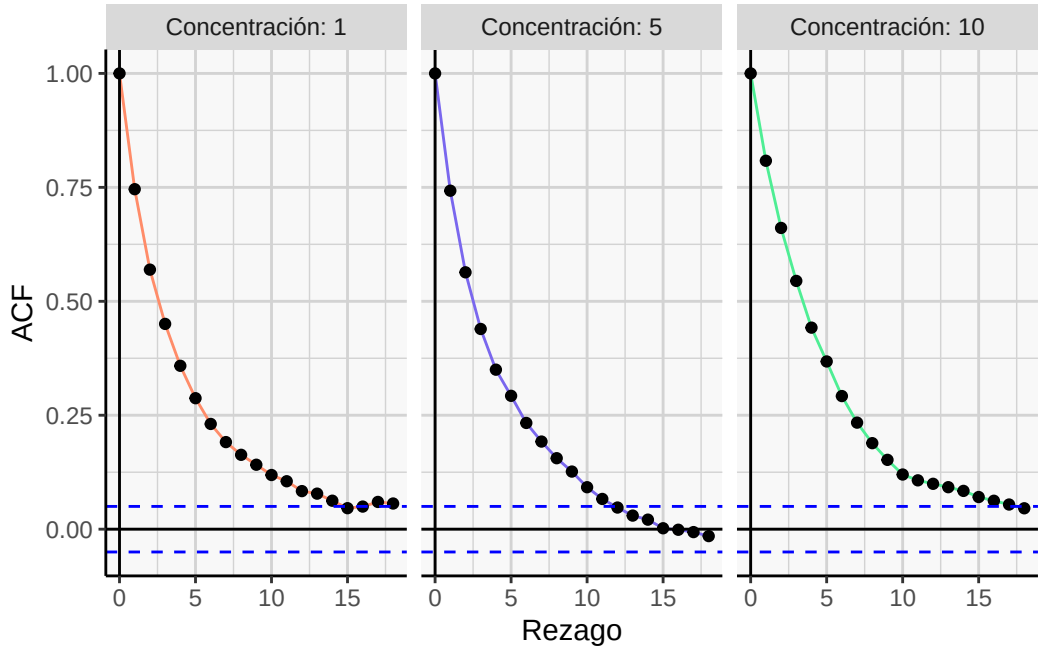


Concentración	1	5	10
Tasa de aceptación	0.219	0.509	0.618

Table 2: Table to test captions and labels.

En la figura x se observa que el metodo funciona mejor con una concentración alta porque oscila mas veces recorriendo la densidad de una manera mas eficiente. Esto se verifica observando las tasas de aceptación de los saltos en la tabla x.





Las tres cadenas convergen a la misma distribución y la cantidad de rezagos que tardan hasta llegar a una autocorrelación despreciable (<0.05) es baja y similar en todas por lo que el metodo funciona correctamente aún variando la dispersión de la distribución del salto.

	Media	Percentil 5	Percentil 95
X_1	0.7987838	0.5508091	0.9525783
X_5	0.7972954	0.5390545	0.9631994
X_{10}	0.8007066	0.5583269	0.9622339
$logit(X_1)$	1.5742044	0.2039405	3.0000926
$logit(X_5)$	1.5952465	0.1565376	3.2647543
$logit(X_{10})$	1.6198775	0.2343745	3.2378466

Table 3: Table to test captions and labels.

Donde:

$$\text{logit}(X) = \log\left(\frac{X}{1-X}\right) = \log(X) - \log(1-X)$$

Todas las medidas descriptivas de X son parecidas para las distintas cadenas generadas, pero si se le aplica la transformación logit a X esas diferencias minimas se agrandan un poco.

Aplicación de Metropolis-Hastings en dos dimensiones

La verdadera utilidad del método presentado anteriormente se encuentra en la extracción de muestras de distribuciones multivariadias. Igual que la situación anterior, no es necesario que las funciones que se quieren muestrear esten normalizadas. Se presentan dos situaciones en las que se puede aplicar la técnica. La primera, un ejemplo simple donde se obtienen muestras de la distribución normal bivariada donde este metodo funciona bien y la segunda, una función dificil de obtener muestras válidas en donde se pueden ver las fallas de Metropolis-Hastings.

Para realizar estas muestras, no sirve la función presentada en la primera parte del informe. Por eso se tiene la necesidad de modificar dicha función.

```
# Se utiliza como dstribución de salto propuesto una normal bivariada

# n: Tamaño de la cadena que se quiere generar

# p_inicial: Primer punto con el que se inicial la cadena. Es igual a 0 por defecto.

# d_objetivo: Función que calcule la densidad de la función que se quiere muestrear.

# matrix_var: Matrix de varianzas y covarianzas de la función de salto normal bivariada.

sample_mh_2d = function(n, p_inicial , d_objetivo, matrix_var = diag(1,2)){
  stopifnot(n > 0)
  muestras = matrix(nrow = n,ncol = 2)
```

```r
#Seleccion del primer valor de la cadena
  muestras[1,] = p_inicial

  #Generación de todos los n-1 puntos siguientes de la cadena
  for (i in 2:n) {
    #Proposición del nuevo valor
    puntos = valor_salto_2d(muestras, matrix_var, i)
    p_actual = puntos[1:2]
    p_nuevo = puntos[3:4]

    #Calculo de la probabilidad de salto
    prob = salto_2d(p_actual, p_nuevo, d_objetivo, matrix_var)

    #Saltamos?
    salto = test_salto_2d(prob)

    #Salto
    muestras[i,] = p_gen_2d(salto, p_actual, p_nuevo)
  }
  if (length(p_inicial) == 1) {
    return(as.vector(muestras))
  }
  return(muestras)
}









#Proposición del nuevo valor

valor_salto_2d = function(muestras, matrix_var, i){
  p_actual = muestras[i-1,]
  p_nuevo = rmvnorm(1, mean = p_actual, sigma = matrix_var)
  puntos = c(p_actual, p_nuevo)
  return(puntos)
}

#Calculo de la probabilidad de salto
```

```
salto_2d = function(p_actual, p_nuevo, d_objetivo, matrix_var) {
  f_nuevo = d_objetivo(p_nuevo)
  f_actual = d_objetivo(p_actual)
  q_actual = dmvnorm(p_actual, p_nuevo, sigma = matrix_var)
  q_nuevo = dmvnorm(p_nuevo, p_actual, sigma = matrix_var)
  prob = (f_nuevo*q_actual) / (q_nuevo*f_actual)
  alfa = min(1, prob)
  return(alfa)
}

#Saltamos?

test_salto_2d = function(alfa){
  result = rbinom(1,1,alfa)
  return(result)
  }

#Salto

p_gen_2d = function(result, p_actual, p_nuevo) {
  if (result){
     return(p_nuevo)
  } else {
    return(p_actual)
  }
  }
```

Distribución normal bivariada

La distribución a muestrear en este caso será una normal bivariada.

$$p(x \mid \mu, \Sigma) = \frac{1}{(2\pi)^{k/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Donde μ es el vector de esperanzas y Σ la matrix de varianzas y covarianzas.

Se controlará que tan buenas son las muestras obtenidas a traves de la evaluación de la cadena generada por el método y de qué tanto se asimilan las probabilidades estimadas por la distribución muestral obtenida a las reales. La normal bivariada de interés viene dada por los siguientes parámetros.

$$\mu^* = \begin{bmatrix} 0.4 \\ 0.75 \end{bmatrix} \quad \Sigma^* = \begin{bmatrix} 1.35 & 0.4 \\ 0.4 & 2.4 \end{bmatrix}$$

Usando la función de Metropolis-Hastings se obtiene un muestra de 5000 observaciones. Para obtener las muestras, primero se debe seleccionar la matriz de covarianzas de la distribución de salto. Esto se logra evaluando el número de muestras efectivas de las dos dimensiones de las muestras generadas por separado, variando los valores en la matriz y quedandose con la que tenga el mayor número de muestras efectivas totales.

El calculo realizado es:

$$N_{eff} = \frac{S}{1 + 2\sum_{k=1}^{N} ACF(k)}$$

Donde S es el número de unidades generadas en la cadena, $ACF(k)$ es la autocorrelación de la cadena con k rezagos y N es la cantidad de rezagos hasta que $ACF(k) < 0.05$ por primera vez.

```
#poner calculo
var = matrix(0,nrow = 85, ncol = 3)
var[85,] = c(1.85, 2.15, 0.5)
```

Los valores

$$\Sigma_{Salto} = \begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{bmatrix}$$

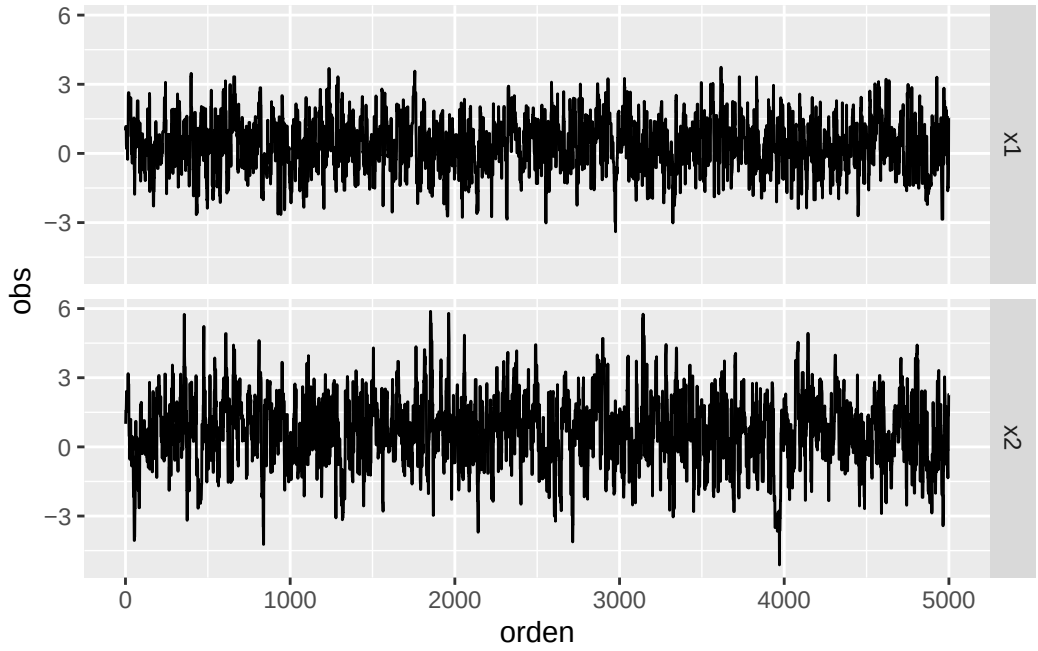
Con:

σ_1^2 variando entre 0.85 y 1.85. σ_2^2 variando entre 1.90 y 2.90. σ_{12} variando entre 0.20 y 0.60.

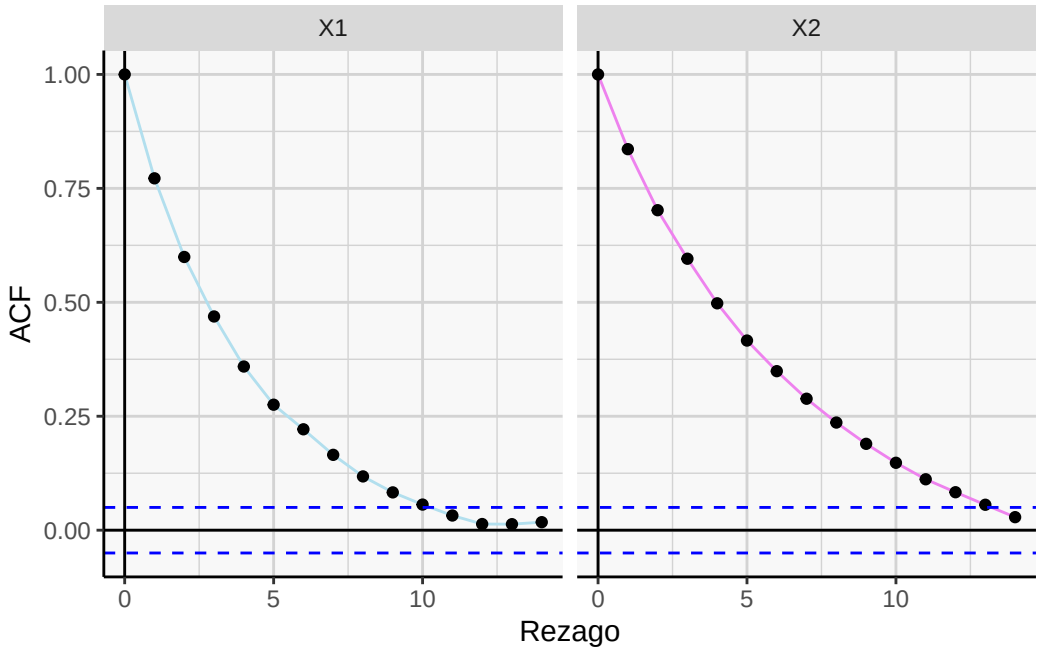
La combinación de dichos valores con el mayor número de muestras efectivas totales es 1.85, 2.15 y 0.5 para σ_1^2, σ_2^2 y σ_{12} respectivamente.

```
set.seed(69)

d_objetivo_normal = function (x) dmvnorm(x, mean = c(.4, .75), sigma = matrix(c(1.35, .4, .4


muestra_normal = sample_mh_2d(5000, p_inicial = c(1,1), d_objetivo = d_objetivo_normal, matri

muestra_normal_df = data.frame(obs = c(muestra_normal[,1],muestra_normal[,2]),
                               orden = rep(1:5000,2),
                               cadena = rep(c("x1", "x2"), each = 5000))
```

```
ggplot(muestra_normal_df) + geom_line(aes(x = orden, y = obs)) + facet_grid(vars(cadena))
```




```
nef_norm = apply(muestra_normal, 2, FUN = neef)
```



Las cadenas en la figura x se ven relativamenta aleatorias y parece recorrer el soporte de la función objetivo, que es indicio de una autocorrelación baja entre las observación generadas. Esto se aprecia en la figura x+1, donde la autocorrelación de ambas cadenas disminuye rápidamente al aumentar el rezago. El número de muestras efectivas para x_1 y x_2 es 684.7769494 y 496.2193513 respectivamente. Estos son valores relativamente grandes, que dan una imagen positiva de la calidad de la muestra generada.

```
Prob_mh_norm_1 = mean(muestra_normal[, 1] > 1 & muestra_normal[, 2] < 0)
Prob_mh_norm_2 = mean(muestra_normal[, 1] > 1 & muestra_normal[, 2] > 2)
Prob_mh_norm_3 = mean(muestra_normal[, 1] > .4 & muestra_normal[, 2] > .75)


rmv = rmvnorm(100000, mean = c(.4, .75), sigma = matrix(c(1.35, .4, .4, 2.4), nrow = 2))
Prob_rmv_1 =pmvnorm(lower = c(1, -Inf), upper = c(Inf, 0), mean = c(0.4, 0.75), sigma = matri
Prob_rmv_2 =pmvnorm(lower = c(1, 2), upper = c(Inf, Inf), mean = c(0.4, 0.75), sigma = matri
Prob_rmv_3 =pmvnorm(lower = c(0.4, 0.75), upper = c(Inf, Inf), mean = c(0.4, 0.75), sigma = m

Error1 = round(abs(Prob_mh_norm_1 - Prob_rmv_1)/Prob_rmv_1, 4)
Error2 = round(abs(Prob_mh_norm_2 - Prob_rmv_2)/Prob_rmv_2, 4)
Error3 = round(abs(Prob_mh_norm_3 - Prob_rmv_3)/Prob_rmv_3, 4)
```

Se calculan tres probabilidades usando las muestras generadas y se comparan con las reales para ver que tan precisas son.

Probabilidades	$\hat{\mathbf{X}}$	$\mathbf{X}$	Error relativo
$\Pr(X_1 > 1, X_2 < 0)$	0.0624	0.0683	0.0857
$\Pr(X_1 > 1, X_2 > 2)$	0.0834	0.087	0.0415
$\Pr(X_1 > 0.40, X_2 > 0.75)$	0.2814	0.2857	0.0149

Table 4: Table to test captions and labels.

En la tabla 4 se observa que cuando se quiere estimar la probabilidad de un evento correspondiente a un área lejana de la masa central de probabilidades, la precisión de la aproximación disminuye, es decir, aumenta el error. Aún así, el método logra obtener una muestra aceptable al no ser tan grande el error relativo en dichos casos.

Función de Rosenbrock

El "valle de Rosenbrock", "banana de Rosenbrock" o simplemente función de Rosenbrock es una función matemática utilizada comunmente en problemas de optimización y prueba para algoritmos de optimización numérica.

La función está definida por:

$$f(x, y) = (a - x)^2 + b(y - x^2)^2$$

En estadística bayesiana, en ciertos casos, se obtiene una densidad de distribución a *posteriori* que toma una forma semejante a dicha función.

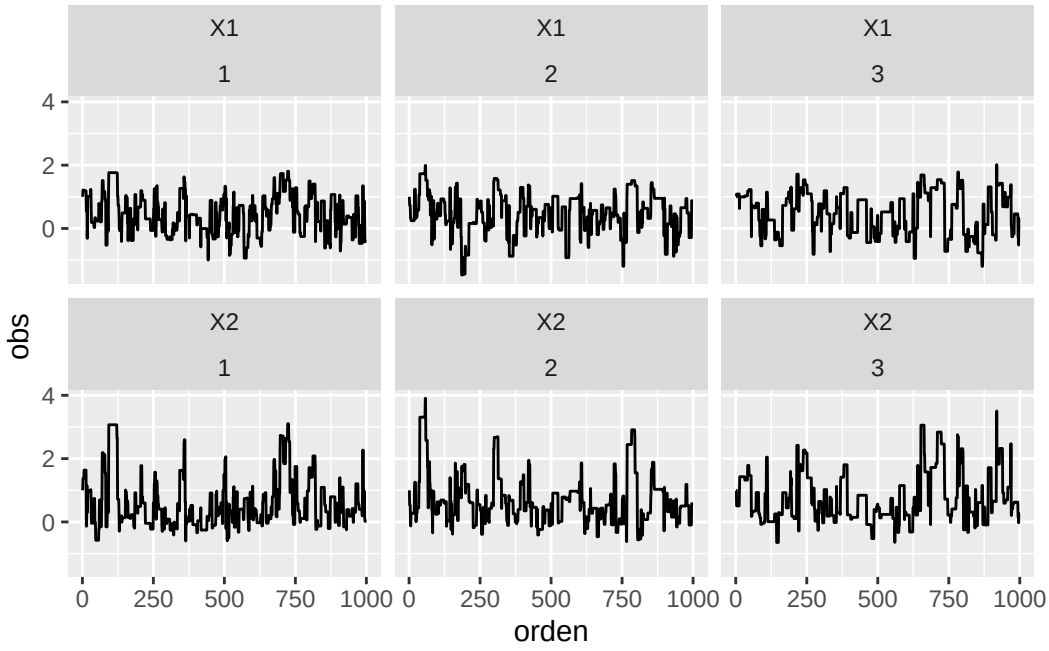
Un ejemplo de esto es la función p^*:

$$p^*(x_1, x_2 \mid a, b) = \exp\left\{-\left[(a - x_1)^2 + b(x_2 - x_1^2)^2\right]\right\}$$

Se quiere probar que tan bien puede Metropolis-Hastings generar muestras de dicha función de probabilidad y para ello se proponen distintas matrices de varianzas y covarianzas de la distribución de salto propuesta evaluando el rendimiento obtenido con cada una de ellas. Se utilizan los valores 0.5 y 5 para los parámetros a y b respectivamente.

```
rosenbrock = function (x) ((.5 - x(1))^2) + 5 * ((x(2) - (x(1)^2))^2)

d_objetivo_rosenbrock = function (x) exp((-1)*(((.5 - x[1])^2) + 5 * ((x[2] - (x[1]^2))^2)))

set.seed(69)

muestra_rosenbrock_1 = sample_mh_2d(1000, p_inicial = c(1,1), d_objetivo = d_objetivo_rosenbr
```

```
                                matrix_var = matrix(c(1, .7, .7, 1), nrow = 2))
muestra_rosenbrock_2 = sample_mh_2d(1000, p_inicial = c(1,1), d_objetivo = d_objetivo_rosenbr
                                    matrix_var = matrix(c(1, .3, .3, 1), nrow = 2))
muestra_rosenbrock_3 = sample_mh_2d(1000, p_inicial = c(1,1), d_objetivo = d_objetivo_rosenbr
                                    matrix_var = matrix(c(1,0,0,2), nrow = 2))


muestra_rosenbrock_df_1 = data.frame(obs = c(muestra_rosenbrock_1[,1],muestra_rosenbrock_1[,2
                                orden = rep(1:1000,2),
                                cadena = rep(c("X1", "X2"), each = 1000),
                                sigma = rep("1",1000))


muestra_rosenbrock_df_2 = data.frame(obs = c(muestra_rosenbrock_2[,1],muestra_rosenbrock_2[,2
                                     orden = rep(1:1000,2),
                                     cadena = rep(c("X1", "X2"), each = 1000),
                                     sigma = rep("2",1000))

muestra_rosenbrock_df_3 = data.frame(obs = c(muestra_rosenbrock_3[,1],muestra_rosenbrock_3[,2
                                     orden = rep(1:1000,2),
                                     cadena = rep(c("X1", "X2"), each = 1000),
                                     sigma = rep("3",1000))

muestra_rosenbrock_df = rbind(muestra_rosenbrock_df_1, muestra_rosenbrock_df_2, muestra_rosen


ggplot(muestra_rosenbrock_df) + geom_line(aes(x = orden, y = obs)) +
  facet_wrap(~cadena + sigma, ncol = 3)
```


X1
1
X1
2
X1
3
X2
1
X2
2
X2
3
obs
4
2
0
0
250
500
750
1000
orden