Cómputo científico para probabilidad y estadística. Tarea 6. MCMC: Metropolis-Hastings.

Juan Esaul González Rangel

Octubre 2023

1. Simular $n = 5$ y $n = 40$ v.a Bernoulli $Be(1/3)$; sea r el número de éxitos en cada caso.

   La simulación de las variables aleaotorias se logra con el siguiente código,

```
# Samples of Bernoulli
sample5 = np.random.binomial(5,1/3,1)
sample40 = np.random.binomial(40,1/3,1)
r5 = np.sum(sample5)
r40 = np.sum(sample40)
```

   Usando la semilla 57 encontramos los siguiente valores para r_5 y r_{40},

$$r_5 = 3,$$
$$r_{40} = 15.$$

2. Implementar el algoritmo Metropolis-Hastings para simular de la posterior

$$f(p|\bar{x}) \propto p^r(1-p)^{n-r}\cos(\pi p)I_{[0,1/2]}(p),$$

   con los dos casos de n y r de arriba. Para ello poner la propuesta $(p'|p) = p' \sim Beta(r+1, n-r+1)$ y la distribución inicial de la cadena $\mu \sim U(0, 1/2)$.

   El algoritmo se encuentra implementado en la función `MH_beta()` del archivo `Tarea6.py`. La función toma tres parámetros con nombre `iterations`, `trials` y `successes`. El segundo y tercer argumentos se refieren a la cantidad de ensayos Bernoulli independientes que se realizaron y cuáles de ellos resultaron en éxito.

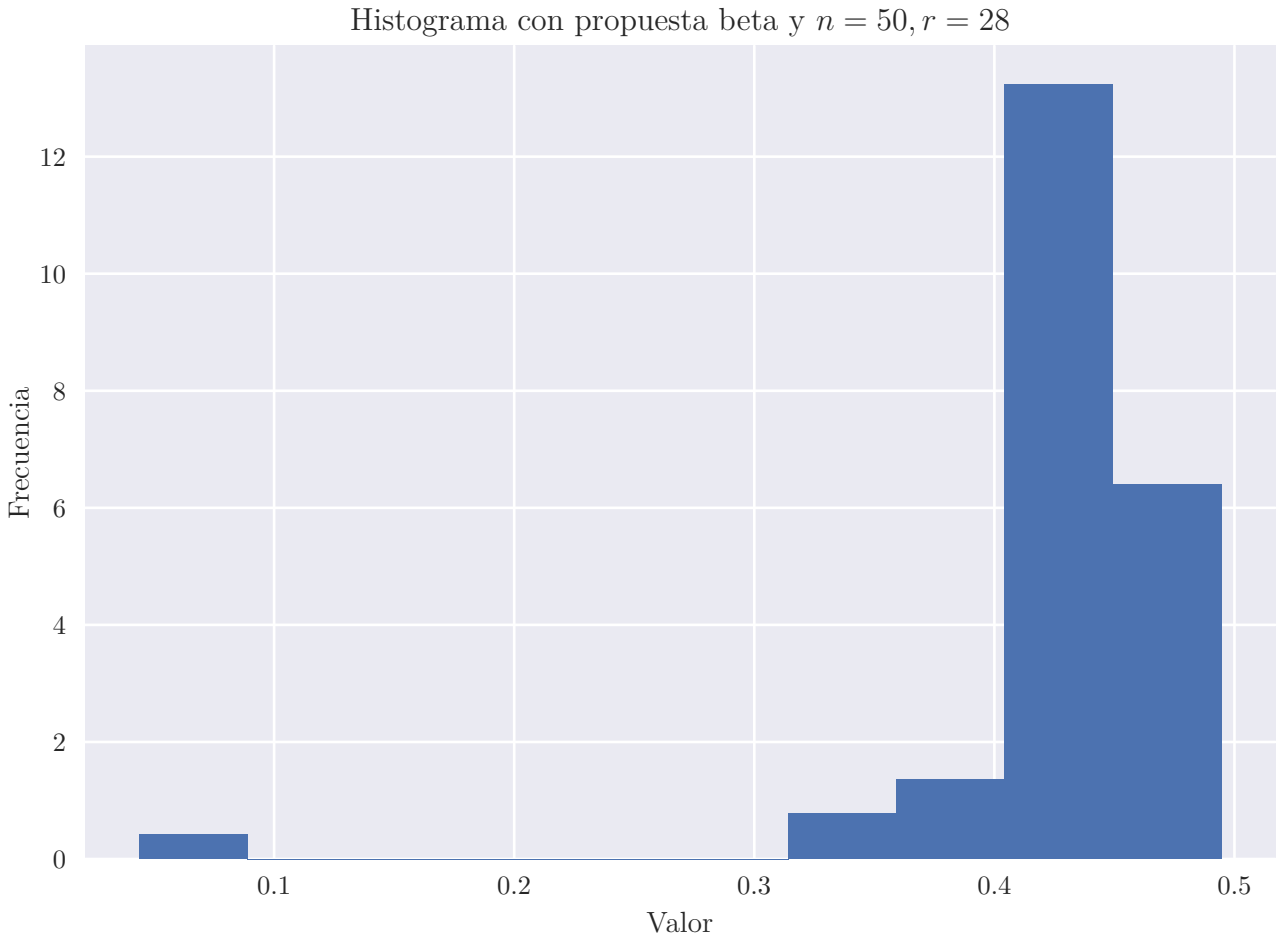
   El primer argumento representa la cantidad de iteraciones que realizará el algoritmo. Este número coincide con la cantidad de valores de la muestra que se obtienen, pero algunos serán repetidos. Una manera de evaluar qué tan buena es la propuesta de transición dada para el problema es cuestión es medir qué tanto tiempo permanece la cadena en un mismo estado. Cuando una cadena tarda más en aceptar la transición, es natural esperar que tarde más en converger a la distribución estacionaria.

   El siguiente es un ejemplo sencillo de uso de la función,

```
simple_sample = MH_beta(1000,50,28)
plt.hist(simple_sample,density=True)
```

   En el código anterior, se considera que se obtuvo una muestra de 50 ensayos Bernoulli de los cuáles 28 fueron éxitos.

   El histograma de la muestra anterior es el siguiente,



3. Argumentar porque la cadena es f-irreducible y porque es ergódica. Implementar el algoritmo con los datos descritos y discutir los resultados.

   Por diseño del kernel de transición de nuestra cadena, este satisface las ecuaciones de balance detallado con $f(p \mid \bar{x})$, de donde se sigue que f es estacionaria para K. Por lo tanto, basta mostrar que K es $f(p \mid \bar{x})$-irreducible para gerantizar que la cadena es Harris-estacionaria, y ergódica.

   Mostramos a continuación que el kernel construido efectivamente es $f(p \mid \bar{x})$-irreducuble.

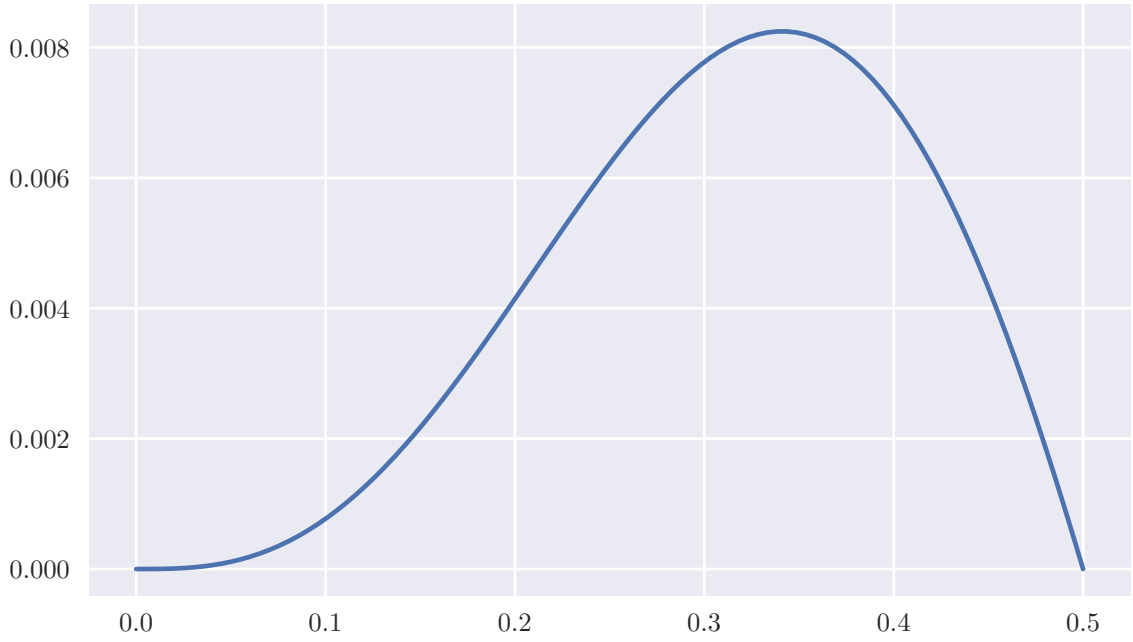
   Demostración. Denotemos por μ_f a la medida de probabilidad asociada a $f(p \mid \bar{x})$, y sea A un conjunto medible tal que $\mu_f(A) > 0$. Notemos que el soporte de f es un subconjunto del soporte de $\cos(\pi p)\mathbb{1}_{[0,1/2]}(p)$ y el soporte de $p^r(1-p)^{n-r}$. Entonces el soporte de f es un subconjunto del intervalo $(0, 1/2)$, y se sigue que $A \subset (0, 1/2)$.

   Sea $a \in A$, entonces $K(x, a) = q(a \mid x)\frac{f(y)q(a|y)}{f(a)q(y|a)} + (1 - r(a))\delta_a(y)$. Como la densidad beta es positiva en $(0, 1)$, en particular es positiva en $(0, 1/2)$. Además, si el punto de transición propuesto está en el soporte de f, entonces existe posibilidad positiva de aceptar la transición, luego, $K(x, a) > 0$ para $a \in A$. Es decir, con la densidad escogida, el kernel de transición en un pasoes positivo sobre el soporte de A.

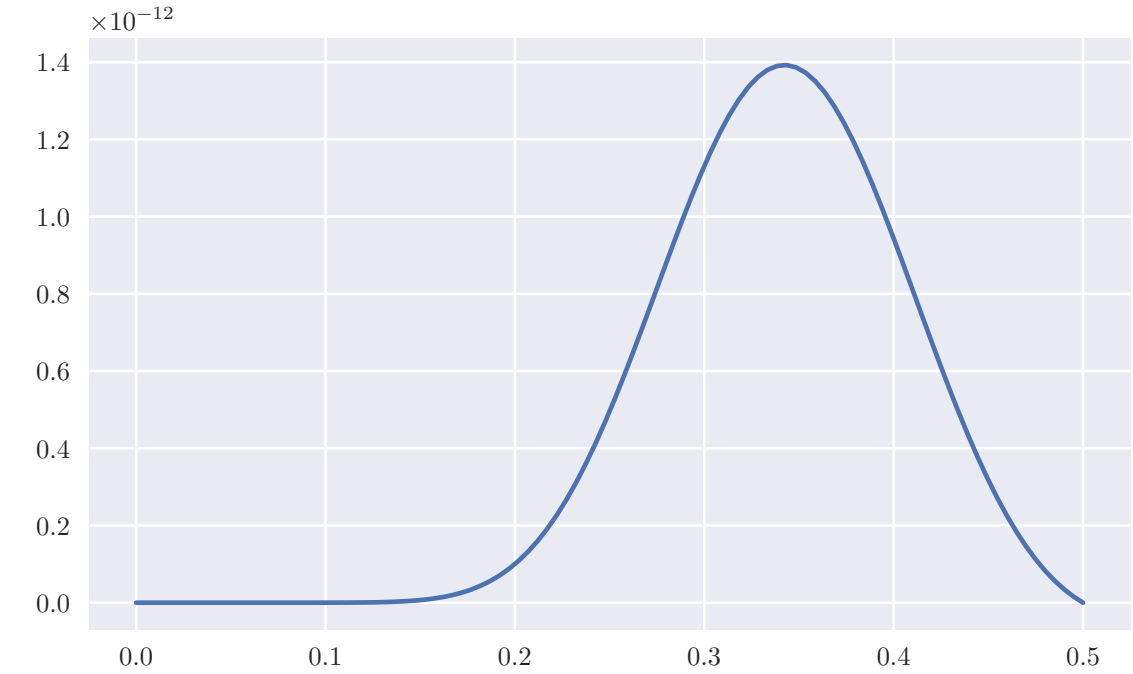
   □

   Antes de implementar el algoritmo de Metropolis-Hastings para simular, graficamos las densidades objetivo con los valores que tenemos de n y r.

Múltiplo de la densidad objetivo con $n = 5, r = 3$



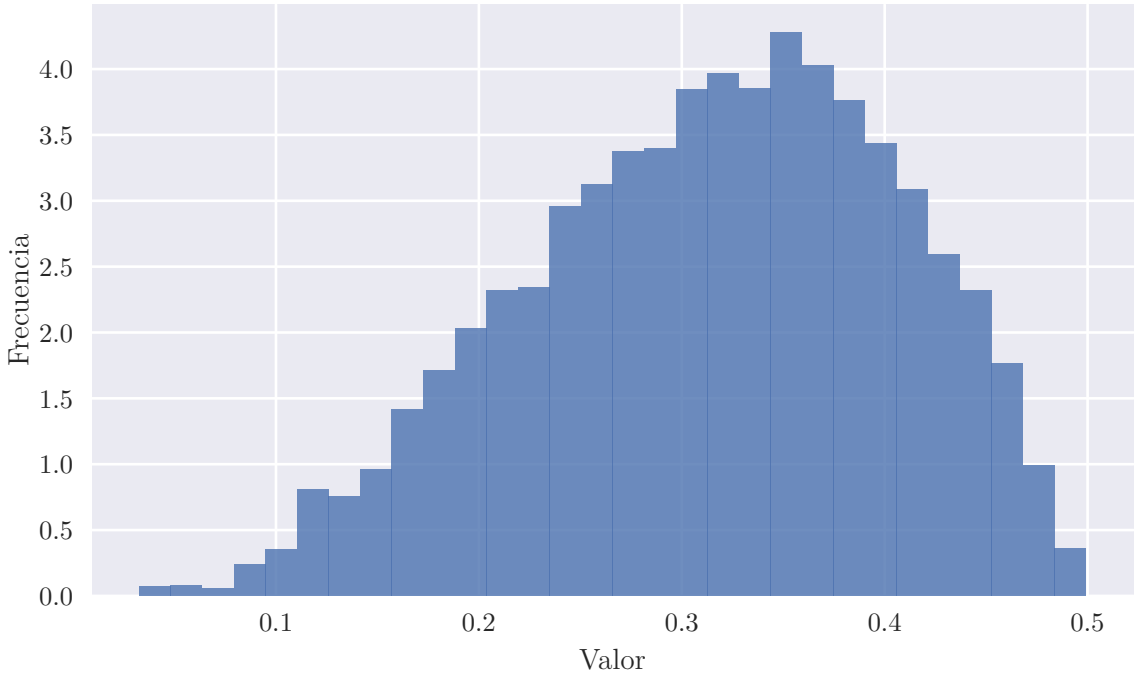
Múltiplo de la densidad objetivo con $n = 40, r = 15$



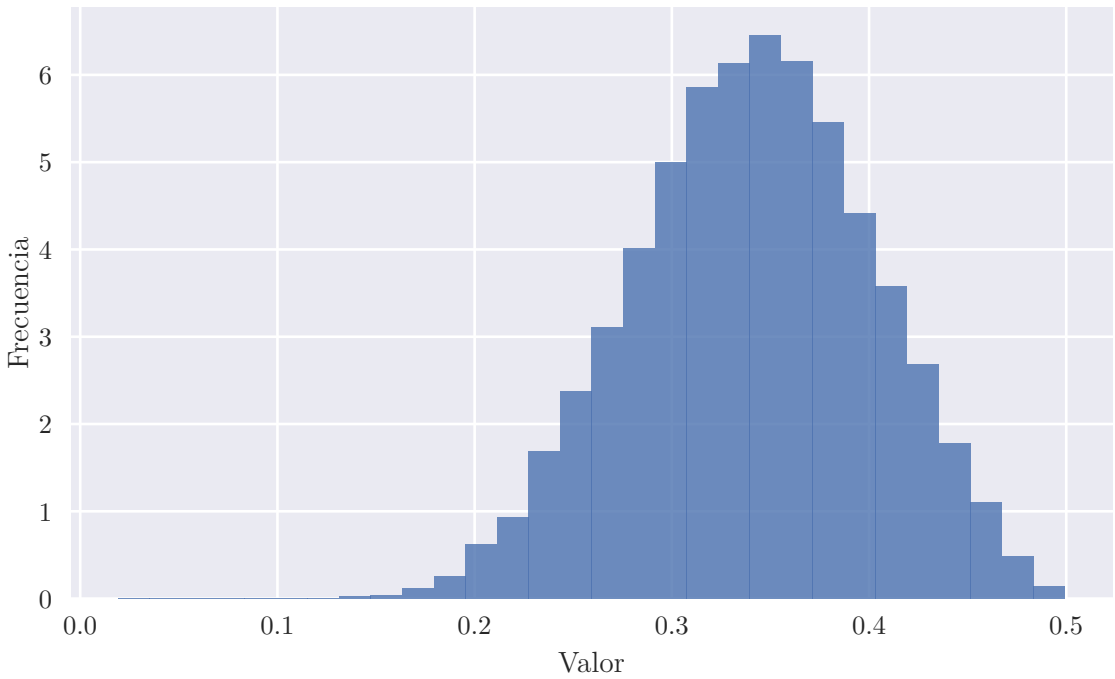
Estas gráficas en realidad corresponden a múltiplos de la densidad, ya que no conocemos sus constantes de normalización. Precisamente parte de la utilidad del algoritmo de Metropolis-Hastings es que únicamente necesitamos conocer un múltiplo de la densidad objetivo para muestrear de ella.

Al implementar el algoritmo con los datos descritos obtenemos los siguientes histogramas

Histograma para la muestra con propuesta beta y $n = 5, r = 3$



Histograma para la muestra con propuesta beta y $n = 40, r = 15$



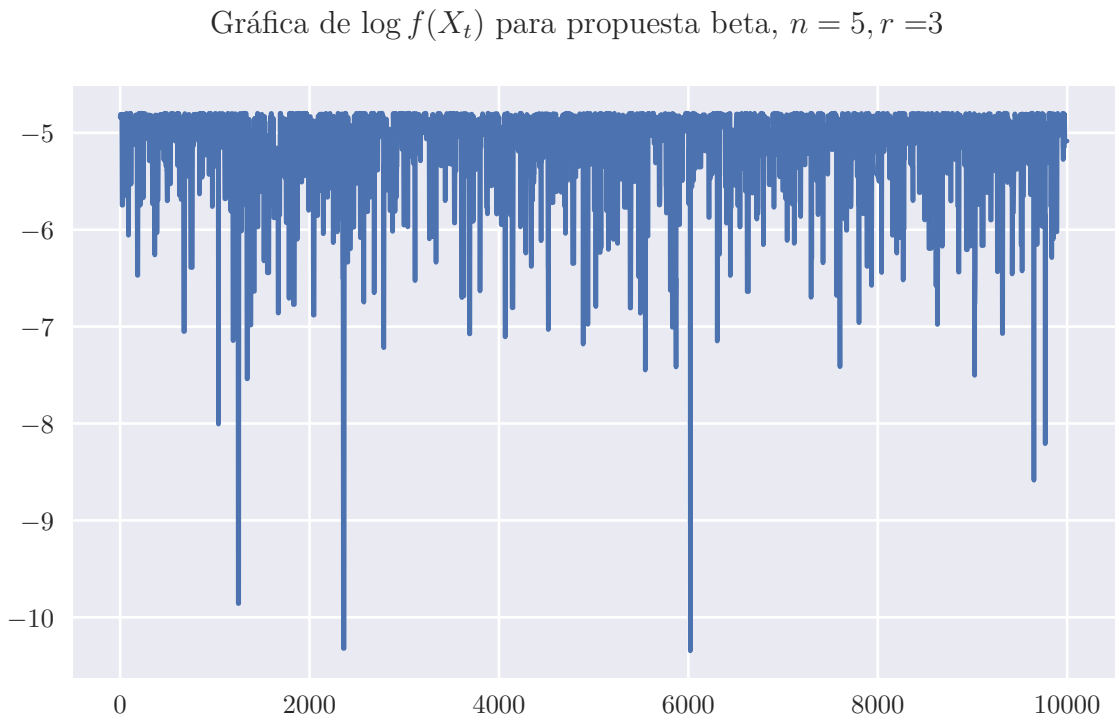
Notamos que en ambos casos los histogramas son parecidos a la densidad objetivo, aunque presentan diferencias, como más masa cerca del punto cero. Esto puede deberse a que el punto inicial uniforme comenzó cerca del cero, lo que ocasiona muestreo de puntos poco probables para la densidad. No podemos afirmar que desde el primer valor que obtuvimos estamos muestreando de f, ya que la convergencia a la distribución objetivo es asintótica. Para poder hablar de un muestreo real de f sería necesario descartar una cantidad fija de valores que se consideran observaciones

de la cadena que aún no son "suficientemente cercanas" a la distribución de f.

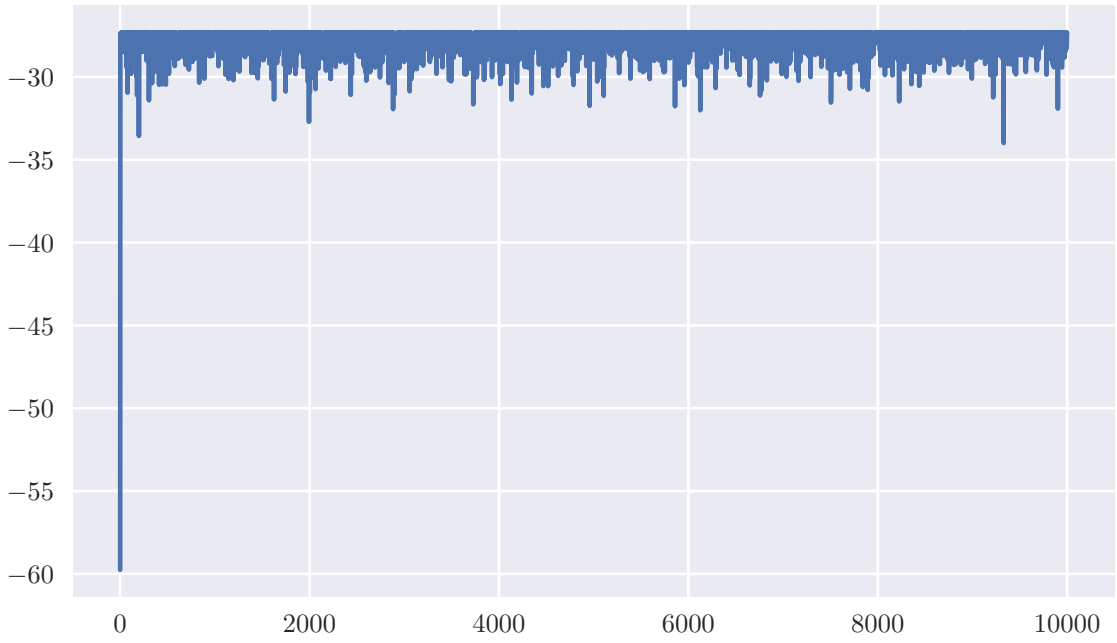
Otra observación importante es que, a pesar de que se obtuvieron 1,000 valores de muestreo, estos no constituyen una muestra i. i. d., pues al provenir de observaciones simuladas de una cadena de Márkov existe una correlación. Para poder considerar a la muestra independiente, sería necesario tomar valores suficientemente espaciados a lo largo de toda la muestra, de manera que la correlación entre ellos sea cercana a cero.

La manera de lidiar el problemas anterior se basa en heurísticas que se revisarán en el curso y se incluirán en tareas posteriores.

Para solucionar el problema de muestrear desde los primeros valores, podemos usar el logaritmo de la densidad evaluada en el proceso X_t y revisar en qué momento este valor parece estabilizarse. Las siguientes figuras muestran como se comporta esta función para los dos muestreos, evaluados en un intervalo de tiempo relativamente chico para apreciar mejor el punto de cambio



Gráfica de $\log f(X_t)$ para propuesta beta, $n = 40, r = 15$



Como podemos notar, en la segunda figura, el algoritmo comienza con un valor de log-densidad muy alejado del resto de los puntos, pero eventualmente se estabiliza. Podemos considerar que una vez que este se ha estabilizado estamos muestreando desde la distribución objetivo. El primer punto en que se estabiliza la densidad parece ser alrededor de $t = 50$

En la segunda figura (para $n = 5, r = 3$) el logaritmo de la densidad parece variar de manera uniforme a lo largo de toda la trayectoria del proceso. Podemos concluir que según esta heurística, es posible considerar desde el primer valor como parte del muestreo de la distribución objetivo.

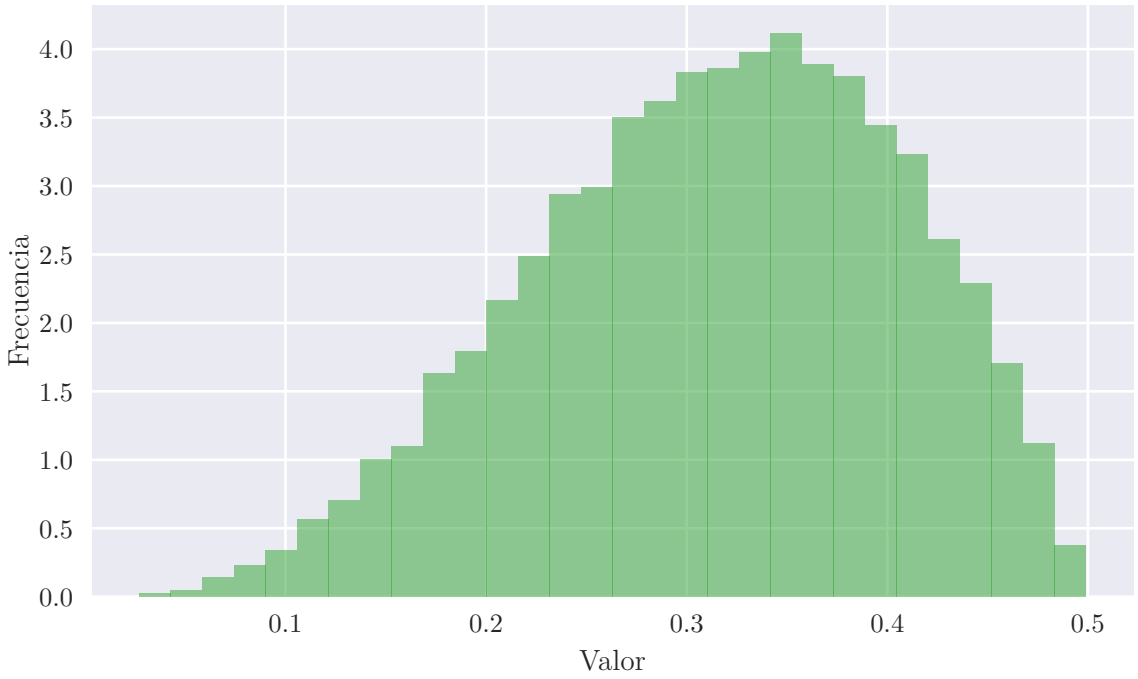
Con base en lo anterior, descartamos las primeras 50 iteraciones del muestreo con $n = 40$ y $r = 15$, miestras que conservamos todas las observaciones cuando $n = 5, r = 3$.

4. Implementar el algoritmo Metropolis-Hastings con la posterior de arriba tomando una propuesta diferente.

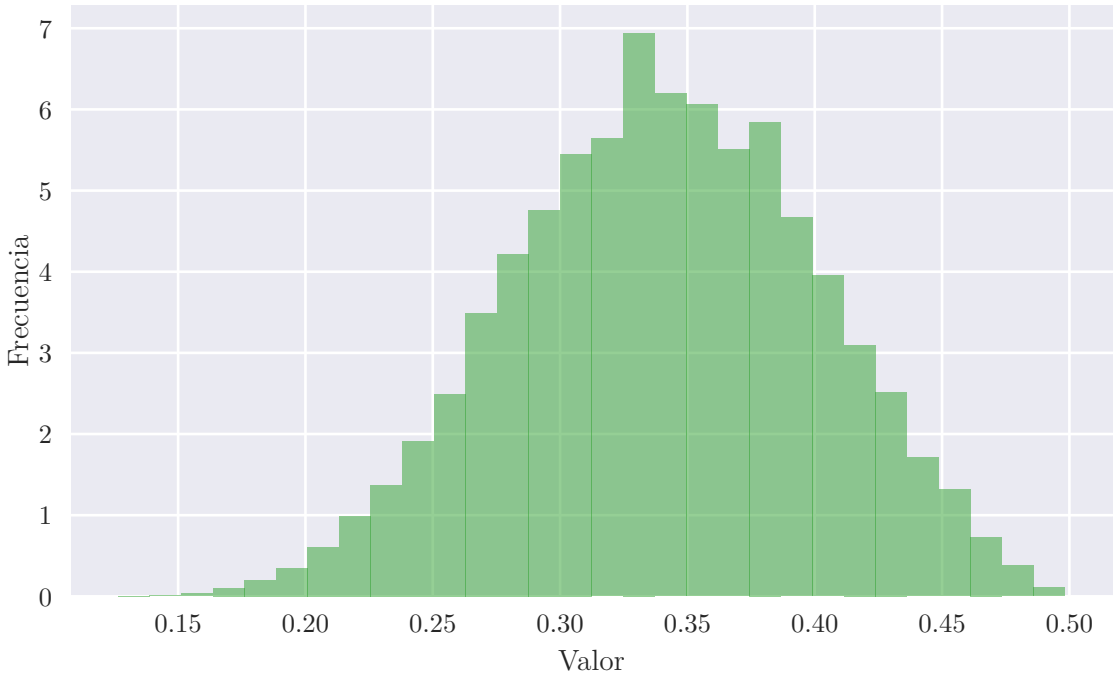
   Como buscamos distribuciones que tengan soporte en $(0, 1/2)$, una propuesta puede ser la uniforme continua en $(0, 1/2)$. Una variante del algoritmo de Metropolis-Hastings que utiliza una densidad uniforme como función de transición de la cadena original se encuentra implementada en el archivo `Tarea6.py` con el nombre de `MH_unif()`. La función toma como entrada los mismos argumentos que la función `MH_beta()`.

   Al hacer 1,000 iteraciones de `MH_unif()` para r_5 y 1,000 para r_{40} encontramos los histogramas siguientes,

Histograma para la muestra con propuesta $U(0,1)$ y $n = 5, r = 3$

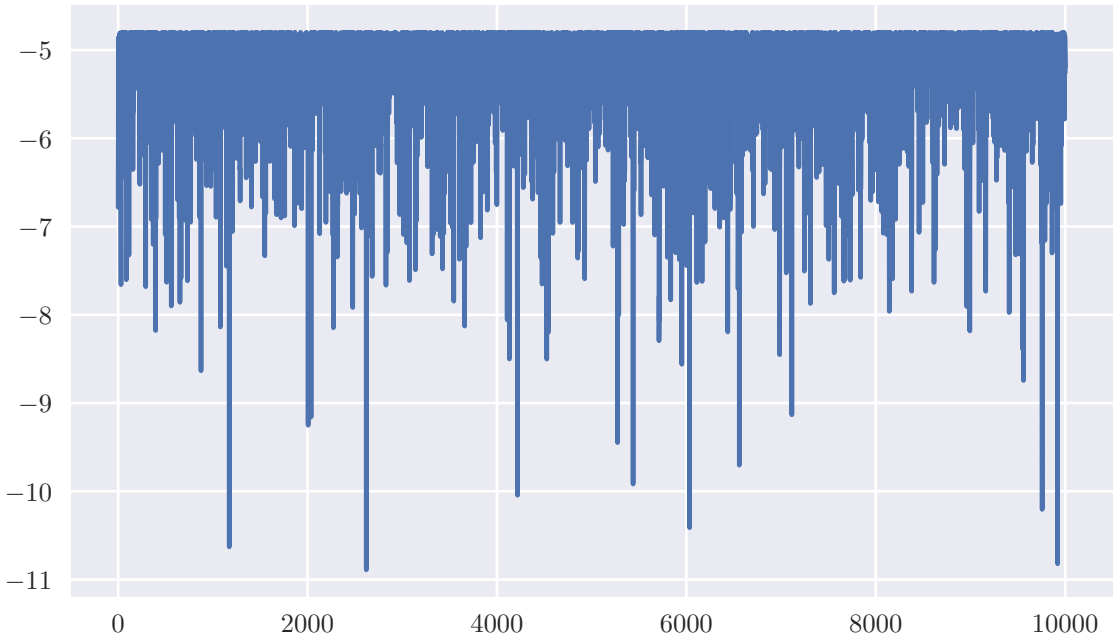


Histograma para la muestra con propuesta $U(0,1)$ y $n = 40, r = 15$

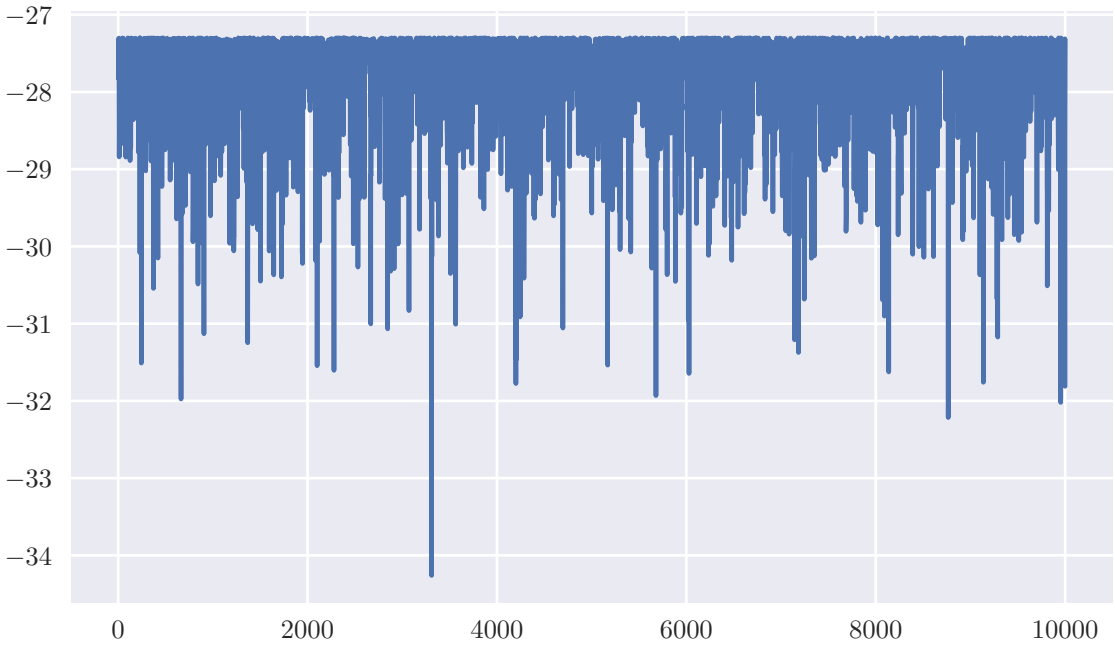


Ambos histogramas parecen acercarse al resultado deseado. Las gráficas para el logaritmo de la densidad evaluadas en X_t (para $0 \leq t \leq 10000$) son las siguientes

Gráfica de $\log f(X_t)$ para propuesta $U(0,1)$, $n = 5, r = 3$



Gráfica de $\log f(X_t)$ para propuesta $U(0,1)$, $n = 40, r = 15$

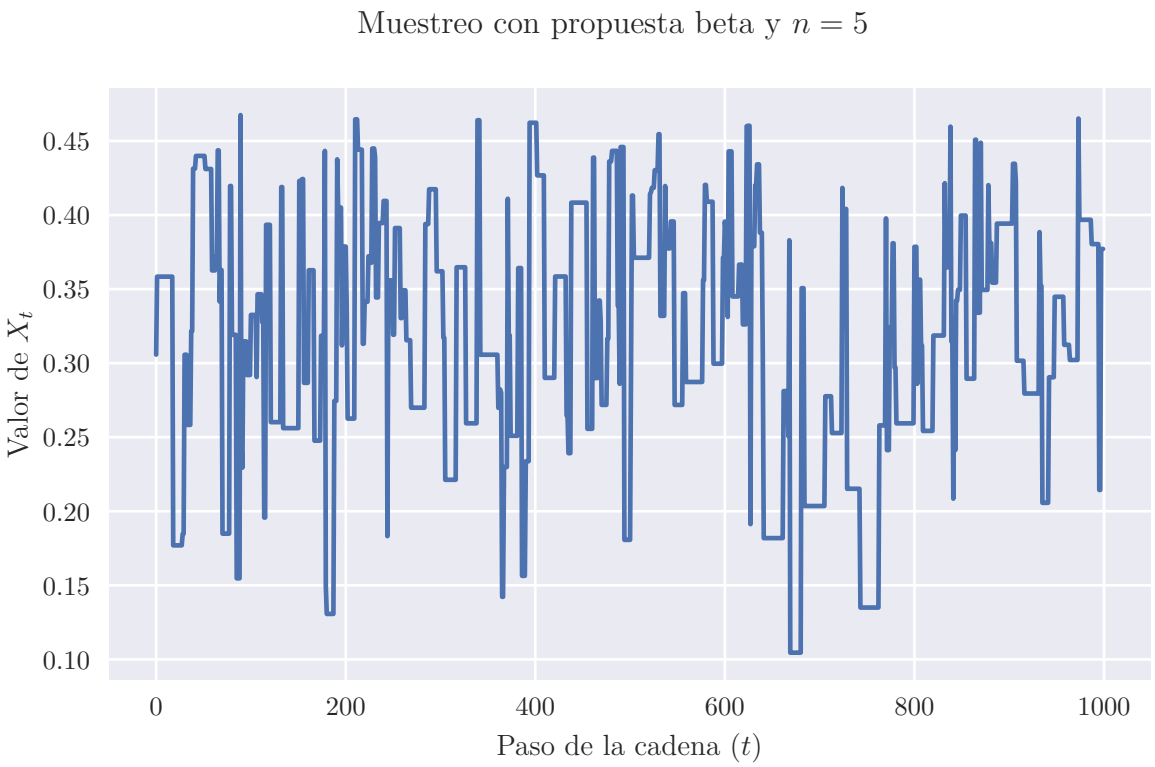


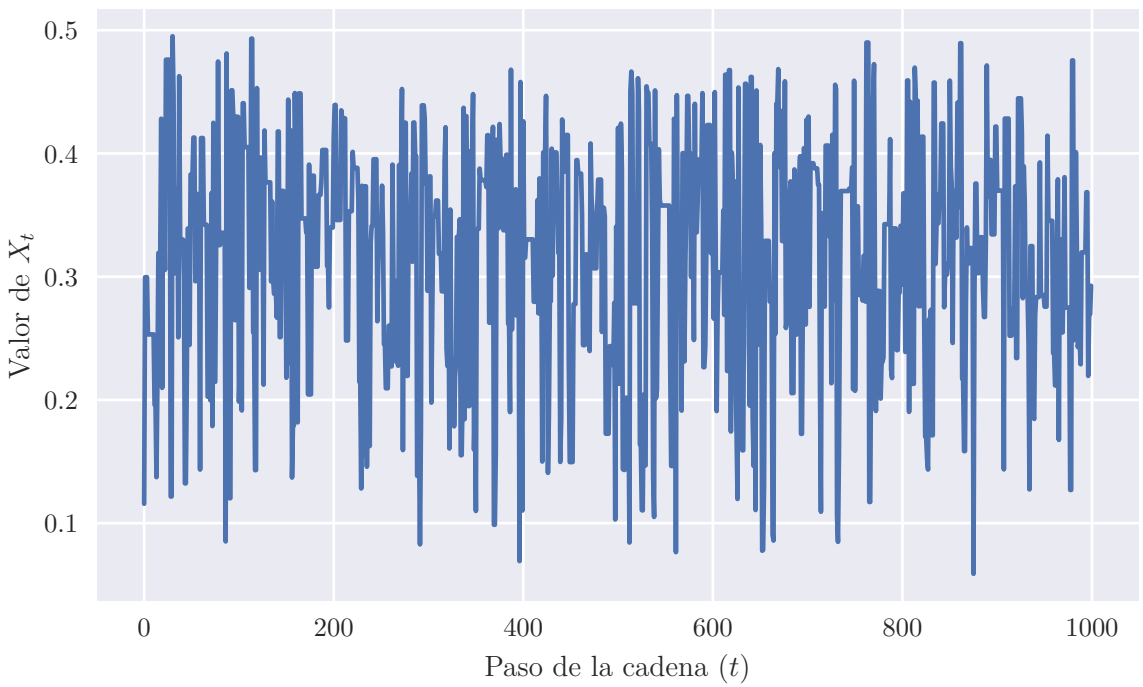
En este caso tenemos que la primera gráfica tiene a estabilizarse relativamente pronto, por lo que se pueden descartar los primeros valores como *burn-in*. La segunda gráfica presenta un comportamiento más irregular que las anteriores, esto puede deberse a que efectivamente se muestrea desde la densidad objetivo desde el comienzo, o a que la propuesta dada no permite una convergencia rápida. Decidimos descartar los primeros 20 valores de la cadena con $n = 5, r = 3$ y conservar todos en el caso de la cadena con $n = 40, r = 15$.

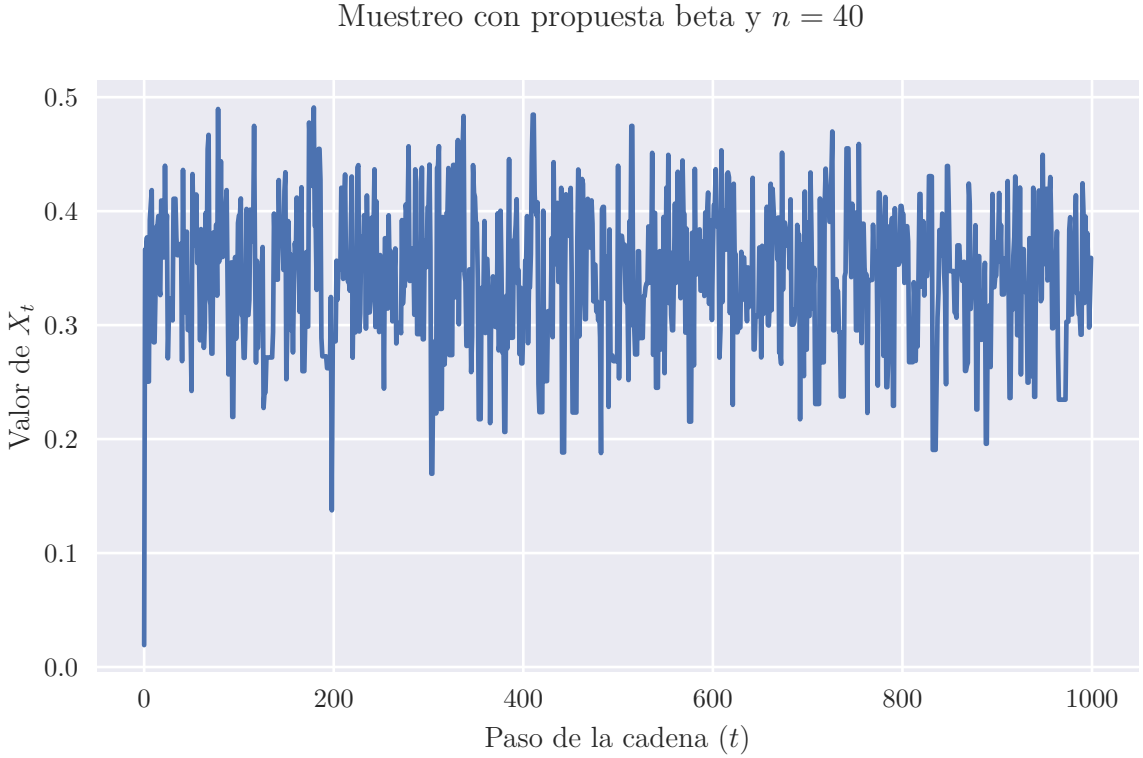
Una pregunta interesante es cuál de las dos funciones de transición propuestas nos da la mejor velocidad de conver-

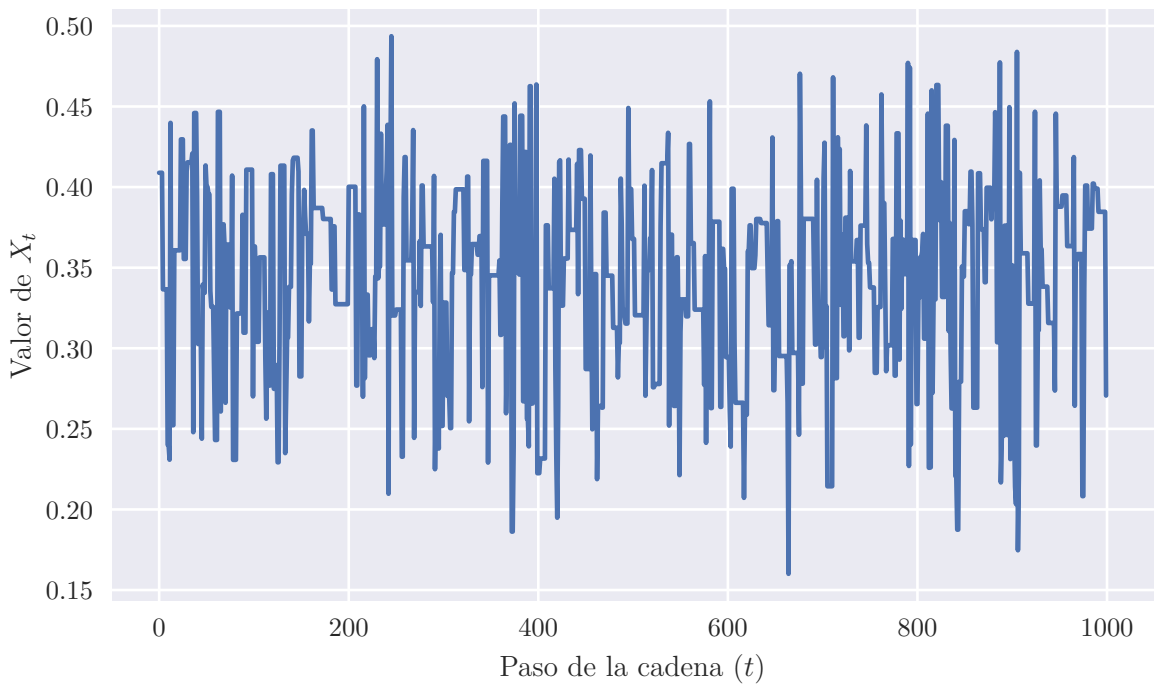
gencia. Un indicador de esta velocidad es que tan frecuentemente se rechaza la transición; si la transición se acepta con más frecuencia, entonces las propuestas son más parecidas a la distribución objetivo, y por lo tanto se muestrea más rápido.

En las siguiente gráficas podemos comparar el comportamiento de las cadenas con transición beta y uniforme para r_5 y r_{40} (Incluyendo sólo los primeros puntos para observar los intervalos de constancia). Notemos que a pesar de ser la misma cantidad de puntos en cada caso, la cadena con distribución beta y $n = 5, r = 3$ rechaza la transición con más frecuencia (tiene más intervalos de constancia), lo que puede ser causado porque el soporte de la densidad beta incluye puntos que no se encuentran en el intervalo $(0, 1/2)$. Lo anterior podría significar que la cadena con función de transición beta realiza un muestreo más lento en este caso. En el caso con $n = 40$ y $r = 13$, la cadena con densidad de transición uniforme parece rechazar con más frecuencia.




Muestreo con propuesta $U(01)$ y $n = 5$
Valor de X_t
0.5
0.4
0.3
0.2
0.1
0
200
400
600
800
1000
Paso de la cadena (t)


Muestreo con propuesta beta y $n = 40$
Valor de X_t
0.5
0.4
0.3
0.2
0.1
0.0
0
200
400
600
800
1000
Paso de la cadena (t)



Tras observar los ejemplos anteriores podemos concluir que, a pesar de que la implementación del algoritmo de Metropolis-Hastings es conceptualmente simple, obtener un algoritmo que pueda muestrear de una distribución objetivo de manera eficiente requiere varias decisiones de diseño no triviales y que afectan notablemente el desempeño práctico del algoritmo.