César Isaí García Cornejo
cesar.cornejo@cimat.mx
Computo Científico

Tarea 5

Semestre 3

CIMAT

Ejercicio 1
Definir la cdf inversa generalizada F_X^{-1} y demostrar que en el caso de variables aleatorias continuas esta coincide con la inversa usual. Demostrar además que en general para simular de X podemos simular $u \sim U(0,1)$ y $F_X^-(u)$ se distribuye como X.

Solución:

Definimos la inversa generalizada de la variable aleatoria X como la función

$$F_X^-(t) = inf\{x \in \mathbb{R} : F_X(x) \geq t\}.$$

Para variables aleatorias continuas sabemos que $F_X(x)$ es una función monótona creciente en el soporte de X. Es decir, tomemos $x, y \in supp(X)$, entonces $x < y$ si y solo si

$$F_X(x) < F_X(y)$$

donde $F(x)$ es una función biyectiva. Luego, existe su función inversa denotada por $F_X^{-1}(t)$. Entonces, usando la invertivilidad de la función de distribución en la definición de inversa generalizada

$$\begin{aligned} F_X^-(t) &= inf\{x \in \mathbb{R} : F_X(x) \geq t\}, \\ &= inf\{x \in \mathbb{R} : x \geq F_X^{-1}(t)\} \end{aligned}$$

como F_X es continua existe x_0 tal que $F_X(x_0) = t$, es decir $x_0 = F_X^{-1}(t)$. Luego, por la monotonía se sigue que el ínfimo se alcanza y

$$F_X^-(t) = F_X^{-1}(t)$$

que nos muestra que para variables X continuas su inversa coincide con la inversa generalizada.

Para la segunda demostración, primeramente probaremos lo siguiente. Sea X una variable aleatoria continua y F_X su función de distribución, entonces la variable aleatoria $Y = F_X(X)$ tiene distribución uniforme en el intervalo (0,1).

Para probarlo, es inmediato

$$\mathbb{P}(Y \leq y) = \mathbb{P}(F_X(X) \leq y) = \begin{cases} 0 & y < 0 \ \ o \ \ y = 0 \ y \ F_X(x) > 0 \quad \forall x \\ 1 & y \geq 1 \end{cases}$$

Por otra parte, si $y \in (0,1)$ o $y = 0$ y $F_X(x) = 0$ para alguna x, definamos:

$$x_0 = \sup\{x \in \mathbb{R} : F_X(x) = y\}$$

Como F_X es continua, se tiene $F_X(x_0) = y$. Además $F_X(x) > y$ para cualquier $x > x_0$ y $F_X(x) \leq y$ para cualquier $x < x_0$. Es decir, $F_X(x) \leq y$ si y sólo si $x \leq x_0$. Por tanto

$$\mathbb{P}(F_X(X) \leq y) = \mathbb{P}(X \leq x_0) = F_X(x_0) = y.$$

Así que

$$\mathbb{P}(Y \leq y) = \mathbb{P}(F_X(X) \leq y) = \begin{cases} 0 & y < 0 \\ y & y \in (0,1) \\ 1 & y \geq 1 \end{cases}$$

que es la distribución de la variable aleatoria uniforme en el intervalo (0,1). Para mostrar el segundo enunciado del Ejercicio 1, usaremos el resultado previo definiendo $X = F_X^{-1}(Y)$. Usando la invertivilidad $F_X(X) = Y$ en el cálculo con $Y \sim$Unif(0,1), es directo

$$F_X(q) = \mathbb{P}(X \leq q) = \mathbb{P}(F(X) \leq F(q)) = \mathbb{P}(Y \leq F(q)) = F(q)$$

para $q \in (0,1)$. Entonces X tiene como función de distribución F_X, que es lo que se quería probar.

Ejercicio 2
Implementar el siguiente algoritmo para simular variables aleatoria uniformes:

$$x_i = 107374182x_{i-1} + 104420x_{i-5} \mod 2^{31} - 1$$

regresa x_i y recorrer el estado, esto es $x_{j-1} = x_j; j = 1, 2, 3, 4, 5$; ¿ Parecen $U(0, 1)$?

Solución:

Para la implementación se creo una función llamada uniforme que toma dos argumentos, n es la cantidad de valores que toma de la recursión de congruencia. El segundo argumento es la semilla, si no se ingresa nada, la semilla predeterminada será dada por el tiempo del computador que nos permite acceder la librería time. Con la semilla determinada, se da un vector de valores arbitrarios para ser el vector inicial.

Posterior a los cinco valores iniciales, tomamos un vector X_t de tamaño n que nos da la recursión dada pero reescalamos cada valor entre el valor máximo, que es $2^{31} - 2$, para tener valores acotados entre cero y uno. Consideremos $n = 100$ y tomemos un histograma

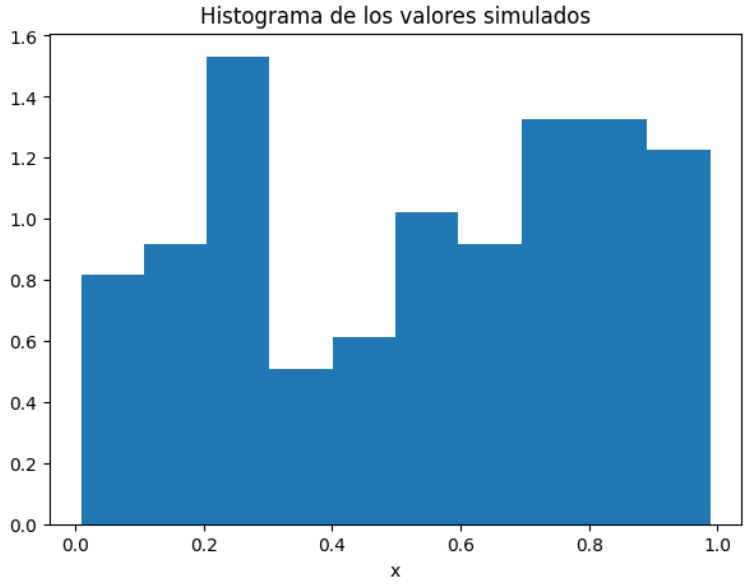

Figura 1: Histograma del generador de congruencia lineal $n = 100$.

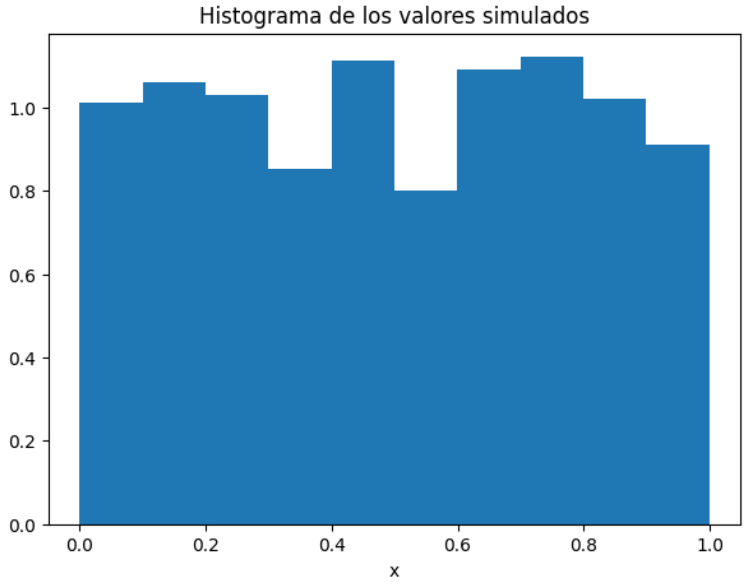

Figura 2: Histograma del generador de congruencia lineal $n = 1000$.

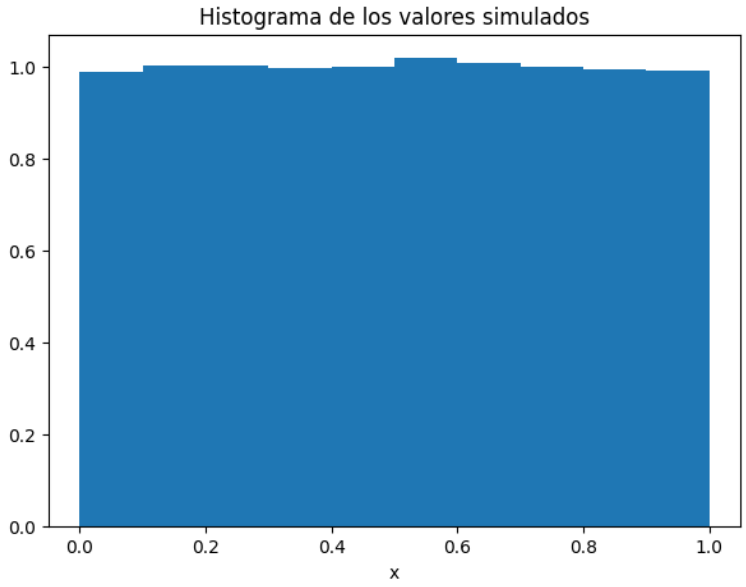

Figura 3: Histograma del generador de congruencia lineal $n = 100000$.

Podemos ver que parece converger a una distribución uniforme en (0,1). En el código se hace la comparación de este generador con el de scipy. Ambos histogramas tienen comportamientos similares. No podemos afirmar que distribuyen uniforme, sin embargo no tenemos evidencia de que no lo sea. Para poder concluir se requiere de una prueba de ajuste de bondad, sin embargo no es objetivo del ejercicio. Entonces, por lo visto en los histogramas podemos pensar que estamos simulando variables aleatorias uniformes.

Ejercicio 3
¿ Cuál es el algoritmo que usa *scipy.stats.uniform* para generar números aleatorios? ¿Cómo se pone la semilla? ¿y en R?

Solución:
El comando scipy.stats.uniform contiene todos los métodos requeridos para generar y trabajar con una distribución uniforme. La función que genera números aleatorios en un intervalo es "scipy.stats.uniform.rvs" y según la documentación utiliza "random_state", si es igual a nulo y usa de semilla "numpy.random.RandomState"; en cambio, si es un entero, ese se usa de semilla esto.

La clase "numpy.random.RandomState" , según la página web de numpy, es un contenedor para generar números pseudoaleatorios utilizando Mersenne Twister. Este realiza cambios a nivel de bits para generar nuevos números aleatorios como se puede ver en la siguiente cita.

Concretamente, se pone la semilla en el comando dentre del método como el siguiente ejemplo donde se simula una variable uniforme en (0,1) como se muestra a continuación

```
uniform.rvs(0,1,random_state = 12)
```

Para el caso de R, usa por default Mersenne Twister. Además, utiliza Wichmann-Hill, Marsaglia-Multicarry, Super-Duper, Knuth-TA0CP-2002, entre otros. Refierase en la siguiente cita.

Ejercicio 4
¿ En *scipy* que funciones hay para simular un variable aleatoria genérica discreta? ¿ tienen preprocesos?

Solución:
La clase "rv_discrete" documentada aquí se utiliza para construir distribuciones para variables aleatorias discretas, y se puede utilizar para cualquier distribución. Utiliza la función numpy.random.Generator que por default tiene el método BitGenerator (proporcionan bits aleatorios basados en diferentes algoritmos véase aquí). Para crear cualquier variable discreta, se debe de pasar un método de inicialización "rv_discrete" a través de values = keyword, que es una tupla de secuencias que describen los valores de X que ocurren con una probabilidad distinta de cero cita.

Ejercicio 5
Implementar el algoritmo Adaptative Rejection Sampling y simular de una $Gama(2,1)$ 10,000 muestras. ¿ cuando es conveniente dejar de adaptar la envolvente? (vea alg. A.7, p 54 Robert y Casella, 2da ed.)

Solución:
Se nos pide usar el método Adaptive Rejection Sampling para simular de una Gamma(2,1) con función de densidad

$$f(x) = xe^{-x}$$

Cómo el cuantil del 95 % es menor que 5, entonces la distribución cae rápidamente a cero para valores posteriores. Consideramos la aproximación del techo para un dominio de [0.01, 8] donde cada extremos son los valores mínimo y máximo de la rejilla inicial

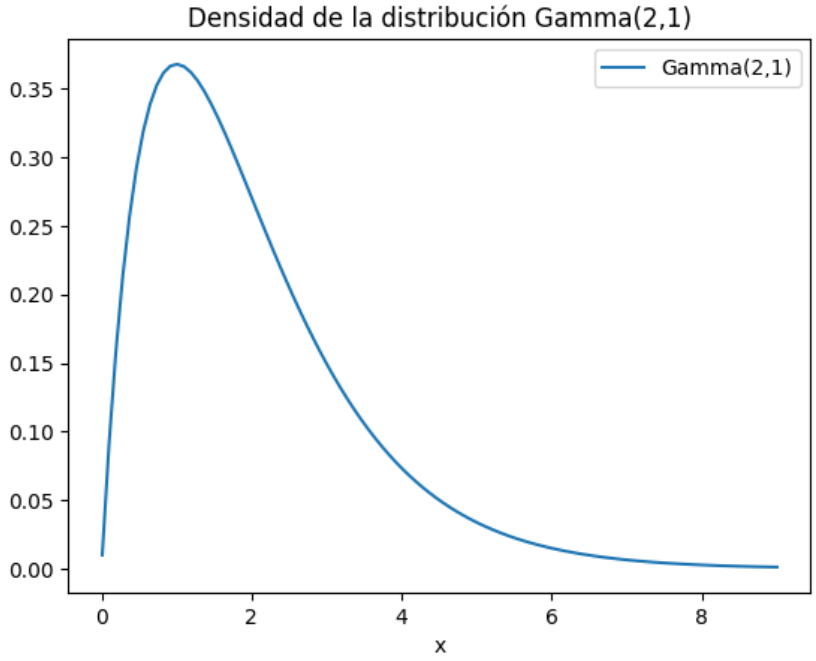

Figura 4: Densidad de la distribución gamma(2,1) para simulación

Calculamos la log-densidad. Como vemos que es cóncava, entonces la distribución Gamma(2,1) pertenece a la familia de las distribuciones log-cóncavas. Entonces, se cumple una de las hipótesis del método ARS. Empecemos por establecer una rejilla inicial de 5 valores. En esta rejilla es necesario obtener una función h(x) tal que

$$\log f(x) \leq h(x)$$

para todo $x \in supp(f(x))$. Tal *envolvente* dada por el método ARS es tal que, para cada valor de la rejilla, se obtienen en términos de los segmentos de las rectas creadas por puntos adyacentes.

Por dar un ejemplo, consideremos la rejilla X = [0.4,1,3,8] entonces, evaluamos cada punto en la rejilla en la log-densidad. Obtenemos la siguiente figura

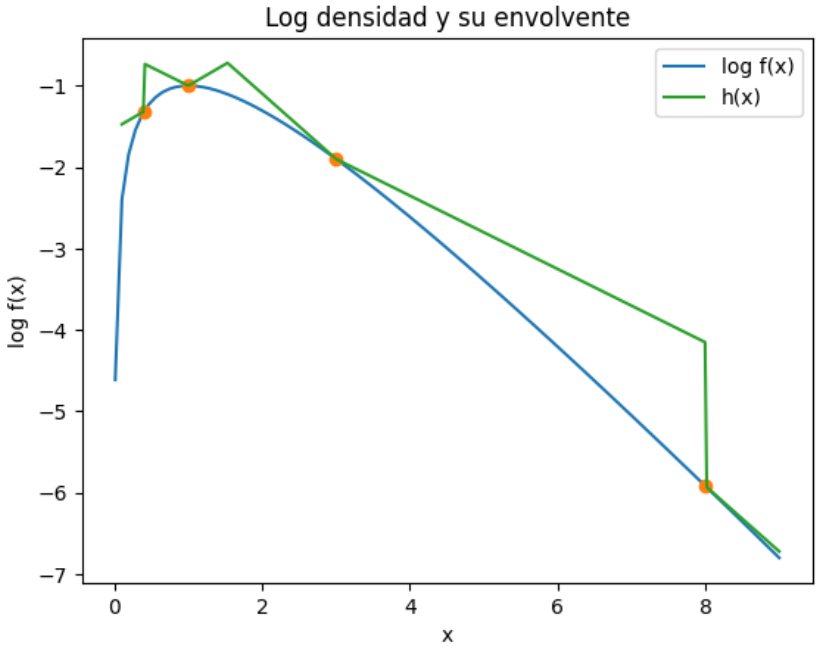

Figura 5: Log f(x) para la distribución Gamma(2,1) y la envolvente construida con la rejilla inicial dada.

Con la envolvente calculada, podemos regresar a la distribución Gamma, sin logaritmos, al igual que a la envolvente, donde vemos que dicha envolvente es una mezcla de exponenciales.

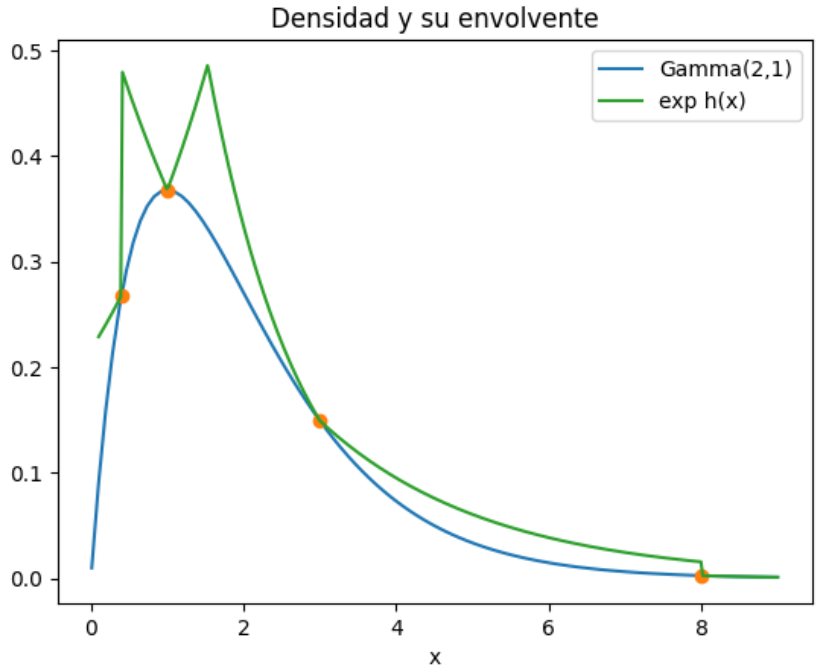

Figura 6: Función de densidad de la Gamma(2,1) y su envolvente

Posteriormente, es necesario simular de la función techo $g(x) = \exp(h(x))$. Para ello usamos el resultado del inciso 1. Es decir, generamos una función acumulada de probabilidad para $g(x)$ con esto tenemos que crear ahora la función para la inversa generalizada dada un $u \in [0,1]$. Simulando la variable aleatoria uniforme u y evaluando en la inversa generalizada $z = g^{-}(u)$ tenemos una simulación de la densidad techo.

Ahora usando el método de accept-reject. Tomamos la simulación z y una nueva simulación de la uniforme u y aceptamos como muestras de la función de densidad f solo para aquellas muestras z que satisfagan que $u * f(z) < g(z)$. En caso de no aceptar se rechaza la muestra y se actualiza con la función constructor() para una nueva rejilla donde se agrega el simulado en la rejilla. Es decir, se vuelve a hacer el mismo procedimiento para simular de la función envolvente g actualizada.

La actualización de la rejilla tras rechazar ciertas muestras nos actualiza la envolvente. En este caso, se decidió tomar solo una cantidad fija de actualizaciones ya que es computacionalmente pesado actualizar arbitrariamente. Para cuatro actualizaciones tenemos que la envolvente en la log-densidad es

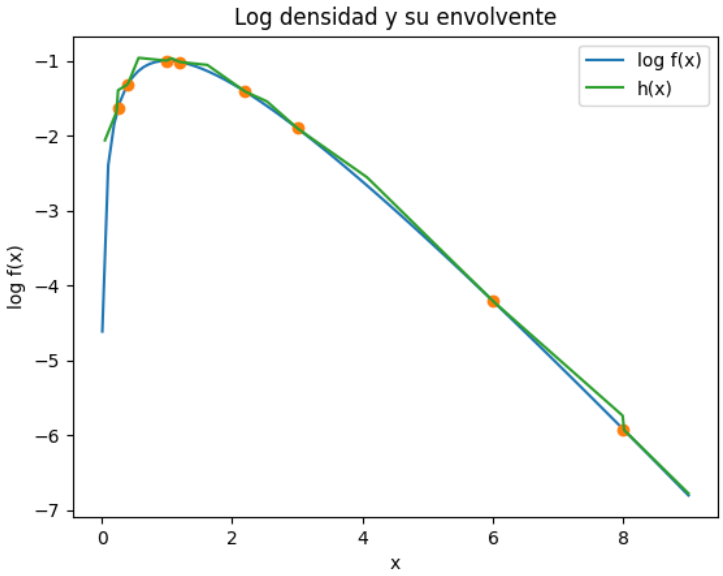

Figura 7: Envolvente modificada con más puntos en la rejilla

Ya con la envolvente modificada, se toman ahora las simulaciones de 100000 elementos y se grafican en un histograma junto a la densidad gamma(2,1).

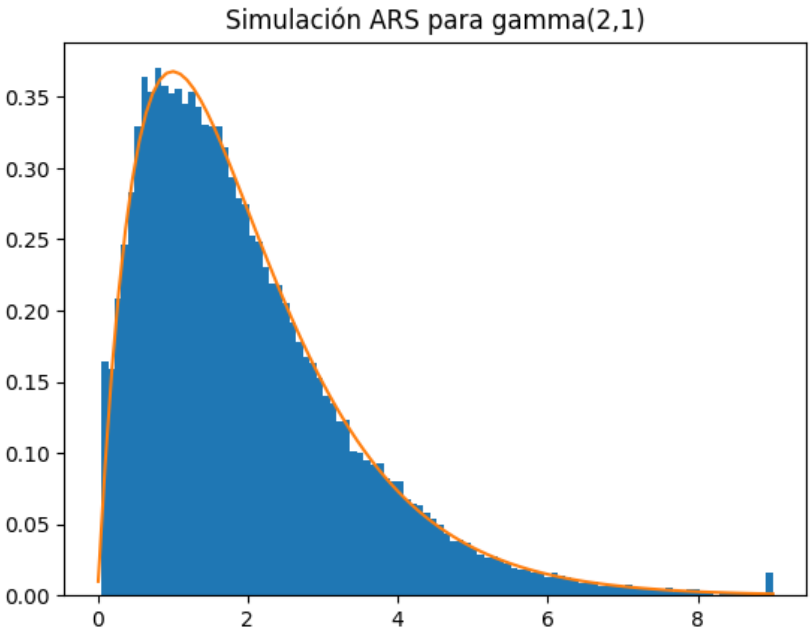

Figura 8: Histograma para las muestras simuladas por el método ARS de la distribución gamma(2,1)

Notemos que tenemos un buen ajuste asintótico para la densidad objetivo. Veamos que por construcción, en los extremos tenemos masa acumulada debido a que en la inversa generalizada, al tener mallas muy finas (linspace) para valores cercanos a 1 (x grandes), no se podia encontrar un valor en la malla que no fuera menor que x. Este problema se debe a que la inversa generalizada cercana a uno puede crecer significativamente, por lo que aquellos valores X de la simulación que sobrepasaran el límite establecido se agrupaban en el último valor del linspace.

Referencias

[1] Robert, C. P., Casella, G., and Casella, G. (1999). Monte Carlo statistical methods (Vol. 2). New York: Springer.