Certamen Simulación Estocástica

MAT468

Alejandro Villazón G.

Octubre 2023

Todos los códigos elaborados para responder este certamen se realizaron con el lenguaje de programación Python, los códigos fueron elaborados de forma parámetrica, es decir, entrega flexibilidad para cambiar parámetros y analizar resultados, por ejemplo, puede modificarse la familia de funciones usada en cierta pregunta y los códigos funcionarán.

Para una mayor comodidad, los archivos de códigos se pueden encontrar en mi repositorio personal de GitHub, AlejandroVillazonG/MAT468, de todas formas estos serán adjuntados al certamen.

Problema 1

En este reporte, se plantea la simulación de muestras a partir de la distribución Wishart $\mathrm{W}_p(k, V)$. En base a conocimientos previos adquiridos en el curso Análisis Estadístico Multivariado (MAT269) sabemos que la simulación es directa y efectiva cuando el parámetro k es un número entero. Sin embargo, surge un desafío interesante cuando k es un valor positivo no entero.
En este reporte, se abordará esta problemática y se propondrá un método de simulación para la generación de muestras de la distribución Wishart en un amplio rango de situaciones. Se comprobará la efectividad del método propuesto validando resultados teóricos haciendo uso de las simulaciones realizadas. Para un caso práctico, se utilizarán las simulaciones con el propósito de generar matrices simétricas y definidas positivas, asegurando que sus valores propios se encuentren dentro del intervalo $[0, \lambda_{\max}]$, donde $\lambda_{\max}$ es un parámetro ajustable.

Método

Se propone como método de simulación el conocido algoritmo *Aceptación-Rechazo*, con densidad objetivo $f(\cdot \mid k, \mathbf{V})$ y densidad instrumental $f(\cdot \mid \lfloor k \rfloor, \mathbf{V})$. El valor de M se obtendrá para cada combinación de parámetros $k, \mathbf{V}$ mediante una búsqueda adaptativa del valor hasta obtener un ratio de aceptación menor a 1 para una cantidad considerable de simulaciones.
Para la generación de simulaciones desde la densidad instrumental haremos uso del siguiente resultado de MAT269:

Resultado. Suponga que $\mathbf{z_1}, \mathbf{z_2}, \ldots, \mathbf{z_k}$ son vectores IID desde $\mathrm{N}_p(\mathbf{0}, \mathbf{V})$. Entonces

$$\mathbf{X} = \sum_{i=1}^{k} \mathbf{z_i} \mathbf{z_i}^\top \sim \mathrm{W}_p(k, \mathbf{V}).$$

Además, sabemos que si $\mathbf{z_0} \sim \mathrm{N}_p(\mathbf{0}, \mathbf{I}_p)$, entonces $\mathbf{L}\mathbf{z_0} \sim \mathrm{N}_p(\mathbf{0}, \mathbf{V})$, donde $\mathbf{L}$ es la descomposición de Cholesky de $\mathbf{V}$, es decir, $\mathbf{V} = \mathbf{L}\mathbf{L}^\top$. Por lo tanto, para k entero simularemos desde $\mathrm{W}_p(k, \mathbf{V})$ mediante el siguiente esquema optimizado:

1. Generar $\mathbf{z_1}, \mathbf{z_2}, \ldots, \mathbf{z_k}$ desde $\mathrm{N}_p(\mathbf{0}, \mathbf{I}_p)$.
2. Calcular $\mathbf{L}$ tal que $\mathbf{V} = \mathbf{L}\mathbf{L}^\top$.
3. Retornar $\mathbf{X} = \mathbf{L} \left(\sum_{i=1}^{k} \mathbf{z_i} \mathbf{z_i}^\top \right) \mathbf{L}^\top$.

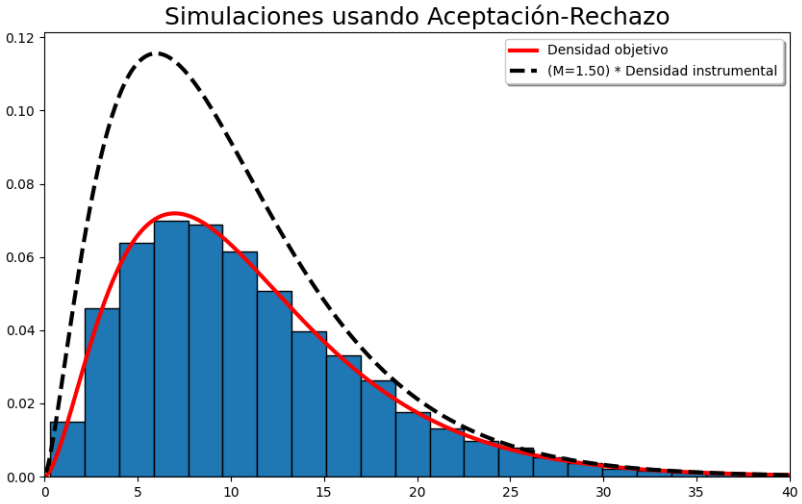
Le llamamos *optimizado*, debido al hecho que el cálculo de la descomposición de Cholesky de una matriz se vuelve costoso a medida que la dimensión crece, usando el esquema anterior este cálculo se realiza solo una vez.
Por otro lado, dada la dificultad de calcular el valor de M óptimo en el caso k no entero, que corresponde a $\sup_{\mathbf{X}\in\mathcal{M}_p(\mathbb{R})} \frac{f(\mathbf{X} \mid k,\mathbf{V})}{f(\mathbf{X} \mid \lfloor k \rfloor,\mathbf{V})}$, proponemos el siguiente algoritmo de búsqueda:

1. Iniciar $M = 1$, $paso > 0$ y $N \in \mathbb{N}$ la cantidad de simulaciones para aceptar M.
2. Generar N simulaciones $\mathbf{X} \sim \mathrm{W}_p(\lfloor k \rfloor, \mathbf{V})$ y comprobar que $\frac{f(\mathbf{X} \mid k,\mathbf{V})}{Mf(\mathbf{X} \mid \lfloor k \rfloor,\mathbf{V})} \leq 1$.
3. En caso de que se cumpla la desigualdad para las N simulaciones se acepta M, en otro caso se le suma $paso$ a M, i.e., $M = M + paso$ y se vuelve al paso anterior.

Una vez obtenido el valor de M experimental, podemos hacer uso del algoritmo *Aceptación-Rechazo* sin ninguna modificación.

Ejemplo básico del método

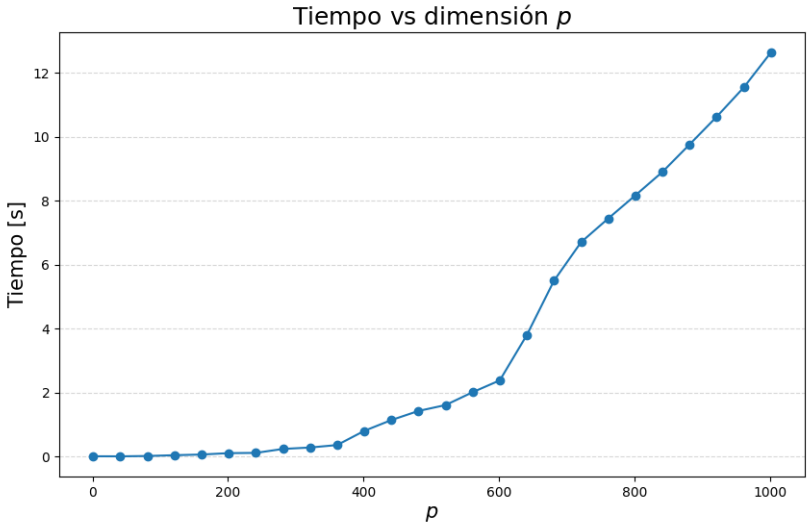
Para ejemplificar nuestro método de simulación de forma visual veamos un ejemplo con dimensión $p = 1$, $k = 5{,}5$ y $\mathbf{V} = 2$. Para el cálculo de M utilizamos $paso = 0{,}1$ y $N = 1000$, es decir, aumentamos M hasta verificar que 1000 ratios fueran menor a 1, obteniendo $M = 1{,}5$. Realizamos 10.000 simulaciones utilizando el método escogido, obteniendo los siguientes resultados comparativos.



Se puede observar que la distribución normalizada de las simulaciones se aproxima de buena forma a la densidad objetivo. Además, notamos que el valor experimental de M logra dejar la densidad instrumental escalada sobre la densidad objetivo.

Costo computacional

Un buen método de simulación debe ser eficiente y funcionar bien incluso cuando se enfrenta a situaciones más complejas o con dimensiones más grandes. En este sentido, evaluaremos cuánto tiempo lleva ejecutar el método a medida que aumentamos la dimensión p del espacio de matrices. Este análisis nos ayudará a determinar si el método es adecuado para su uso en situaciones más desafiantes. Con el objetivo de analizar el parámetro p versus el tiempo de computo, se mantendrá constante el valor de k en 1002 y se establecerá la matriz $\mathbf{V}$ como la matriz identidad de tamaño p. Para respetar la condición del modelo $k > p - 1$, se variará el valor de p dentro del rango de 1 a 1002, con incrementos de 40 unidades. Este enfoque permitirá explorar una amplia gama de dimensiones y evaluar cómo afecta el parámetro p en los tiempos de cómputo. A continuación, se presentan los resultados obtenido:



En el gráfico observamos que, a medida que aumenta el valor de p, se produce un incremento en el tiempo de realización de una simulación. A pesar de que este aumento no es excesivamente pronunciado, como lo demuestra el hecho que para una dimensión de $p = 1000$, el tiempo de cómputo es de al menos 10 segundos, debemos considerar que este tiempo representa lo que demora el algoritmo en realizar una sola simulación. En consecuencia, si extrapolamos estos tiempos para simular 10 matrices de tamaño $p = 1000$, podríamos incurrir en un tiempo que supera el minuto y medio, lo cual ya constituye un intervalo de tiempo considerable. Sin embargo, si la dimensión es menor a 200, observamos que el tiempo necesario para realizar una simulación es cercano a 0. Esto implica que nuestro método se presenta como una herramienta útil y eficiente para generar un gran volumen de simulaciones si las dimensiones son pequeñas, logrando el objetivo en un tiempo razonable, como se podrá observar en lo que sigue de reporte.

Evaluación del método

Hasta este punto, hemos comprobado la eficacia de nuestro método en el caso $p = 1$. Ahora, examinemos su rendimiento en dimensiones mayores. La efectividad del método será validada mediante la comparación de resultados obtenidos con simulaciones frente a los resultados teóricos esperados.

Simulaciones realizadas

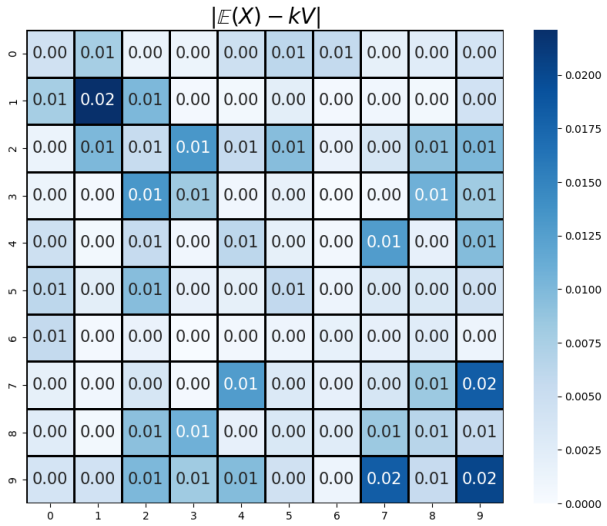
En el contexto de llevar a cabo una validación reproducible por el lector, se ha fijado la dimensión $p = 10$ y los parámetros $k, \mathbf{V}$ se han establecido de la siguiente manera: $k = p + \varepsilon_0$ y $\mathbf{V} = \text{diag}(\varepsilon_1, \ldots, \varepsilon_p)$, donde $\varepsilon_i \sim \text{U}(0, 1)$ para todos $i = 0, 1, \ldots, p$. Es importante destacar que, aunque esta configuración específica se ha utilizado para fines de presentación, el método es paramétrico y se puede adaptar a cualquier combinación coherente de valores k y $\mathbf{V}$.
En la línea de los parámetros escogidos, la búsqueda experimental de M tardó alrededor de 12 segundos, obteniendo $M = 2{,}5$, por lo que, se espera una tasa de aceptación del método igual a $1/M = 40\,\%$. Realizamos 50.000 simulaciones tardando casi 6 minutos con un ratio de aceptación igual a $39{,}86\,\%$, levemente menor al esperado.

Validación de resultados teóricos

Para justificar la efectividad del método de simulación haremos uso de distintos resultados, el primero de ellos es:

> **Resultado.** Si $\mathbf{X} \sim \text{W}_p(k, \mathbf{V})$, entonces $\mathbb{E}[\mathbf{X}] = k\mathbf{V}$.

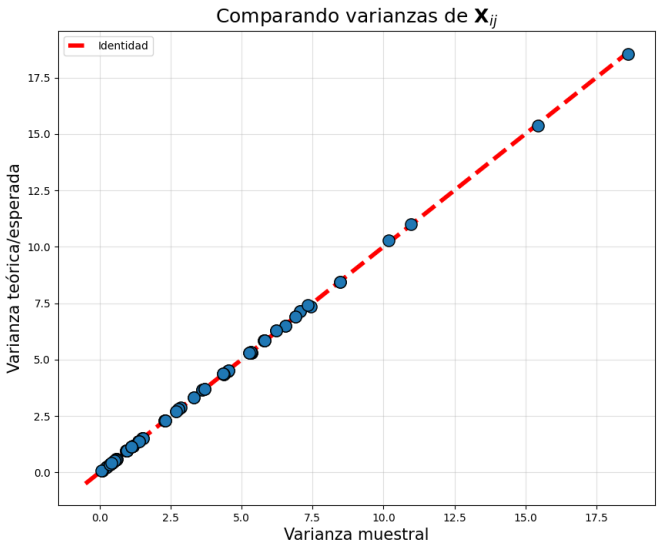
Para esto calculamos el promedio de las matrices simuladas, esto corresponde al promedio en cada coordenada, la cual comparamos con la esperanza esperada $k\mathbf{V}$. Para visualizar los resultados de la comparación, presentamos un mapa de calor de la matriz de error absoluto, es decir la diferencia en valor absoluto de las coordenadas.



Se puede observar que los valores son cercanos a 0, obteniendo como máximo una diferencia absoluta de 0,02, lo cual es pequeño en comparación a la magnitud de los valores de la matriz, ya que el valor máximo de la matriz $k\mathbf{V}$ es cercano a 10. De esta forma, los resultados obtenidos validan la eficacia del método.
Por otro lado, recordemos el siguiente resultado:

Resultado. Si $\mathbf{X} \sim \mathrm{W}_p(k, \mathbf{V})$, entonces $\mathrm{var}(\mathbf{X}_{ij}) = k\left(\mathbf{V}_{ij}^2 + \mathbf{V}_{ii}\mathbf{V}_{jj}\right)$ para todo $i, j = 1, \ldots, p$.

Para realizar la comparación con las simulaciones, calculamos la matriz de varianzas teóricas dadas por el resultado anterior y la matriz de varianzas de las simulaciones, esto es la varianza de cada coordenada. Comparamos los valores en un gráfico de dispersión, como se puede observar a continuación:

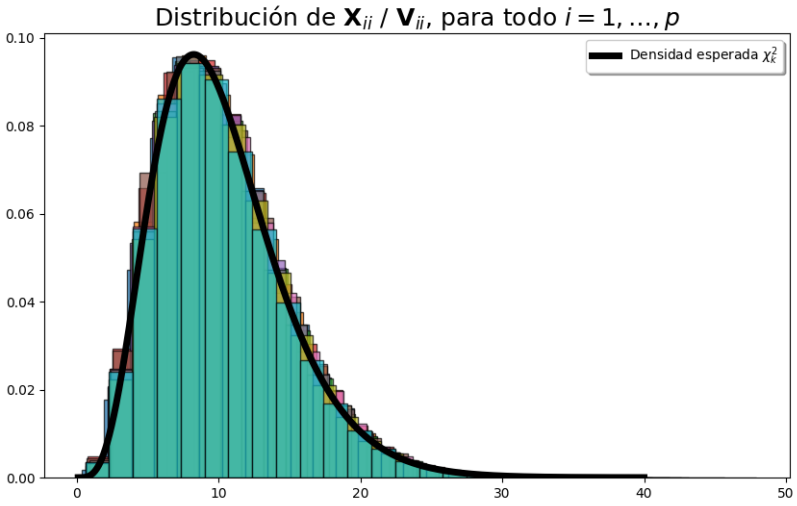


Se puede observar que el par de varianzas para cada coordenada de las matrices simuladas $\mathbf{X}$ están cercanos a la identidad, esto quiere decir que la varianza muestral se aproxima bien a la esperada. Incluso visualmente se puede observar que los puntos están sobre la identidad, utilizando la métrica del error cuadrático medio obtenemos un error de: 0,00087, el cual es notablemente bajo, justificando así la eficacia del método de simulación.

Para finalizar la justificación de nuestro método, recordamos el siguiente resultado:

Resultado. Si $\mathbf{X} \sim \mathrm{W}_p(k, \mathbf{V})$, entonces $\mathbf{X}_{ii} \sim \mathbf{V}_{ii}\chi_k^2$ para todo $i = 1, \ldots, p$.

El resultado anterior es equivalente a que $\mathbf{X}_{ii}/\mathbf{V}_{ii} \sim \chi^2_k$ para todo $i = 1, \ldots, p$. Con las simulaciones obtenidas realizamos el histograma normalizado para cada variable de la diagonal $\mathbf{X}_{ii}$, escaladas por su factor $\mathbf{V}_{ii}$ correspondiente, comparamos estas distribuciones con la densidad esperada χ^2_k.



En el gráfico anterior, es evidente que la forma de la distribución de las variables escaladas se asemeja a la densidad teórica esperada para todas las variables de la diagonal. Aunque el ajuste no es perfecto, consideramos que la distribución efectivamente refleja la densidad esperada, respaldando así la validez de nuestro método.

Tras analizar y verificar los resultados previamente expuestos a través de las simulaciones generadas, queda demostrado que nuestro método es eficaz y logra la generación de simulaciones acorde a la densidad deseada.

Uso práctico

Un uso práctico de nuestro método de simulación es generar matrices simétricas con valores propios positivos y acotados por un valor $\lambda_{\max}$ previamente escogido. Para esto basta simular una matriz $\mathbf{A}$ desde nuestro método y escalarla por el factor $\lambda_{\max}/\lambda_{\text{máx}}(\mathbf{A})$, donde $\lambda_{\text{máx}}(\mathbf{A})$ corresponde al valor propio más grande de la matriz $\mathbf{A}$, es decir, $\lambda_{\text{máx}}(\mathbf{A}) = \text{máx}_{\lambda \in \sigma(\mathbf{A})}\, \lambda$. Demostremos este resultado de álgebra lineal.
Supongamos que $\mathbf{v}$ es un vector propio arbitrario de $\mathbf{A}$, es decir, existe $\lambda \neq 0$ tal que $\mathbf{Av} = \lambda\mathbf{v}$, si escalamos por el factor tenemos que

$$\left(\frac{\lambda_{\max}}{\lambda_{\text{máx}}(\mathbf{A})}\mathbf{A}\right)\mathbf{v} = \frac{\lambda_{\max}}{\lambda_{\text{máx}}(\mathbf{A})}\mathbf{Av} = \frac{\lambda_{\max}}{\lambda_{\text{máx}}(\mathbf{A})}\lambda\mathbf{v} = \lambda_{\max}\left(\frac{\lambda}{\lambda_{\text{máx}}(\mathbf{A})}\right)\mathbf{v}$$

Por lo tanto,

$$\sigma\left(\frac{\lambda_{\max}}{\lambda_{\text{máx}}(\mathbf{A})}\mathbf{A}\right) = \left\{\lambda_{\max}\left(\frac{\lambda}{\lambda_{\text{máx}}(\mathbf{A})}\right) \mid \lambda \in \sigma(\mathbf{A})\right\}$$

Dado que $\mathbf{A}$ es definida positiva y por definición de $\lambda_{\text{máx}}(\cdot)$ tenemos que $0 < \lambda \leq \lambda_{\text{máx}}(\mathbf{A})$ para todo $\lambda \in \sigma(\mathbf{A})$, entonces

$$\lambda_{\max}\left(\frac{\lambda}{\lambda_{\text{máx}}(\mathbf{A})}\right) \leq \lambda_{\max}$$

De donde concluimos que,

$$\sigma\left(\frac{\lambda_{\max}}{\lambda_{\text{máx}}(\mathbf{A})}\mathbf{A}\right) \subseteq [0, \lambda_{\max}].$$

Por lo tanto, hemos demostrado que para generar matrices simétricas con valores propios positivos acotados superiormente por un valor específico, basta con generar matrices utilizando nuestro método y luego escalarlas por un factor pertinente. Para implementar este objetivo, definimos la función `matrix_eigvals_bounded` que recibe la dimensión de la matriz y el valor máximo para los valores propios, dado que el parámetro k queda libre, haremos uso de la versión simplificada del método para valores de k entero. Por lo tanto, para generar la matriz inicial consideramos $k = p$ y $\mathbf{V} = \mathbf{I}_p$.
Para verificar que el método funciona, llevamos a cabo distintas simulaciones definiendo el valor máximo y la dimensión de la matriz, comprobamos que el método funciona calculando los valores propios de la matriz resultante. Por ejemplo, para $p = 10$, $\lambda_{\max} = 6$, obtenemos una matriz con los siguientes valores propios aproximados: 6, 5.02, 4.64, 2.58, 2.11, 1.12, 9.4e-1, 5e-1, 2.3e-4, 4.5e-2, verificando en este caso la cota superior a los valores propios.

Hardware utilizado y Reproducibilidad

Todos los resultados presentados en este reporte son reproducibles mediante los códigos disponibles en el archivo `P1.ipynb`. El hardware utilizado para obtener estos resultados se componen por los siguientes:

- Procesador: 11th Gen Intel(R) Core(TM) i5-1135G7 2.40GHz, 2419 Mhz, 4 procesadores principales, 8 procesadores lógicos.
- Memoria física instalada (RAM): 8.00 GB.
- Tarjeta gráfica: Intel(R) Iris(R) Xe Graphics.
- Sistema operativo: Windows 11.

Problema 2

a) Para todo α tenemos que π_α es la distribución uniforme discreta en el conjunto $\{1,2,3,4,5\}$. Notamos que hay $1/2$ de probabilidad de que el algoritmo se llame recursivamente a si mismo, por lo tanto, el algoritmo no terminará si se llama infinitamente a si mismo, esto nos indica que,

$$\mathbb{P}(\text{Algoritmo termine}) = 1 - \mathbb{P}(\text{Algoritmo no termine}) = 1 - \lim_{n\to\infty} (1/2)^n = 1$$

de donde concluimos que el algoritmo termina en tiempo finito con probabilidad 1.

Veamos que el algoritmo regresa una muestra desde π_α, sea $i \in \{1,2,3,4,5\}$, luego dado que el algoritmo se llama así mismo con el mismo parámetro, i.e. exactamente de la misma forma, tenemos que la probabilidad de que la salida final sea i es la probabilidad de que en la elección inicial sin recurrencia se obtenga i, más la probabilidad de que se llame recursivamente al algoritmo $(1/2)$ por la probabilidad de que en la recurrencia la salida sea i. Por lo tanto, tenemos que,

$$\mathbb{P}(X = i) = \frac{1}{10} + \frac{1}{2}\mathbb{P}(X = i) \qquad \Longrightarrow \qquad \mathbb{P}(X = i) = \frac{1}{5}$$

para todo $i \in \{1,2,3,4,5\}$. Luego, por definición tenemos que ***Algo1***(α) es un *algoritmo de simulación perfecta*.

b) Primero notamos que ***Algo2***(α) es un esquema recursivo probabilístico. Dado α un entero positivo, tenemos que si $\alpha \leq 5$ el algoritmo termina por definición en la primera iteración. Considere $\alpha > 5$, luego la probabilidad de que el algoritmo no termine en la primera iteración es $1 - 5/\alpha$ obteniendo como resultado α_1 tal que $5 < \alpha_1 \leq \alpha$. Supongamos que el algoritmo no termina, obteniendo α_2 tal que $5 < \alpha_2 \leq \alpha_1$, esto sucede con probabilidad $(1 - 5/\alpha)(1 - 5/\alpha_1) \leq (1 - 5/\alpha)^2$. Siguiendo la idea anterior, la probabilidad de que el algoritmo no termine en la iteración n es $(1 - 5/\alpha)\prod_{i=1}^{n-1}(1 - 5/\alpha_i) \leq (1 - 5/\alpha)^n$, donde $\alpha_i > 5$ es el retorno del algoritmo en el paso i-ésimo. Por lo tanto, como $\alpha > 5$

$$\mathbb{P}(\text{Algoritmo no termine}) = (1 - 5/\alpha) \lim_{n\to\infty} \prod_{i=1}^{n-1}(1 - 5/\alpha_i) \leq \lim_{n\to\infty}(1 - 5/\alpha)^n = 0$$

de donde concluimos que el algoritmo termina en tiempo finito con probabilidad 1 para todo α.

Probemos que el algoritmo es localmente correcto respecto a la distribución uniforme, para esto notamos que si $\alpha \leq 5$ el resultado es directo por definición del algoritmo. Supongamos que $\alpha > 5$. Sea $i \in \{1,2,3,4,5\}$, notamos que la probabilidad de que la salida final del algoritmo sea igual a i es la probabilidad de que en la elección inicial se obtenga $X = i$ más la probabilidad de que el algoritmo sea llamado nuevamente $(X > 5)$ por la probabilidad de que el resultado de la recursión sea igual a i, es decir,

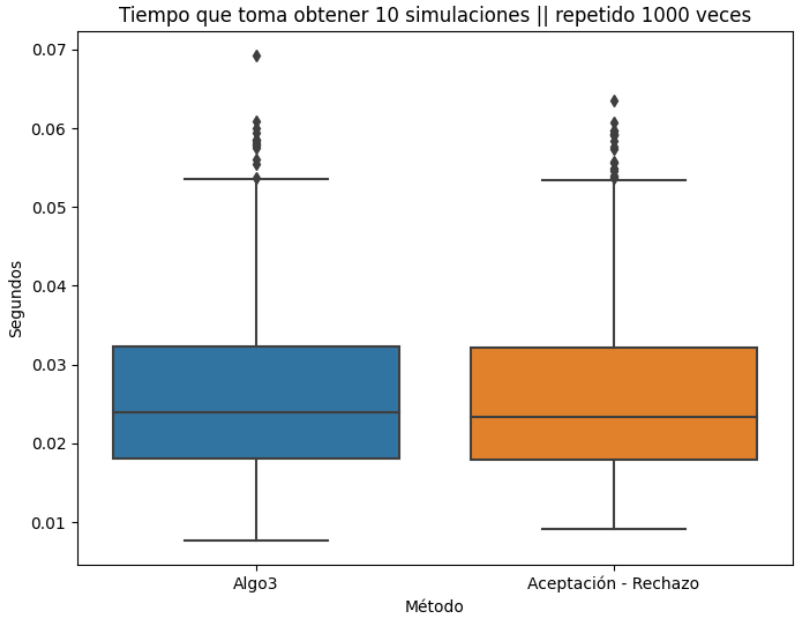
$$\mathbb{P}(X = i) = \frac{1}{\alpha} + \left(1 - \frac{5}{\alpha}\right)\mathbb{P}(X = i) \qquad \Longrightarrow \qquad \mathbb{P}(X = i) = \frac{1}{5}$$

Por lo tanto, el algoritmo es *localmente correcto* con respecto a la distribución uniforme, luego, por el Teorema fundamental de la simulación perfecta concluimos que ***Algo2***(α) es un *algoritmo de simulación perfecta*.

c) Para motivar la comparación de los algoritmos implementaremos el *Example 5.6.7* del libro *Statistical Inference, Casella, Berger*. El objetivo es simular desde $Y \sim \text{Beta}(a, b)$ con a, b no enteros, en particular, usaremos $a = 2{,}7$, $b = 6{,}3$. Este es un ejemplo que introduce el algoritmo Aceptación-Rechazo en el libro, por lo que, usaremos como densidad instrumental la distribución uniforme estándar, con $M = \text{máx}_y\, f_Y(y) = 2{,}669$.

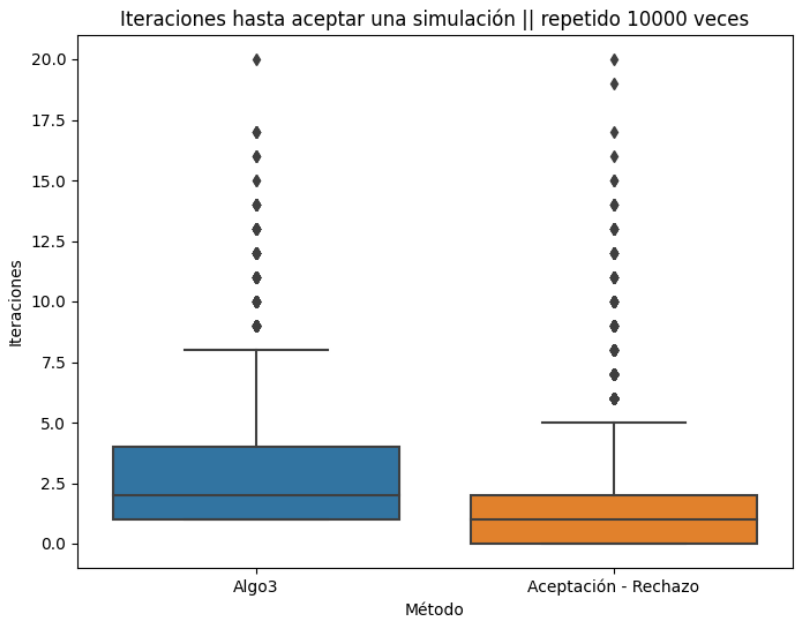
Para llevar a cabo la comparación entre los algoritmos, se programó cada uno de estos en el lenguaje `Python`, los detalles de código se pueden encontrar en el archivo `P2_c.ipynb`, donde también se encuentran los códigos para generar los resultados que se presentan a continuación.

Para una justa comparación se utilizaron las mismas funciones en cada algoritmo. Una forma de comparar el desempeño de los algoritmos es a través de los tiempos de ejecución, para esto se registró el tiempo de ejecución 10.000 veces para el proceso de obtener 10 simulaciones desde cada algoritmo. Estos tiempos de ejecución se midieron en segundos, obteniendo los siguientes resultados:



A simple vista, los resultados parecen muy similares, ya que el 50 % de los tiempos de ejecución se sitúa aproximadamente entre 0.02 y 0.03 segundos para ambos algoritmos. Ambos algoritmos muestran valores atípicos en ocasiones, llegando incluso a tomar 0.07 segundos para realizar 10 simulaciones en el caso de *Algo3*. Sin embargo, al analizar en detalle, encontramos que *Algo3*, en promedio, tarda 0.02624 segundos en completar las 10 simulaciones, mientras que el algoritmo *Aceptación-Rechazo* toma 0.02578 segundos. Este resultado no sorprende, ya que concuerda con la naturaleza recurrente de Algo3.

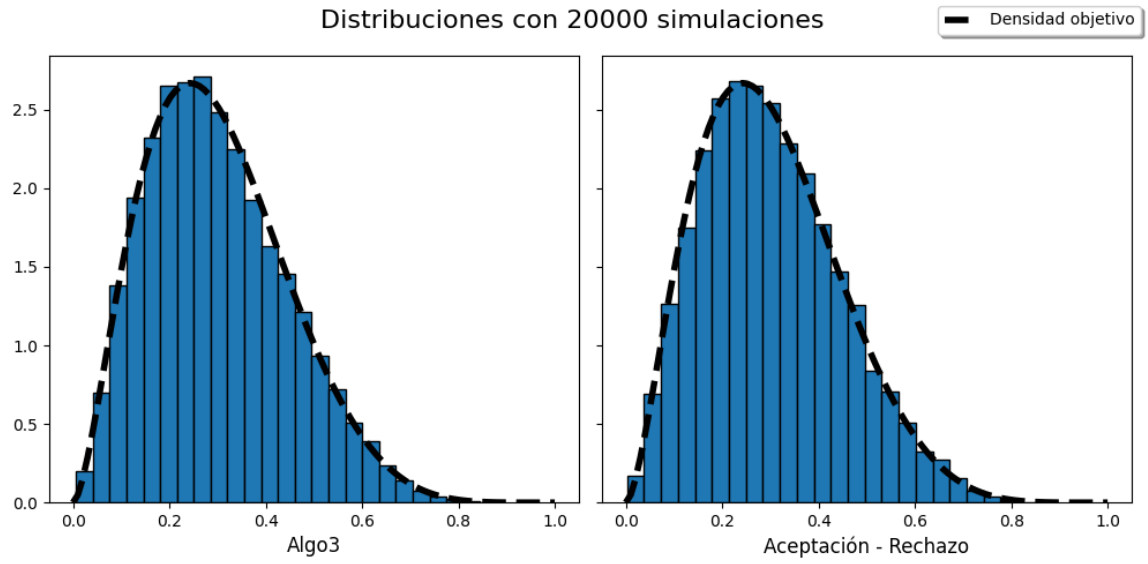
Otra forma de comparar los algoritmos es analizando la cantidad de iteraciones necesarias para aceptar una simulación. Para llevar a cabo esta comparación, realizamos 10.000 simulaciones y registramos el número de iteraciones requeridas para aceptar cada simulación. Es importante tener en cuenta que ingresar al algoritmo ya cuenta como una iteración, por lo que al menos se requiere una iteración para aceptar una simulación. A continuación, presentamos los resultados obtenidos en un boxplot:



En el boxplot anterior, se puede observar que el algoritmo *Algo3*, en promedio, requiere más iteraciones para aceptar una simulación en comparación con el algoritmo de *Aceptación-Rechazo*. Específicamente, *Algo3* requiere un promedio de aproximadamente 2.69 iteraciones para aceptar una simulación, mientras que *Aceptación-Rechazo* necesita un promedio de aproximadamente 1.68 iteraciones. Esto representa una diferencia promedio de 1 iteración a favor del algoritmo de *Aceptación-Rechazo*.

Además, *Algo3* muestra una mayor variabilidad en la cantidad de iteraciones necesarias en comparación con el algoritmo *Aceptación-Rechazo*. También notamos que, en el caso de *Algo3*, en el 50 % de las simulaciones, se necesitaron menos de 5 iteraciones para aceptar una simulación, mientras que en el caso del algoritmo de *Aceptación-Rechazo*, se necesitaron menos de 2.5 iteraciones en el 50 % de las veces. Sin embargo, ambos algoritmos muestran valores atípicos, llegando incluso a realizar alrededor de 20 iteraciones para aceptar una simulación. Esto podría explicarse por la baja tasa de aceptación teórica ($1/M \approx 37\,\%$) que tiene nuestro problema.

Para verificar que los algoritmos generen simulaciones correctamente desde la distribución objetivo, realizamos 20.000 simulaciones desde cada algoritmo y comparamos los histogramas normalizados resultantes con la densidad objetivo. A continuación, se presentan los resultados:



Podemos observar que ambos algoritmos se acercan a la densidad esperada al utilizar 25 barras en cada histograma. Si observamos con detalle, notamos que visualmente la aproximación de *Aceptación-Rechazo* es ligeramente mejor en el punto máximo de la densidad.

Para resumir la comparación, se presenta una tabla comparativa entre ambos algoritmos:

Algoritmo	Algo3	Aceptación-Rechazo
Promedio tiempo ejecución (s)	0,026236	0,02578
Promedio iter. de aceptación	$\sim 2{,}7$	$\sim 1{,}7$

En resumen, al comparar los algoritmos *Algo3* y *Aceptación-Rechazo* en la simulación de Beta(2,7, 6,3), encontramos que ambos algoritmos muestran un desempeño similar en términos de la aproximación de la densidad objetivo. Sin embargo, existen diferencias en cuanto a los tiempos de ejecución y la cantidad de iteraciones necesarias para aceptar una simulación. *Aceptación-Rechazo* demuestra ser más eficiente en términos de iteraciones, con una diferencia promedio de aproximadamente 1 iteración a su favor. Aunque *Algo3* tiende a requerir más iteraciones, las diferencias no son significativas en la práctica. En cuanto a los tiempos de ejecución, *Algo3* muestra un tiempo promedio ligeramente mayor que *Aceptación-Rechazo*, lo cual se alinea con su naturaleza recurrente. En general, ambos algoritmos muestran un comportamiento similar, lo cual es esperado, ya que en la práctica realizan la misma tarea y la única diferencia radica en su forma de implementación en el código, uno utiliza recurrencias, mientras que el otro emplea un ciclo para llevar a cabo la simulación.

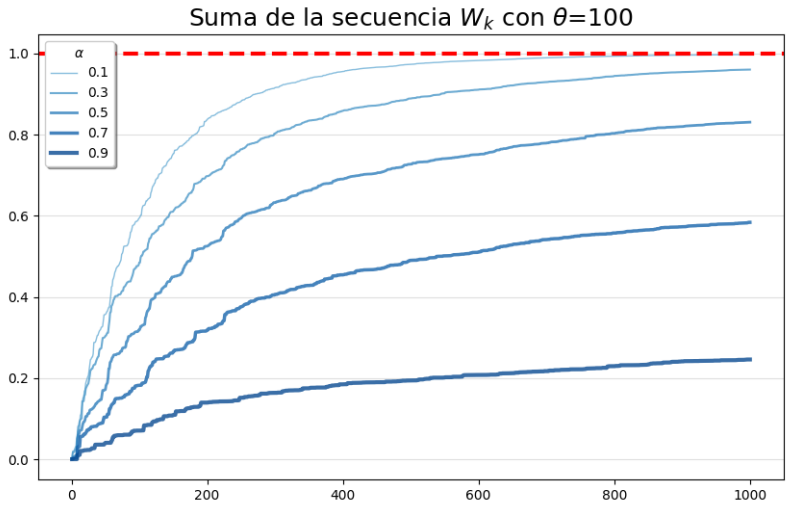
Problema 3

Pregunta a)

Los códigos de esta pregunta se encuentran en el archivo `P3_a.ipynb`. Al programar las secuencias de números aleatorios primero se comprobó experimentalmente las propiedades de la secuencia W_k para distintos valores de α, θ y largo de la secuencia. Posteriormente, se llevaron a cabo simulaciones de secuencias W_k utilizando distintos valores de parámetros. Se intentó generar diferentes tipos de gráficos, incluyendo histogramas, gráficos de dispersión y otros, con el propósito de comparar los efectos de los parámetros. Sin embargo, los resultados obtenidos no resultaron interpretables en su mayoría, ya que en la mayoría de los casos se observó una marcada tendencia de los valores de las secuencias hacia el valor 0. Dada la distribución de S_k y la definición de W_k es esperado que los valores de la secuencia W_k sean pequeños, pues $0 \leq 1 - S_k \leq 1$, luego al tomar el producto de estos se obtendrá un valor más pequeño. Como se mencionó, en los gráficos realizados en primera instancia los últimos valores de las secuencias truncadas simuladas eran muy cercanas a 0, incluso se observaba como si todas fueran 0. Es por esto que se buscó otra forma de comparar las secuencias generadas por distintos parámetros.

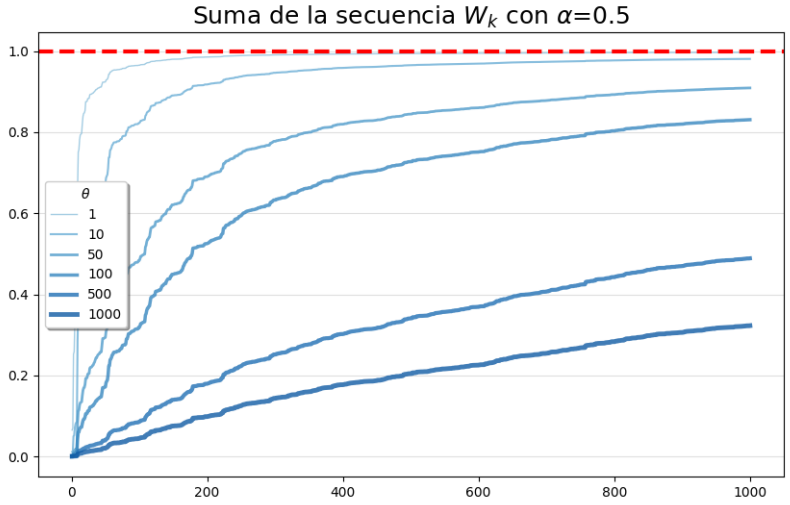
Sabemos que las sumas parciales de la secuencia W_k se aproximan a 1, por lo que, para el mismo largo de secuencias (1000), gráficamos la secuencia de sumas parciales de la secuencia W_k variando un parámetro a la vez, manteniendo fijo el otro.

Para analizar el efecto de α, se fijó $\theta = 100$, obteniendo el siguiente resultado,

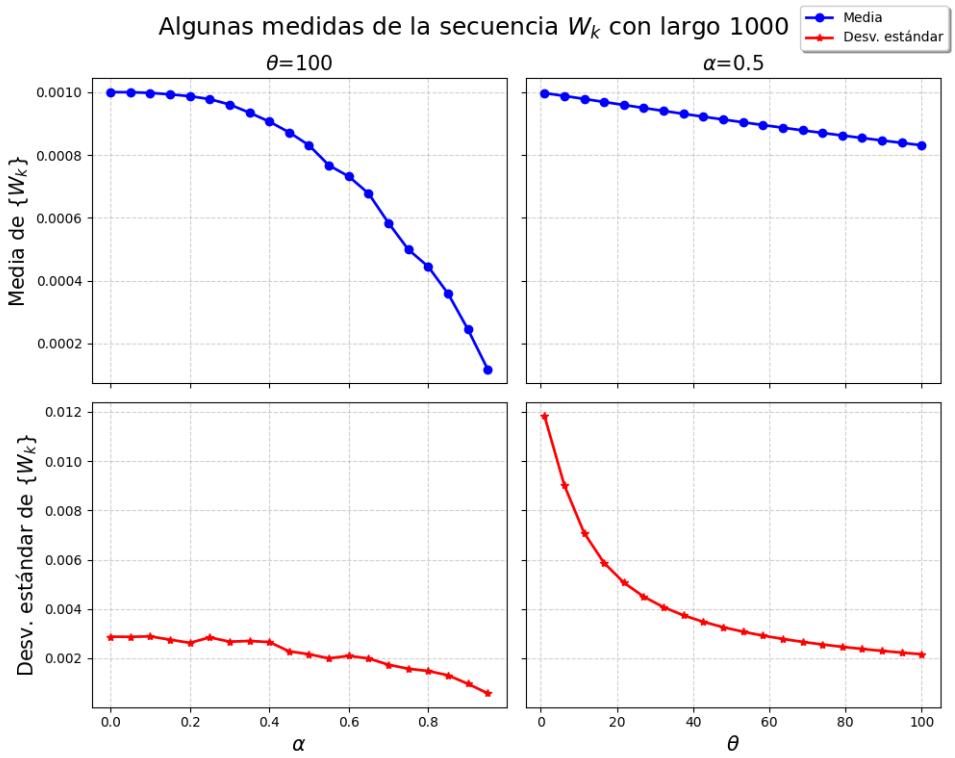


es claro el efecto que tiene el parámetro α en la secuencia W_k cuando se mantiene fijo el parámetro θ. A medida que aumenta el valor de α se necesita alargar la secuencia para que sus sumas parciales converjan a 1, es decir,

α controla la tasa de crecimiento de las sumas parciales. Este resultado lo podemos interpretar de otra forma, a medida que aumenta α para θ fijo, se observa que los valores de la secuencia son más pequeños, es decir, para α mayor la distribución de W_k se concentra en 0. Por otro lado, si fijamos $\alpha = 0{,}5$ y variamos θ, obtenemos resultados similares como se observa a continuación,



Para pequeños valores de θ las sumas parciales de las secuencias W_k requieren de menos cantidad para converger a 1, lo que nos dice que, para valores pequeños de θ con α fijo se obtiene que los primeros valores de la secuencia W_k tiene mayor magnitud que para θ grande.
Como se mencionó previamente, la representación en forma de histograma de la secuencia W_k no tiene gran valor debido a su fuerte tendencia hacia el valor 0. En lugar de eso, con el propósito de evaluar el efecto de los parámetros en la distribución de W_k, optamos por presentar un gráfico que muestra ciertas medidas de la secuencia, tales como la media y la varianza para distintos valores de los parámetros.



La imagen anterior respalda las observaciones previas, ya que al aumentar uno de los parámetros mientras se mantiene el otro constante, los valores de la secuencia W_k tienden a concentrarse cerca de 0. Esto se refleja en la disminución tanto de la media como de la desviación estándar, que se vuelven próximas a 0. Estos valores proporcionan información importante sobre el efecto de los parámetros en la distribución de W_k, indicándonos

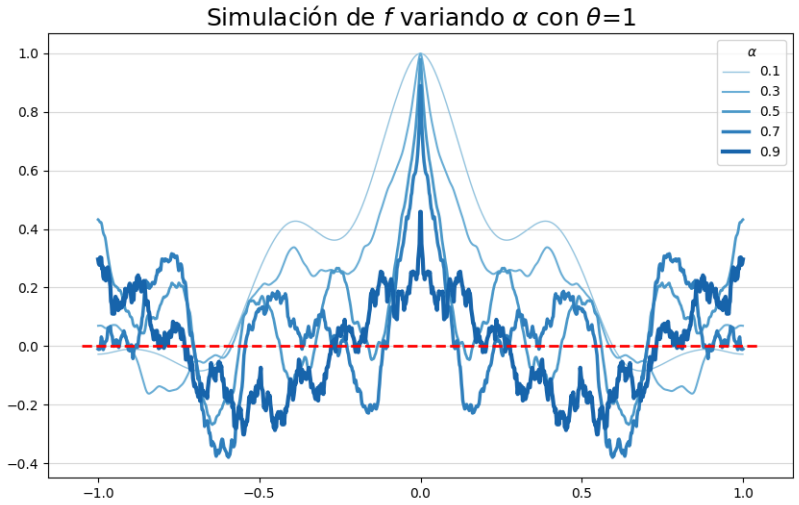
que una disminución en la media sugiere que la distribución se traslada al 0 si aumentamos el parámetro, y una disminución en la desviación estándar nos sugiere que la variación alrededor de la media se reduce, es decir, la distribución se concentra al aumentar uno de los parámetros.

Pregunta b)

Los códigos de esta pregunta se encuentran en el archivo `P3_b.ipynb`.

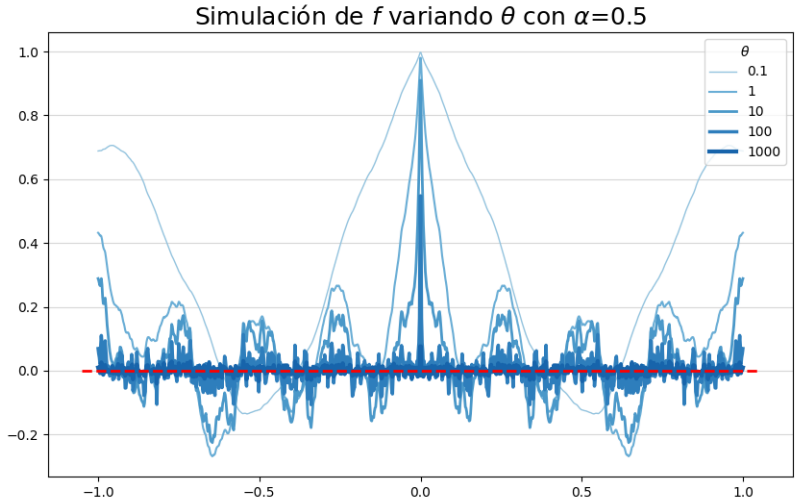
1) Para generar funciones pares y periódicas haremos uso de la conocida base de cosenos de $L^2[-1,1]$ dada por $\{\varphi_n\}_{n=0}^{\infty}$ con $\varphi_n(x) = \cos(n\pi x)$. Además, para hacer uso de la representación de (2) consideraremos como pesos la secuencia W_k. Esta combinación de pesos y base ortogonal, nos aseguran que las funciones simuladas tengan las propiedades pedidas, la paridad y periodicidad se heredan directamente de la base de cosenos.

   A continuación, presentamos el gráfico de las funciones simuladas para distintos valores de los parámetros. Primero analizamos el comportamiento del parámetro α, para esto fijamos el valor de $\theta = 1$ y consideramos las funciones truncadas hasta el término $k = 1000$. Dada la restricción de los valores de α en la distribución Beta, probamos valores tal que $\alpha < 1$.



   Sabemos que la función f sin truncar debe estar anclada a 1 en $x = 0$, pues $\sum_{i=0}^{\infty} W_k = 1$, en esta linea observamos que al aumentar α la función se aleja de 1 en $x = 0$, lo cual se alinea con las observaciones realizadas en la pregunta anterior, para θ fijo si se aumenta α se requieren más términos para que la suma parcial converja a 1. Además, es evidente cómo las funciones oscilan alrededor del eje X. A medida que el valor de α aumenta, las funciones exhiben una mayor cantidad de oscilaciones, volviéndose más 'erráticas'. Por el contrario, cuando α es pequeño, las funciones tienden a adquirir un carácter más suave y menos oscilante. Por otro lado, para examinar el efecto de θ, fijamos $\alpha = 0{,}5$ y truncamos hasta el término $k = 1000$

   las funciones. A continuación, se presentan los resultados de funciones simuladas para distintos valores de θ en el rango 0.1, 1000.

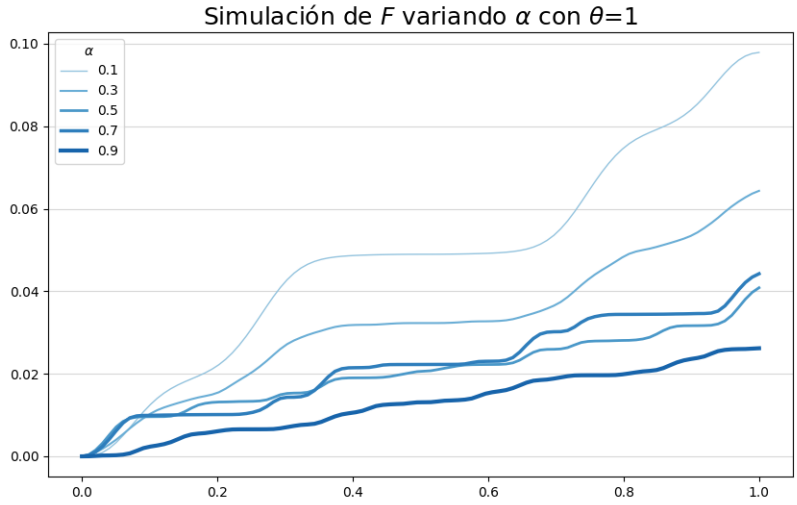


Se observa un comportamiento análogo al caso de variar α, se puede apreciar, como al aumentar θ las funciones simuladas tienden a oscilar en torno al eje X aumentando la cantidad de oscilaciones y como en $x = 0$ las funciones se alejan del valor esperado 1.

2) Para generar funciones con las características requeridas, haremos uso de la representación (2) del enunciado como paso intermedio, pues le aplicaremos transformaciones para formar la función final. En específico, usaremos la base ortogonal de senos de $L^2[0,1]$ dada por $\{\varphi_n\}_{n=0}^{\infty}$ con $\varphi_n(x) = \sin(2n\pi x)$ y la secuencia W_k como pesos para la simulación de f con la representación (2). Luego, componemos con una función $g \geq 0$ a elección, por ejemplo $g(x) = \exp(x), x^2$, entre otras. Finalmente, definimos la función final como sigue,

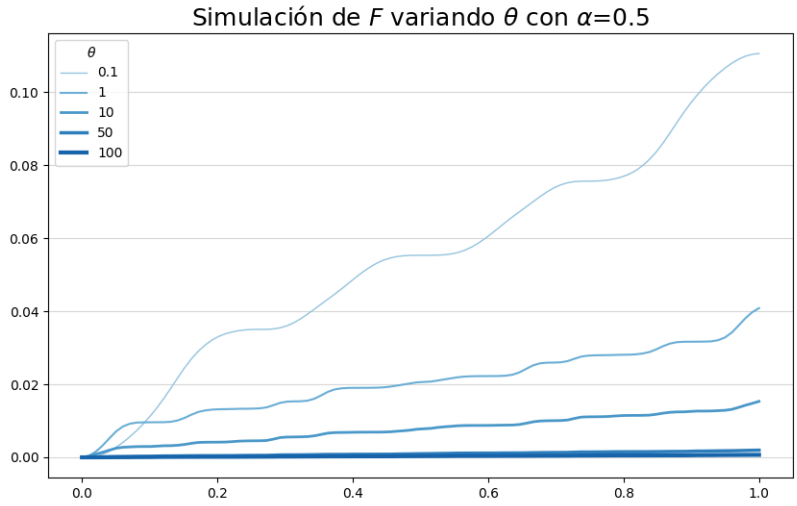
$$F(x) = \int_0^x g(f(t))\mathrm{d}t$$

dado que $g \circ f$ es una función positiva, tenemos que F es estrictamente creciente, asegurando la propiedad pedida. Para implementar el método, elegimos $g(x) = x^2$ y aproximamos el valor de la integral numérica usando la regla del trapecio implementada en la librería *NumPy*. Analicemos el comportamiento de las funciones simuladas, para distintos valores de los parámetros. Primero analizamos α, para esto fijamos $\theta = 1$ y variamos el parámetro en el intervalo $(0, 1)$. Obtuvimos el siguiente resultado gráfico,



en la imagen anterior se puede observar que efectivamente las funciones simuladas son crecientes en el intervalo $[0, 1]$. Además, es claro el impacto que presenta α al simular las funciones: a medida que α aumenta a 1 el máximo de la función simulada disminuye y se acerca a 0. Esta observación complementa lo observado en la pregunta anterior, cuando aumenta α para θ fijo, los valores de W_k son más cercanos a 0, por lo que, la función a integrar es más cercana a 0 y por ende su función integral igual.

Un comportamiento similar ocurre al fijar el parámetro $\alpha = 0{,}5$ y variar θ, como se puede observar a continuación,

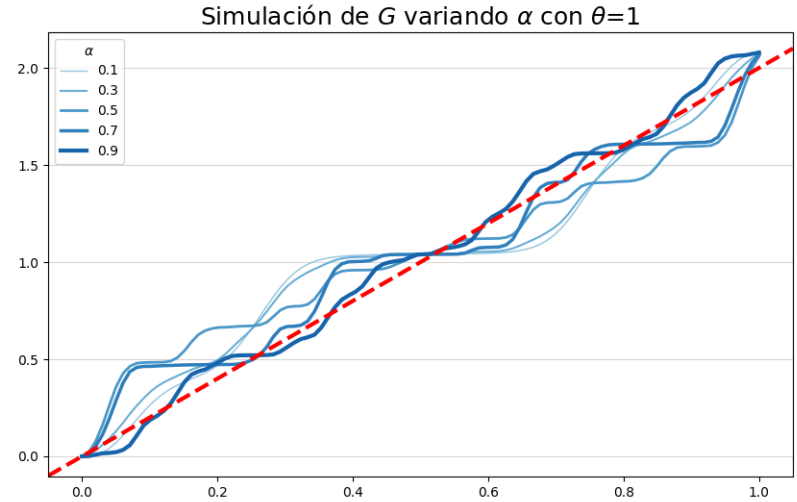


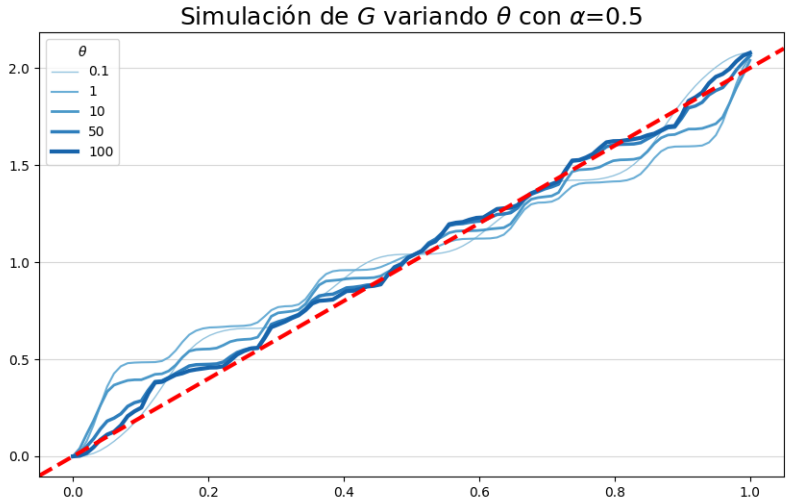
A medida que aumenta θ, disminuye el rango de la función, esto se puede explicar debido a que cuando aumenta θ los W_k se vuelven más pequeños y cercanos a 0, análogo a la interpretación dada al variar el parámetro α.

3) Para generar densidades en el $[0, 1]$, es decir, funciones positivas que integren 1, haremos uso de la representación (2) del enunciado, utilizando la base de senos y la secuencia W_k, generando f. Luego, basta componer con una función $g \geq 0$, para finalmente normalizar por el área en el intervalo $[0, 1]$, de esta forma obtenemos funciones G con las propiedades pedidas,

$$G(x) = \frac{1}{\int_0^1 g(f(t))\mathrm{d}t} g(f(x)).$$

Para obtener resultados interesantes, usaremos las funciones obtenidas en la pregunta 3b)2), es decir, funciones estrictamente crecientes y estas las normalizaremos. Simulamos variando un parámetro a la vez, obteniendo los siguientes resultados,

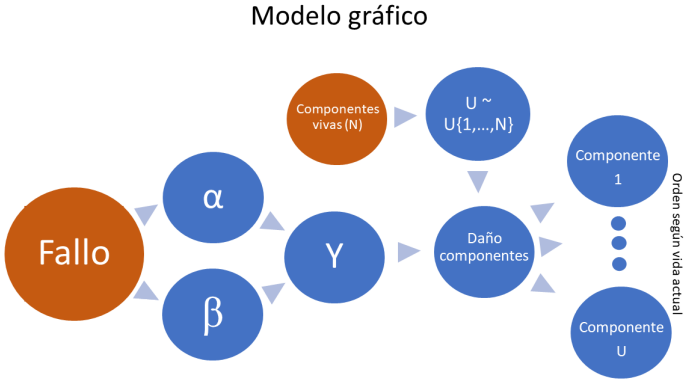




No se observa un patrón o impacto que tienen los parámetros en las funciones simuladas. Sin embargo, notamos un hecho interesante, debido a la naturaleza creciente de las funciones tenemos que estas oscilan en torno a la función $y = 2x$ (rojo), lo cual se explica pues está función integra exactamente 1 en el intervalo $[0, 1]$.

Problema 4

Para llevar a cabo el estudio de simulación del problema propuesto, se implementó un código en Python, que puede ser reproducible a través del archivo `p4.ipynb`. En este código se programó el Interruptor como objeto *class* de Python, lo que nos permite mediante métodos: aplicar daño, verificar la vida del interruptor en cualquier instante, conocer la vida de cada componente, observar el daño realizado hasta el momento a cada componente, entre otros. A continuación, se presenta de forma gráfica el esquema de modelación que se implemento:

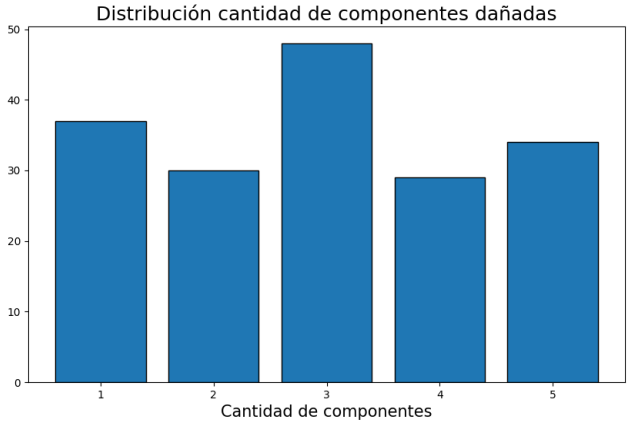


En palabras, cuando ocurre una falla se simulan los parámetros α, β desde sus distribuciones correspondientes sugeridas en el enunciado, luego, se simula el daño total (Y) que será distribuido a las componentes, se comprueba cuantos componentes continúan vivos para simular la cantidad de componentes que recibirán el daño, esto es una distribución uniforme discreta con máximo la cantidad de componentes vivas hasta ese momento. Una vez conocida la cantidad de componentes a dañar, se reparte uniformemente el daño entre las componentes con mayor vida hasta ese momento, finalizando la simulación de fallo al interruptor.
Se realizaron los siguientes supuestos intuitivos al modelo:

- Una vez muerto un componente, este no puede recibir más daño.
- Si el interruptor está inoperativo, este no puede recibir más daño, es decir, no puede fallar.

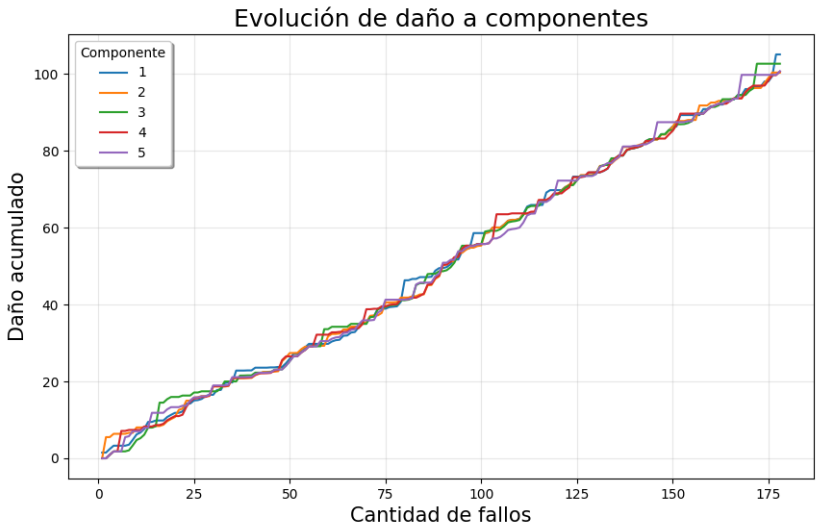
Simulación de un interruptor

Antes de pasar a los tiempos entre fallas, se hizo la simulación de dejar inactivo un interruptor a través de fallas, obteniendo algunos gráficos de interés, como la distribución de la variable aleatoria U presentada en el modelo gráfico,



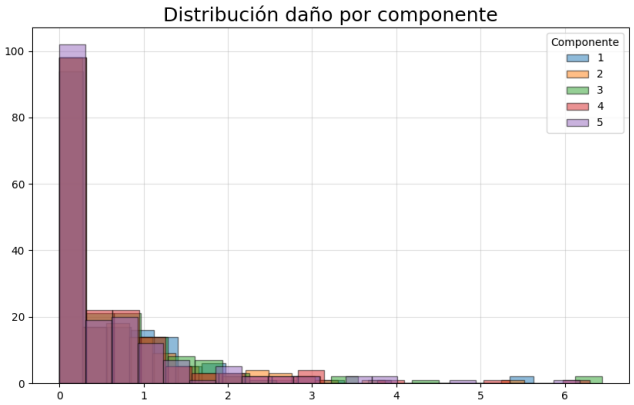
no se observa gran tendencia por algún valor, se espera que la distribución sea cercana a la distribución $U(\{1, \ldots, 5\})$, pues gran parte del proceso estarán vivas todas las componentes, luego de muchas fallas la cantidad de componentes vivas disminuye, por lo que, se esperaría un sesgo a valores pequeños, pero son tan pocos fallos restantes que no es observable.

Por otro lado, presentamos la evolución del daño en cada componente hasta la muerte del interruptor,



Observamos que la tasa de daño a cada componente es similar para todas, en otras palabras, ninguna componente se adelanta a morir, esto tiene sentido, pues nuestro modelo garantiza que el daño se reparte en las componentes que han tenido menos daño hasta ese momento, por eso el daño entre las componentes se va compensando a medida que ocurren fallas.

Otro gráfico de interés, es la distribución del daño por componente,

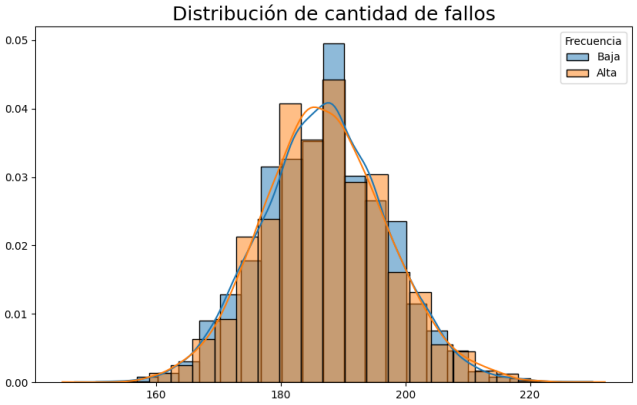


en linea con el gráfico de evolución de daño, observamos que la distribución de daño en las componentes es similar para todas, con gran sesgo a 0, esto se explica pues el daño total se reparte entre componentes, y por construcción del modelo algunos interruptores pueden no recibir daño en un fallo, en esas ocasiones su daño es 0, lo que queda en el historial de daños y es observado en el histograma.

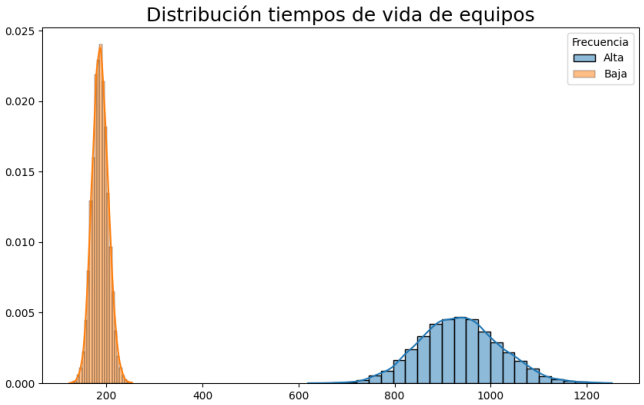
Tiempos entre fallas

Para volver más realista nuestro modelo, proponemos distribuciones para tiempos entre fallas, consideramos dos tipos según el tráfico del interruptor: alta y baja frecuencia entre fallas. Dada la interpretabilidad del parámetro, se propone una distribución Exponencial, con parámetro $\lambda_{\text{baja}} = 1$ y $\lambda_{\text{alta}} = 5$ con unidad [falla/unidad de tiempo], la unidad de tiempo podría ser: semanas, días, horas, entre otros. En nuestro caso, creemos coherente que un interruptor falle 1 vez por semana, por lo que usaremos esa unidad de tiempo al interpretar los resultados.

Para reportar resultados, simularemos para cada frecuencia 10.000 interruptores, aplicando daño hasta dejarlos inoperativos, almacenamos la cantidad de fallas necesarias hasta dejarlos inoperativos y el tiempo de vida de los equipos. A continuación, presentamos el histograma de la cantidad de fallas,



observamos que la cantidad de fallos distribuye de igual forma para ambas frecuencias, esto no hace sentido, pues la cantidad de fallas es independiente del tiempo entre fallas, a simple vista la distribución se asemeja a una normal, cual se podría comprobar o rechazar evaluando un test de Kolmogorov-Smirnov. La media de cantidad de fallos para frecuencia baja es 187.0259, mientras que para frecuencia alta es 187.0766, valores muy cercanos lo que nos da una estimación de fallas que soporta un interruptor según nuestro modelo, esto podría ser de gran interés en la industria.
Por otro lado, observamos a continuación la distribución del tiempo esperado para cada frecuencia,



observamos como es esperado una clara diferencia en la media de los tiempos para cada frecuencia, para frecuencia baja es 187.0119 y para frecuencia alta 935.6135. Los valores anteriores tienen fuerte relación con la cantidad de fallos esperada,

$$\text{media(cantidad de fallas)} \cdot \lambda = \text{tiempo esperado}$$

para frecuencia baja como $\lambda_{\text{baja}} = 1$ se cumple lo observado en la ecuación, pues la media de las fallas es ~ 187. Por otro lado, recordemos que consideramos para frecuencia alta $\lambda_{\text{alta}} = 5$, observando el siguiente cálculo,

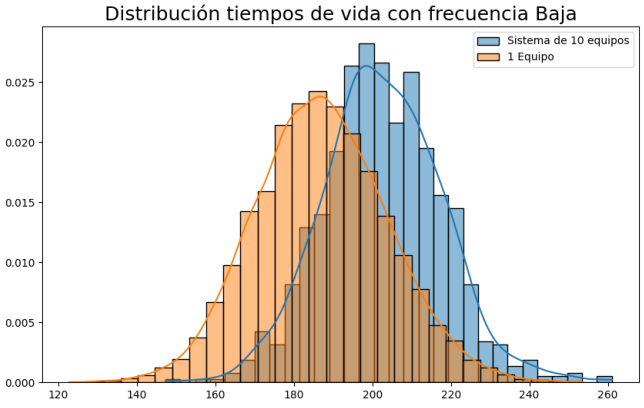
$$\text{media(cantidad de fallas)} \cdot \lambda \approx 187 \cdot 5 = 935 \approx 935{,}6135 = \text{tiempo esperado}$$

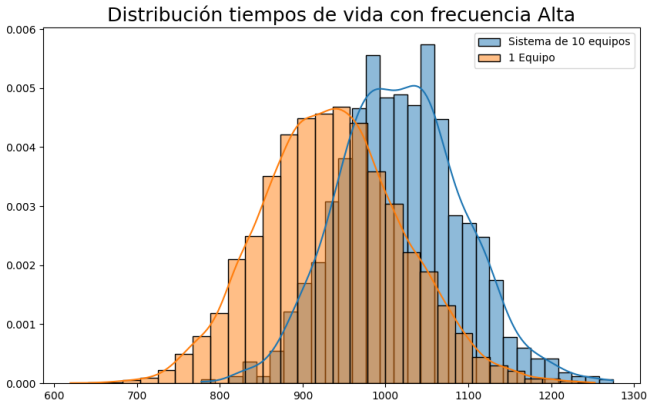
notamos que la relación se cumple. Esta relación es de gran valor, pues si conocemos en la práctica la media de fallas que soportan los equipos y cada cuanto fallan, podemos estimar cada cuanto tiempo debemos cambiar los equipos.

Sistemas de interruptores

Para finalizar con la modelación, se nos plantea una medida que toma la empresa, colocar varios equipos en paralelo, para esto asumiremos un simplificación importante en nuestro modelo: los interruptores en paralelo tienen fallas independientes, es decir, cuando falla un sistema, falla cada uno de los interruptores de forma independiente según el modelo planteado inicialmente.
Dado que un sistema queda inoperativo una vez todos sus interruptores están inoperativos, es de gran interés conocer si se cumple la idea intuitiva de que los sistemas tienen mayor esperanza de vida que un solo interruptor, esto da respuesta a la pregunta: ¿Un sistema entrega mayor protección?

Para responder a esta pregunta realizamos 10.000 simulaciones de sistemas con 10 equipos cada uno, almacenamos el tiempo de vida de los sistemas, es decir, el tiempo que tarda en quedar inoperativo. Realizamos un gráfico comparativo de la distribución del tiempo de falla de los sistemas versus la de tener un solo interruptor funcionando. A continuación, presentamos los resultados para cada frecuencia por separado,





observamos que el tiempo esperado tiene un desplazamiento si comparamos el sistema versus un solo interruptor, lo cual nos dice visualmente, que un sistema tiene un promedio de vida mayor a un solo interruptor, respondiendo directamente la interrogante. Lo anterior, tiene sentido, pues hemos asumido que los daños a los interruptores son independientes, por lo tanto, el tiempo de muerte de un sistema es el máximo tiempo de muerte de sus equipos, lo cual explica el desplazamiento de la distribución. Para argumentar con valores, tenemos que para la frecuencia baja el tiempo esperado hasta quedar inoperativo para un sistema de 10 interruptores es 202.64, en cambio para un solo interruptor con esta frecuencia es de 187.01, para alta frecuencia la relación es análoga. Por lo tanto, podemos concluir que un sistema de interruptores tiene mayor esperanza de vida que un solo interruptor, tal como nuestra intuición nos decía.

Por temas de tiempo no se pudo seguir mejorando el modelo o encontrando nuevas relaciones, para un trabajo futuro se podría,

- Realizar test de hipótesis a las diferentes distribuciones obtenidas.
- Investigar si existe relación entre la cantidad de interruptores que tiene un sistema y el aumento en tiempo esperado, que intuitivamente se espera que si, que la relación sea directa, a más interruptores, más tiempo de vida de un sistema.
- Mejorar el modelo para los sistemas de interruptores, asumimos independencia en los daños, se podría escalar la idea de reparto de daño de los interruptores a sus componentes, a los sistemas. Es decir, proponer un modelo que reciba daño para sistemas, este se reparta de cierta forma a modelar entre sus interruptores y el daño a cada interruptor se reparta entre las componentes.