**Tarea 1**

**Presentado por:**

Jaime Darley Angulo Tenorio – [jangulot@unal.edu.co](mailto:jangulot@unal.edu.co)

John Alejandro Pastor Sandoval – [jpastor@unal.edu.co](mailto:jpastor@unal.edu.co)

Juan Camilo Vergara Tao – [juvergarat@unal.edu.co](mailto:juvergarat@unal.edu.co)

Juan Diego Velásquez Pinzón – [jvelasquezpi@unal.edu.co](mailto:jvelasquezpi@unal.edu.co)

**Modelos y Simulación**

Grupo ModSim\_303

**Profesor:**

Luis Gerardo Astaiza Amado

lgastaizaa@unal.edu.co

**Logotipo, nombre de la empresa

El contenido generado por IA puede ser incorrecto.**

**Universidad Nacional de Colombia**

**Facultad de Ingeniería**

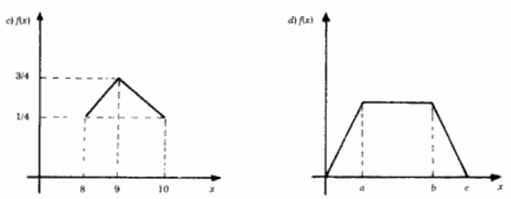
**Viernes, 9 de mayo de 2025**

# Generación de variables aleatorias

## Ejercicio 1

Desarrollar los algoritmos correspondientes al:

* 1. Método de la transformada inversa
  2. Método de composición que siga las siguientes funciones de densidad de probabilidad.



1. La transformada inversa es una técnica para generar números aleatorios que siguen una distribución específica, partiendo de números generados con una distribución uniforme en el intervalo [0,1].

### Marco Teórico a

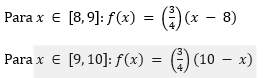
1. **Función de Distribución Acumulativa (CDF):**  
    Se parte de una función de distribución acumulativa *F(x)*, que representa la probabilidad de que una variable aleatoria tome un valor menor o igual que *x*.
2. **Generación de Variables Uniformes**:  
    Se generan valores aleatorios *R* distribuidos uniformemente entre *0* y *1*.
3. **Inversión de la CDF:**  
    Se resuelve la ecuación para encontrar el valor *x* correspondiente al valor uniforme *R*. Esta es la parte clave del método.
4. **Obtención de la Variable Aleatoria:**  
    Evaluando , se obtiene una variable aleatoria con la distribución deseada.

Para la gráfica con distribución triangular hacemos lo siguiente:

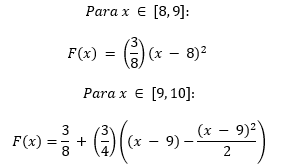
Esta es una distribución triangular definida entre *x = 8 y x = 10*, con un pico en *x = 9* y altura máxima de . La función de densidad está dividida en dos tramos.

### Desarrollo de algoritmo a

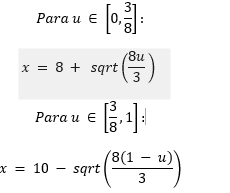
**1. Función de densidad**



**2. Función de distribución acumulativa (CDF)**

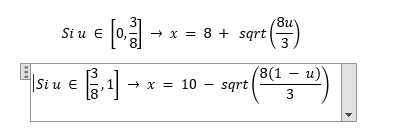


**3. Inversión de la CDF**



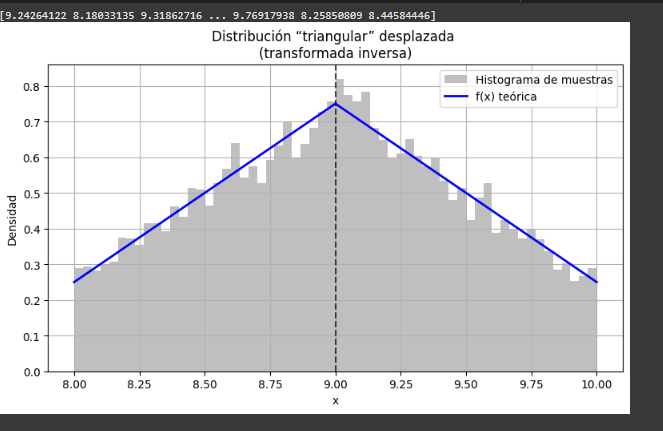
**4. Resultado final**

Función inversa por tramos:



Con el código de python[Generacion de variables aleatorias.ipynb](https://colab.research.google.com/drive/18YD8dXChf4wN2HT4ZXNeNt4yEcGXhjQ8#scrollTo=ZdEvq9YQa2cv) obtendremos lo siguiente:

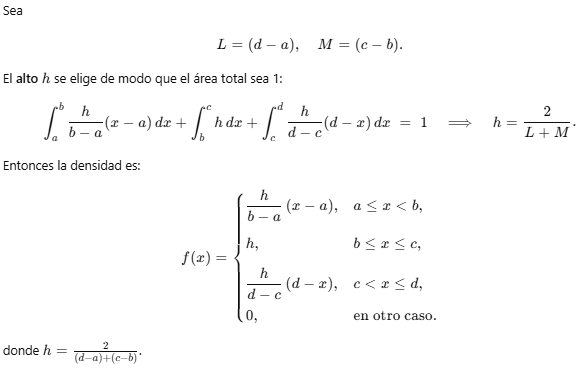
### Gráfica inversa triangular:



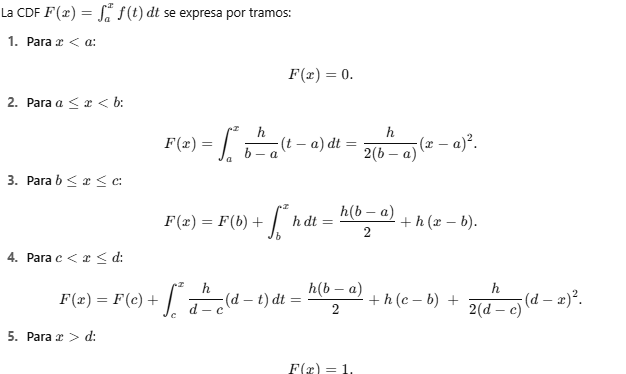
Para la distribución trapezoidal la conoceremos como una generalización de la triangular en la que la densidad es constante en un intervalo central, y crece/decrece linealmente en los tramos de los extremos. Se suele parametrizar por cuatro puntos , donde:

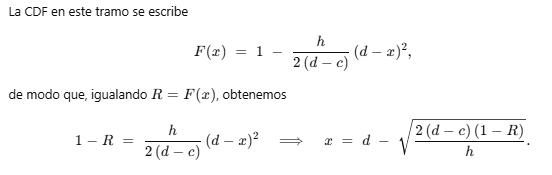
* El soporte es [a,d].
* La densidad crece linealmente desde *0* en *x = a* hasta un valor máximo *h* en *x = b*.
* Permanece constante igual a *h* sobre [b,c].
* Decrece linealmente de *h* en *x = c* hasta *0* en *x = d*.

1. **Función de densidad (PDF)**

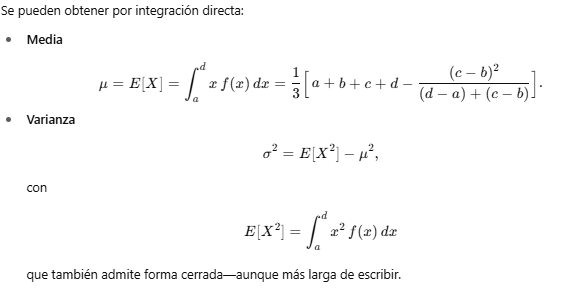


1. **Función de distribución acumulada (CDF)**

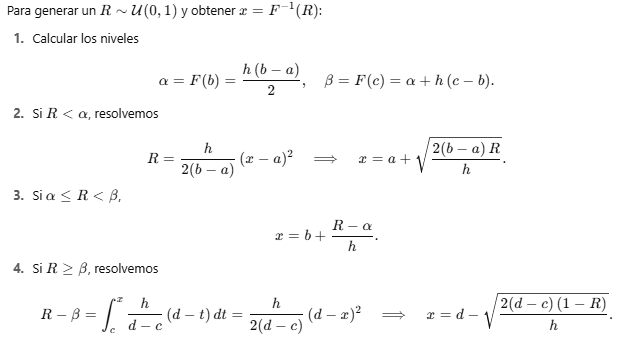
****

****

1. **Media y varianza**

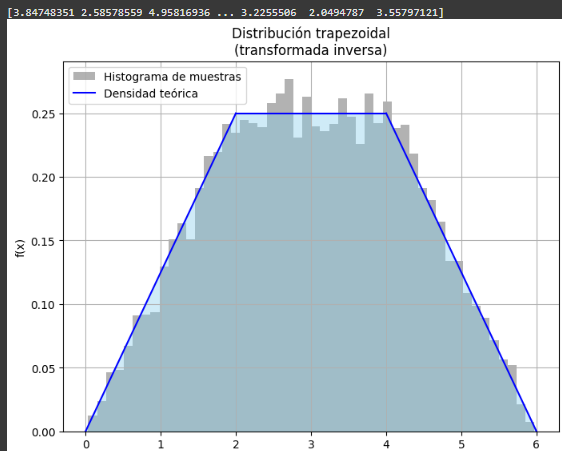
****

1. **Método de la transformada inversa**

****

Con el código[Generacion de variables aleatorias.ipynb](https://colab.research.google.com/drive/18YD8dXChf4wN2HT4ZXNeNt4yEcGXhjQ8#scrollTo=V8ZT_-H1a9s7) de python obtendremos lo siguiente:

### Gráfica inversa trapezoidal:



**b.** Para la generación de números aleatorios por el método de composición, debemos seguir unos pasos específicos, ya que esta técnica para generar variables aleatorias usa las relaciones que hay entre distintas distribuciones de probabilidad para generar una más compleja viniendo de unas simples.

### Marco Teórico b

1. **Identificación de las funciones de distribución y sus pesos:** Se eligen varias distribuciones conocidas **,,...,**, cada una con un peso asociado. Estos pesos deben ser positivos y la suma de todos ellos debe ser igual a 1.
2. **Obtención de las funciones de densidad :** Para cada distribución , se obtiene su correspondiente función de densidad de probabilidad .
3. **Aplicación del método de la transformada inversa a cada :  
    Se calcula la función inversa:**

para poder generar variables aleatorias que sigan cada distribución individual.

1. **Generación del primer número aleatorio :** Se genera un valor ​ uniformemente distribuido en el intervalo *(0,1)*.
2. **Selección de la función correspondiente:** Se determina en qué tramo cae el valor de ​, de acuerdo con los pesos.  
    Por ejemplo:  
   * Si ​, usar
   * Si ​, usar
   * Y así sucesivamente.
3. **Generación del segundo número aleatorio ​:** Se genera otro valor aleatorio ​ también uniforme en *(0,1)*, y se evalúa en la función inversa correspondiente al tramo seleccionado.

### Desarrollo de algoritmo b

Por lo tanto para la primera gráfica triangular tenemos:

* En el intervalo [8,9] hallamos la pendiente para la ecuación de la recta

* Hacemos el mismo proceso para [9,10]

* toma esos valores dependiendo del intervalo, y es 0 si el valor está fuera de esos intervalos. Ahora procederemos a integrar en ambos lados de la función para obtener la acumulada.
* De esta manera conseguimos que esta función sea igual a si el valor de *x* está entre [8,9] de lo contrario es 0,repetimos el procesos para el otro intervalo [9,10]
* Acá conseguimos que la función sea igual a si el valor de *x* está entre [9,10] de lo contrario es 0

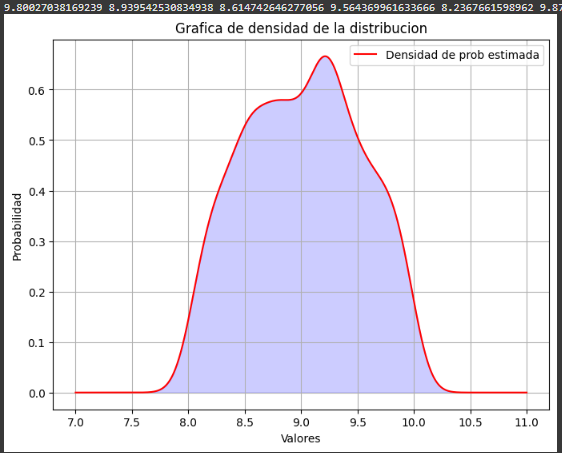
De esta manera logramos que , para entonces determinar las funciones acumuladas: y , haciendo uso de python usaremos la función random de python que nos da números entre 0.0 y 1.0, mientras manualmente reemplazamos las *f(x)* con random() para conseguir unos números que denominaremos y consiguiendo los siguientes valores para *x*:

para :

de lo contrario:

Cuando tenemos todo esto, configuramos el código en python [Generacion de variables aleatorias.ipynb](https://colab.research.google.com/drive/18YD8dXChf4wN2HT4ZXNeNt4yEcGXhjQ8#scrollTo=vipEhgWiUB9d)y obtenemos lo siguiente:

### Gráfica composición triangular:



Para la segunda gráfica nos interesa que el área total del trapecio valga 1, este trapecio está dividido en 3 tramos ya que aca no sirve la formula para calcular el área del trapecio.

sabemos que el área vale 1, entonces despejamos la fórmula para encontrar *h* y nos queda :

Esta altura la usaremos para definir la función para los 3 tramos:

Tramo [0,a]: recta que va desde hasta , osea tiene pendiente:

entonces

Tramo [a,b]: recta que va desde hasta , osea tiene pendiente:

entonces

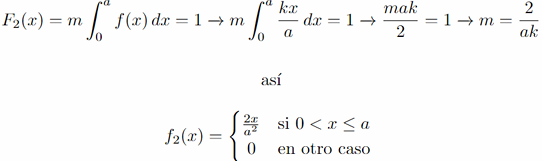
Tramo [b,c]: recta que va desde hasta , osea tiene pendiente:

entonces

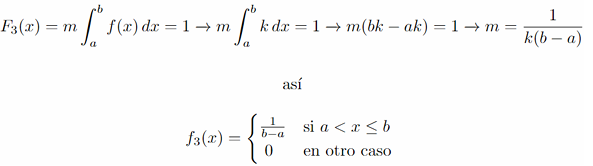
entonces la función de densidad donde los valores que toma *f(x)* son:

Con la función de densidad de probabilidad integramos en los tramos de la función para hallar *m* y la distribución acumulada:

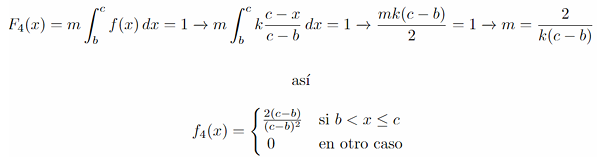
* Para el intervalo [0,a]



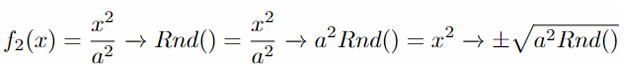
* Para el intervalo [a,b]



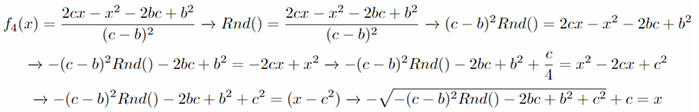
* Para el intervalo [b,c]



sí se satisface que y se determinan las funciones de distribución acumulada como , y , se sustituye ,, con la función random y despejamos *x* obteniendo.







Se genera un número aleatorio R1 y R2:

para :

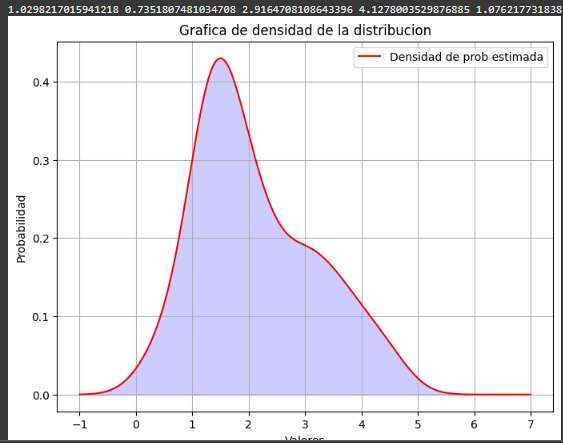
para :

de lo contrario:



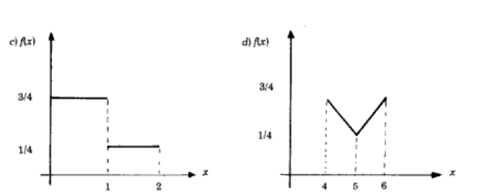
Cuando tenemos todo esto, configuramos el código en python [Generacion de variables aleatorias.ipynb](https://colab.research.google.com/drive/18YD8dXChf4wN2HT4ZXNeNt4yEcGXhjQ8#scrollTo=CB6eULOSUYAQ)y obtenemos lo siguiente:

### Gráfica composición trapezoidal:

****

## Ejercicio 2

Generar por el método de rechazo, números al azar que sigan las siguientes distribuciones de probabilidad:



#### 1. Método

El método de rechazo, es un algoritmo probabilístico empleado para generar muestras aleatorias a partir de una distribución objetivo , que puede no ser fácil de muestrear directamente. Este método utiliza una distribución propuesta más simple , de la cual se pueden generar muestras fácilmente, y una constante M, que asegura que en todo el dominio de interés.

#### 2. Metodo de aplicacion

1. Se elige una distribución propuesta , usualmente uniforme o una gaussiana, que cubra el soporte de .
2. Se selecciona una constante M tal que para todo x. Esto asegura que la probabilidad de aceptación sea válida en cualquier caso.
3. **Generación de Muestras:**

* Se genera un valor x a partir de .
* Se genera un valor uniforme uu en el intervalo .
* Se acepta x si . En caso contrario, se rechaza y se repite el proceso.

El resultado es un conjunto de muestras distribuidas según la función objetivo .

#### 3. Ventajas y Desventajas

**Ventajas:**

* Muy flexible: funciona con variedad de distribuciones , sin forma cerrada o incluso no normal.
* Permite manejar distribuciones no estándar que serían difíciles de muestrear directamente.

**Desventajas:**

* La eficiencia depende de la elección de y M. Si M es muy grande o no es similar a , muchos valores serán rechazados.
* Dependencia del Escalamiento, puede resultar complicado hallar una constante pequeña cuando es muy irregular o multivariada.

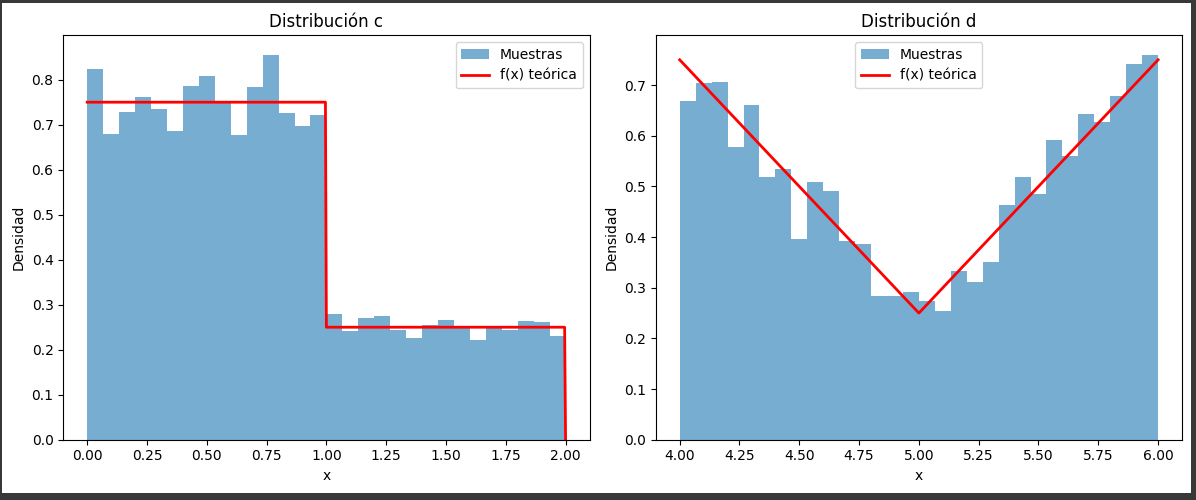
#### 4. Modo de Simulación El algoritmo se implementa a través del paquete *SimPy*, que modela procesos en el tiempo. Cada intento de generación de muestras se simula como un evento, en donde:

* **Tiempo de Evento:** Cada generación de muestra se realiza con un retraso constante o aleatorio, simulando así la ocurrencia de eventos a intervalos.
* **Criterio de Aceptación:** Se evalúa para cada muestra si cumple el criterio .

**5. Algoritmo**

[**Generacion de variables aleatorias.ipynb**](https://colab.research.google.com/drive/18YD8dXChf4wN2HT4ZXNeNt4yEcGXhjQ8#scrollTo=hpC-rab0Izix)

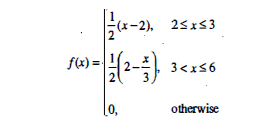
**6. Resultados**



Con los histogramas logramos ver que las muestras generadas por el método de rechazo respetan fielmente las densidades objetivo. Las pequeñas variaciones locales se explican por el número finito de muestras y el ancho de cada intervalo, pero en conjunto confirman que el algoritmo funciona correctamente y produce la forma deseada con cierto error razonable.

## Ejercicio 3

Desarrollar un esquema general para generar valores aleatorios para la distribución triangular con pdf



Genere 10 valores de la variable aleatoria, evalué la media muestral, y compare con la media de la distribución.

### Marco Teórico

Los conceptos y métodos utilizados en el desarrollo del algoritmo fueron los siguientes:

* **Distribución Triangular:** La distribución triangular es una distribución de probabilidad continua definida por límites, los cuáles son: inferior (a), superior (b) y un valor de moda (c) entre ellos. Su función de densidad de probabilidad (PDF) tiene una forma triangular, de ahí su nombre. Se utiliza cuando se tienen estimaciones de los valores mínimo, máximo y más probable de una variable. [1]
* **Función de Densidad de Probabilidad y Función de Distribución Acumulada:** La función de densidad de probabilidad, también conocida como PDF y denotada *f(x)* describe la probabilidad relativa de una variable continua tomando un valor específico; la función de distribución acumulada, también conocida como CDF y denotada *F(x)* da la probabilidad de que la variable sea menor o igual a x. [2]
* **Método de la Transformada Inversa:** Es una técnical para generar números aleatorios que siguen una distribución de probabilidad específica, dada su función de distribución acumulativa (CDF). El método se basa en el principio de que si U es una variable aleatoria que sigue una distribución uniforme en el intervalo (0,1), entonces la variable aleatoria X=F−1(U) tendrá la misma distribución que la función de distribución acumulativa F, donde F−1 es la función inversa de la CDF. [3]

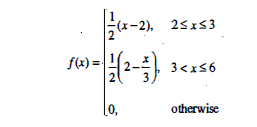
Los pasos a seguir son los siguientes:

1. Encontrar la función de distribución acumulada *F(x)* de la distribución deseada.
2. Obtener la inversa de la CDF,; se hace y se resuelve para en términos de .
3. Generar un número aleatorio u a partir de una distribución uniforme U(0,1).
4. El valor tendrá la distribución deseada.

* **Media de una distribución continua:** También conocido como valor esperado, y denotada , es el valor promedio de X, una variable aleatoria continua con función de densidad de probabilidad f(x), ponderado por su probabilidad de ocurrencia. Representa el centro de la distribución de probabilidad y matemáticamente se define así: [4]
* **Media muestral:** La media muestral (barx) es el promedio de una muestra de datos. La *Ley de los Grandes Números* indica que a medida que el tamaño de la muestra aumenta, la media muestral se acerca a la media de la población, o media teórica en el caso de una distribución de probabilidad continua. [5]
* **Herramientas digitales:**
  + **Python:** Es un lenguaje de programación de alto nivel, versátil y ampliamente utilizado en la ciencia de datos, la estadística y la simulación. [6] Fue el lenguaje de programación utilizado para implementar el algoritmo.
  + **Google Colaboratory (Colab):** Es un entorno de cuaderno Jupyter gratuito basado en la nube que permite escribir y ejecutar código Python a través de un navegador web. [7] Fue el ambiente de desarrollo empleado para implementar el algoritmo.
  + **NumPy y Random:** La primera es una biblioteca fundamental para la computación numérica en Python [8]; y la segunda es un módulo incorporado que implementa generadores de números pseudoaleatorios para diversas distribuciones [9]. Fueron las importaciones utilizadas para implementar el algoritmo, usando random.random() para obtener números aleatorios según una distribución uniforme U(0, 1); y usando np.mean() para calcular la media muestral de los números obtenidos.
  + **Matplotlib:** es un módulo dentro de la biblioteca Matplotlib en Python que proporciona una interfaz similar a la de MATLAB para la creación de gráficos y visualizaciones es un módulo dentro de la biblioteca Matplotlib en Python que proporciona una interfaz similar a la de MATLAB para la creación de gráficos y visualizaciones [10]. Se ha empleado para la generación de las gráficas de función de densidad y el histograma de los valores generados por la variable aleatoria.

### Desarrollo de algoritmo

La función de densidad de probabilidad dada es la siguiente:



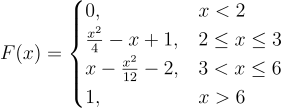
Para generar valores aleatorios a partir de esta distribución, utilizaremos el método de la transformada inversa, a partir de los siguientes pasos:

1. **Encontrar la :**

La función de distribución acumulada se obtiene integrando la función de densidad de probabilidad, de la siguiente manera:

* Para :
* Para :
  + Primero evaluamos :
  + Ahora, integramos la segunda parte:

La función de distribución acumulada completa es:



1. **Invertir (encontrar ):**

Sea , donde . Necesitamos resolver para en términos de .

* Para :

Dado que , tomamos la raíz positiva:

* Para :

Usando la fórmula cuadrática:

Dado que , tomamos la raíz negativa:

La función inversa de la función de distribución acumulada es:



1. **Media de la distribución:**

La media teórica de la distribución dada es:

* Primer integral:
* Segunda integral:

La media de la distribución es entonces:

1. **Algoritmo en Python**

El código del algoritmo implementado se puede encontrar en el siguiente Google Colab:

[Generacion de variables aleatorias.ipynb](https://colab.research.google.com/drive/18YD8dXChf4wN2HT4ZXNeNt4yEcGXhjQ8#scrollTo=8iEMfsfJqI-J)

La funcionalidad se explica a continuación:

1. Crear una lista llamada valores
2. Para i desde 1 hasta 10:
3. Generar un valor aleatorio según una distribución uniforme usando random.random().
4. Si : Calcular .
5. Si : Calcular .
6. Agregar el valor de a valores.
7. Calcular la media muestral de valores usando np.mean(valores) y comprar con la media teórica (11/3).

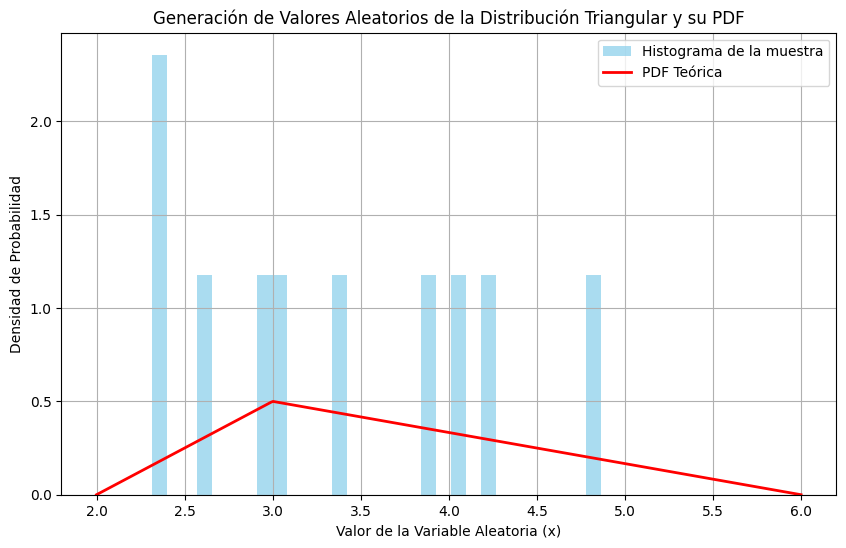
### Gráficas de la función de densidad

A partir del modelo implementado se puede graficar en un histograma la distribución de los valores generados por el algoritmo para la distribución triangular. Para ello se utilizó Python, usando el mismo modelo anteriormente desarrollado, y utilizando la herramienta Matplotlib para graficar un histograma de las variables generadas y compararla con la función de densidad dada (triangular). El código para elaborar estas gráficas con el módulo mencionado se encuentra en el siguiente Google Colab:

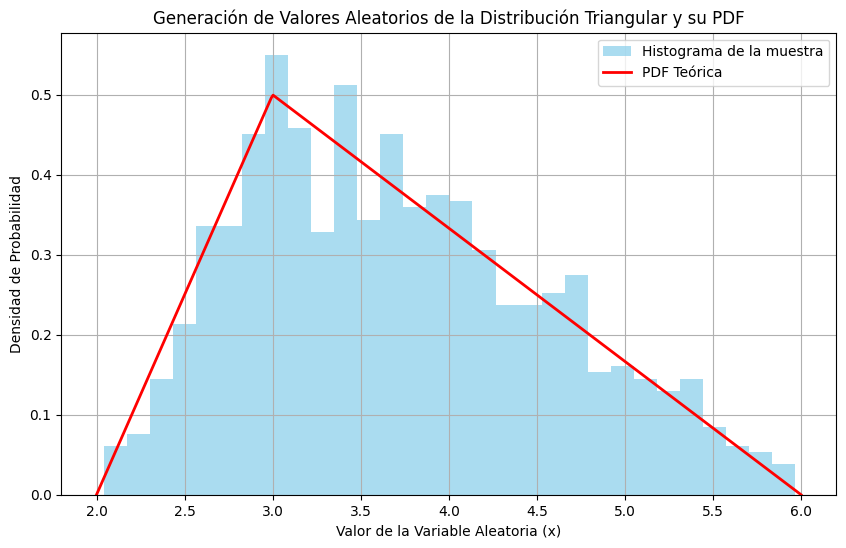
[Generacion de variables aleatorias.ipynb](https://colab.research.google.com/drive/18YD8dXChf4wN2HT4ZXNeNt4yEcGXhjQ8#scrollTo=8iEMfsfJqI-J)

Con el fin de mantener la reproducibilidad del código se ha empleado una semilla para la librería random, de modo que los resultados siempre son los mismos.

Inicialmente se elaboró la gráfica con los 10 valores iniciales, proporcionados por el algoritmo anterior, obteniéndose el siguiente resultado:



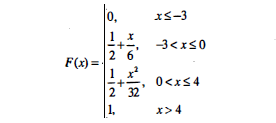
Como se puede observar, la distribución no parece ser triangular, pues el histograma no se ajusta bien a la función de densidad triangular del ejercicio. Esto se puede explicar por la cantidad de valores generados, puesto que 10 números aleatorios representa una muestra muy pequeña para acercarse a la distribución esperada (triangular). Por ello, se elaboró una nueva gráfica con más datos, en específico 1000 valores generados por la variable aleatoria, obteniéndose los siguientes resultados:



En esta ocasión se puede observar un mejor acercamiento a la distribución por parte del histograma, generado a partir de los valores aleatorios generados. Esto se debe a que con una mayor cantidad de datos la distribución aplicada se observa de mejor manera, y también va de la mano con la ley de los grandes números, haciendo que la media muestral se acerque cada vez más a la media teórica.

## Ejercicio 4

Dada la siguiente cdf para una variable aleatoria continua con rango desde – 3 a 4, desarrolle un generador para la variable.



**Marco teórico**

**Distribución uniforme a trozos**

Es una variable aleatoria continua cuya densidad es constante dentro de cada subintervalo de su soporte, pero puede diferir de un tramo a otro.

En este caso



densidad “uniforme” por tramos, con distinta “altura” en cada uno

**Función de densidad de probabilidad (PDF) y función de distribución acumulada (CDF)**

La **PDF**, f(x), cuantifica la probabilidad relativa de que X tome un valor cercano a x.

La CDF, da la probabilidad de que la variable sea menor o igual a 𝑥.

Se verifica siempre

1. 

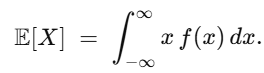
𝐹 crece monótonamente.

**Método de la transformada inversa**

* Permite generar X con una CDF dada 𝐹
* si  entonces sigue la distribución deseada
* Pasos

1. Generar U ∈ (0,1).
2. Determinar en qué tramo de F cae U.
3. Calcular X resolviendo U=F(x) para x.

**Cálculo del valor esperado (media teórica)**

Para variable continua X con PDF f(x), el valor esperado es

## 

## 

## Representa el “centro de gravedad” de la distribución de probabilidad

**Media muestral y Ley de los Grandes Números**

Dada una muestra {X1​,…,Xn​}, la media muestral es



La Ley de los Grandes Números afirma que Xˉ→E[X] cuando n→∞

**Herramientas digitales**

**Python:** lenguaje de alto nivel, muy usado en simulación y ciencia de datos

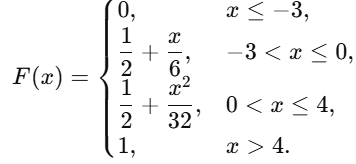
**Google Colaboratory**: entorno Jupyter en la nube, facilita la ejecución de código sin instalar nada localmente

**NumPy** y **módulo random**:

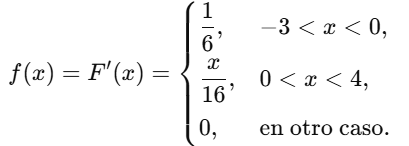
* *NumPy* provee estructuras y operaciones eficientes sobre arreglos [8].
* *random.random()* genera )U∼(0,1).

**Matplotlib**: biblioteca estándar para graficar funciones de densidad y histogramas en Python.

## Desarrollo del algoritmo

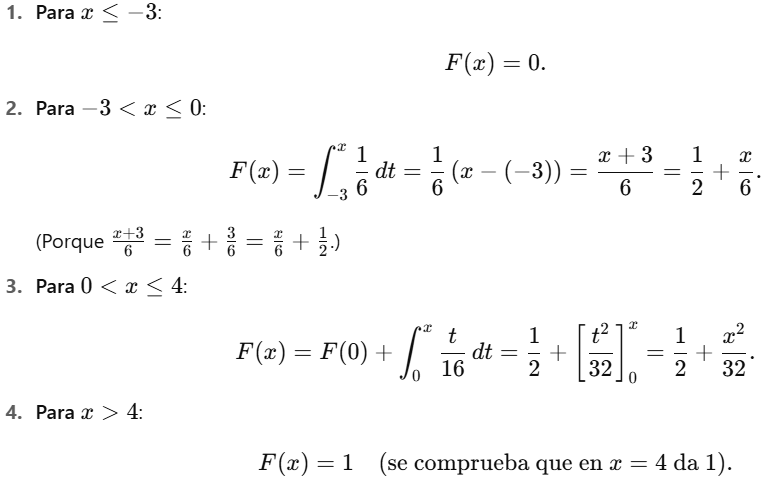
Partimos de la CDF que nos daban y su derivada:

Por tanto la **PDF** es:

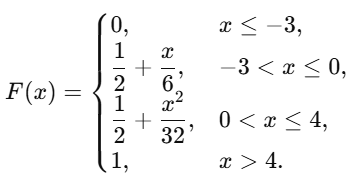


**Calcular la CDF F(x) integrando f(x)**

Para verificar y “re-derivar” F(x) a partir de f(x):

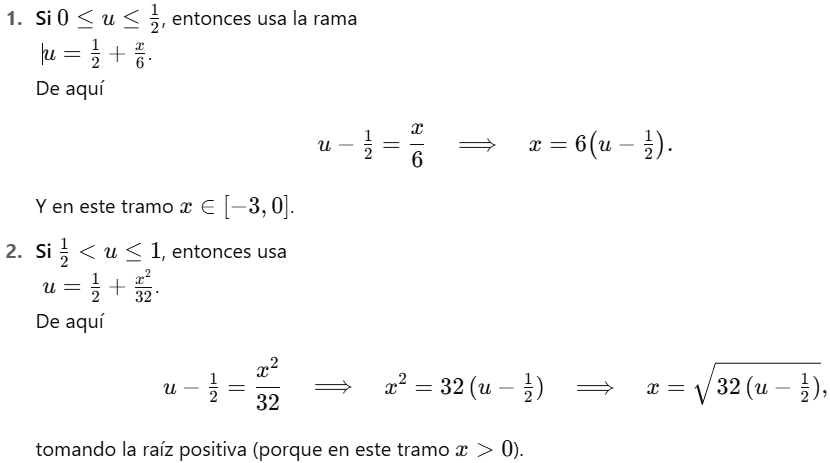


De este modo obtenemos de nuevo

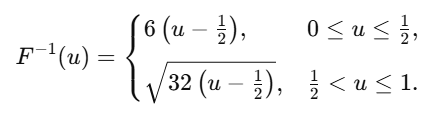


Invertir la CDF: hallar F−1(u)

Sea U∼U(0,1). Queremos X = F^{-1}(U). Dividimos en dos casos:



Por tanto,



**Algoritmo en Python**

El código completo de generación de la variable aleatoria para nuestro problema está disponible en el Google Colab: [Generacion de variables aleatorias.ipynb](https://colab.research.google.com/drive/18YD8dXChf4wN2HT4ZXNeNt4yEcGXhjQ8#scrollTo=TpQiU-iMqUL9)

**Sacar un número al azar** Cada vez que queremos un valor nuevo, primero extraemos un número completamente aleatorio entre 0 y 1.

Su funcionalidad es de este manera:

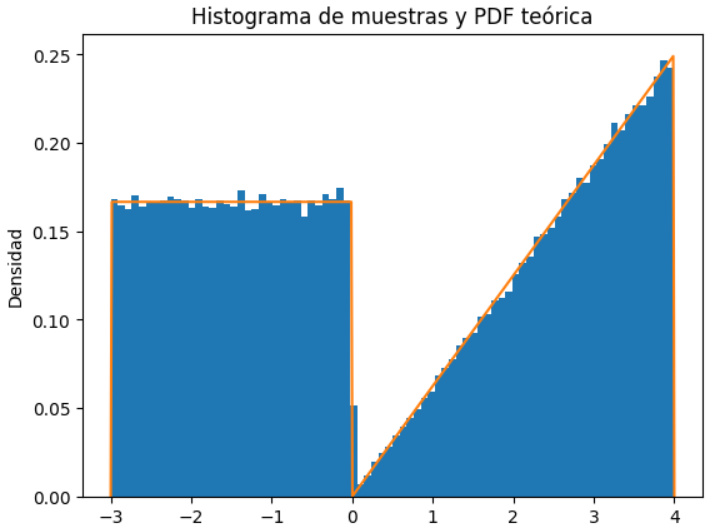
**Decidir en qué tramo caer** Dividimos ese intervalo en dos mitades. Si nuestro número aleatorio cae en la primera mitad, lo asignamos al tramo “izquierdo” de la distribución; si cae en la segunda mitad, lo asignamos al tramo “derecho”.

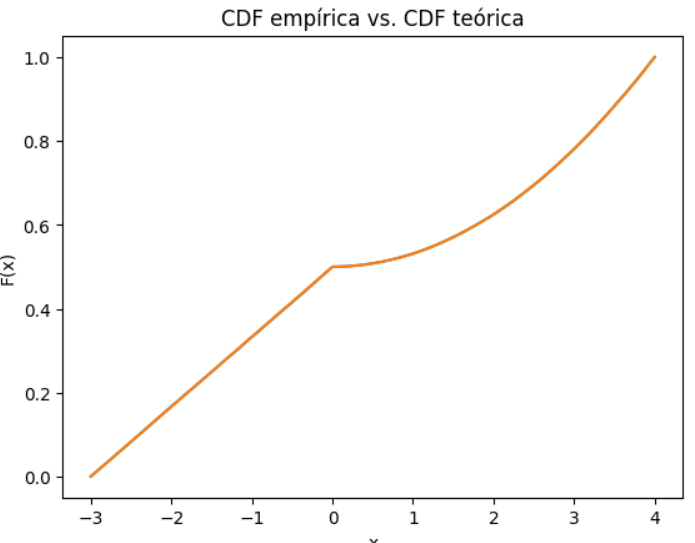
**Transformar según el tramo** Dependiendo de cuál de las dos mitades haya tocado, aplicamos una transformación distinta para llevar ese número uniforme a un valor que siga la forma deseada en la zona correspondiente.

**Repetir para obtener muestra** Repetimos estos tres pasos tantas veces como los valores necesitemos. Así vamos construyendo una colección de datos que, en conjunto, reproduce la “forma” de nuestra distribución.

**Verificar resultados** Con esa colección de valores podemos calcular su promedio y dibujar gráficos que muestran cómo, al aumentar la cantidad de datos, el comportamiento de la muestra converge a la distribución teórica esperada.

### Gráficas de la función de densidad





**Histograma y curva de densidad**

**Barras (histograma)**: cada barra muestra cuántos valores de nuestra muestra cayeron en un pequeño intervalo de la línea. Con pocas muestras las barras van muy arriba y abajo, pero a medida que recogemos más datos el perfil de las barras empieza a adquirir una forma suave.

**Línea (densidad teórica)**: es la curva que esperábamos de antemano, con dos tramos bien definidos—uno plano y otro que crece—y cuya área total suma 1.

**Interpretación:** cuando el número de muestras es grande, el contorno formado por las barras queda muy cerca de esa línea, lo que indica que la frecuencia con que nuestros valores caen en cada zona coincide con la “altura” de la densidad que buscábamos.

**Curva escalonada (ECDF) y CDF teórica**

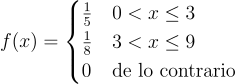
**Escalones (distribución empírica):** ordenamos todos los valores y, por cada uno, subimos un peldaño proporcional al tamaño de la muestra. El resultado es una línea por tramos que va de 0 hasta 1.

**Línea suave (CDF teórica)**: es la función acumulada que esperábamos, que se queda en 0 hasta el inicio del soporte, crece de forma lineal en el primer tramo y luego de forma cuadrática en el segundo, hasta llegar a 1.

**Interpretación:** al principio, con pocos datos, los escalones son grandes y la curva empírica está muy alejada de la teórica. Conforme agregamos más muestras, los peldaños se vuelven más pequeños y la curva escalonada se ajusta casi perfectamente a la línea suave, mostrando que la proporción acumulada de valores coincide con lo que dicta nuestra función de distribución.

## Ejercicio 5

Desarrolle el generador para la variable aleatoria con pdf dado por



### Marco Teórico

Los conceptos y métodos utilizados en el desarrollo del algoritmo fueron los siguientes:

* **Distribución Uniforme a Trozos:** Es una variable aleatoria en la cual la probabilidad es uniforme dentro de cada segmento del rango de la variable aleatoria, pero la altura de la PDF puede cambiar entre segmentos [11]. Es la naturaleza de la variable aleatoria dada por la función de densidad de probabilidad dada por el enunciado del ejercicio.
* **Función de Densidad de Probabilidad y Función de Distribución Acumulada:** La función de densidad de probabilidad, también conocida como PDF y denotada *f(x)* describe la probabilidad relativa de una variable continua tomando un valor específico; la función de distribución acumulada, también conocida como CDF y denotada *F(x)* da la probabilidad de que la variable sea menor o igual a x. [2]
* **Método de la Transformada Inversa:** Es una técnical para generar números aleatorios que siguen una distribución de probabilidad específica, dada su función de distribución acumulativa (CDF). El método se basa en el principio de que si U es una variable aleatoria que sigue una distribución uniforme en el intervalo (0,1), entonces la variable aleatoria X=F−1(U) tendrá la misma distribución que la función de distribución acumulativa F, donde F−1 es la función inversa de la CDF. [3]

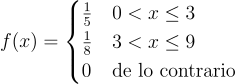
Los pasos a seguir son los siguientes:

1. Encontrar la función de distribución acumulada *F(x)* de la distribución deseada.
2. Obtener la inversa de la CDF,; se hace y se resuelve para en términos de .
3. Generar un número aleatorio u a partir de una distribución uniforme U(0,1).
4. El valor tendrá la distribución deseada.

* **Media de una distribución continua:** También conocido como valor esperado, y denotada , es el valor promedio de X, una variable aleatoria continua con función de densidad de probabilidad f(x), ponderado por su probabilidad de ocurrencia. Representa el centro de la distribución de probabilidad y matemáticamente se define así: [4]
* **Media muestral:** La media muestral (barx) es el promedio de una muestra de datos. La *Ley de los Grandes Números* indica que a medida que el tamaño de la muestra aumenta, la media muestral se acerca a la media de la población, o media teórica en el caso de una distribución de probabilidad continua. [5]
* **Herramientas digitales:**
  + **Python:** Es un lenguaje de programación de alto nivel, versátil y ampliamente utilizado en la ciencia de datos, la estadística y la simulación. [6] Fue el lenguaje de programación utilizado para implementar el algoritmo.
  + **Google Colaboratory (Colab):** Es un entorno de cuaderno Jupyter gratuito basado en la nube que permite escribir y ejecutar código Python a través de un navegador web. [7] Fue el ambiente de desarrollo empleado para implementar el algoritmo.
  + **NumPy y Random:** La primera es una biblioteca fundamental para la computación numérica en Python [8]; y la segunda es un módulo incorporado que implementa generadores de números pseudoaleatorios para diversas distribuciones [9]. Fueron las importaciones utilizadas para implementar el algoritmo, usando random.random() para obtener números aleatorios según una distribución uniforme U(0, 1); y usando np.mean() para calcular la media muestral de los números obtenidos.
  + **Matplotlib:** es un módulo dentro de la biblioteca Matplotlib en Python que proporciona una interfaz similar a la de MATLAB para la creación de gráficos y visualizaciones es un módulo dentro de la biblioteca Matplotlib en Python que proporciona una interfaz similar a la de MATLAB para la creación de gráficos y visualizaciones [10]. Se ha empleado para la generación de las gráficas de función de densidad y el histograma de los valores generados por la variable aleatoria.

### Desarrollo de algoritmo

La función de densidad de probabilidad dada es la siguiente:



Para generar valores aleatorios a partir de esta distribución, utilizaremos el método de la transformada inversa, a partir de los siguientes pasos:

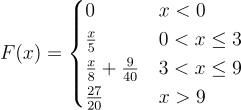
1. **Encontrar la :**

La función de distribución acumulada se obtiene integrando la función de densidad de probabilidad, de la siguiente manera:

* Para :
* Para :
  + Primero evaluamos :
  + Ahora, integramos la segunda parte:

Luego se tiene que

La función de distribución acumulada completa es:



Acá es importante resaltar que la función de distribución acumulada sobrepasa el valor de 1 siguiendo esta forma, lo cuál se debe a un problema en el planteamiento de la función de distribución uniforme a trozos del enunciado del ejercicio.

Dado que este enunciado es fijo, se procedió con el desarrollo del generador de variable aleatoria con esta función de distribución acumulada.

1. **Invertir (encontrar ):**

Sea , donde . Necesitamos resolver para en términos de .

* Para :
* Para :

La función inversa de la función de distribución acumulada es:



1. **Media de la distribución:**

La media teórica de la distribución dada es:

* Primer integral:
* Segunda integral:

La media de la distribución es entonces:

1. **Algoritmo en Python**

El código del algoritmo implementado se puede encontrar en el siguiente Google Colab:

[Generacion de variables aleatorias.ipynb](https://colab.research.google.com/drive/18YD8dXChf4wN2HT4ZXNeNt4yEcGXhjQ8#scrollTo=EdDvBHIU64FC&line=1&uniqifier=1)

La funcionalidad se explica a continuación:

1. Crear una lista llamada valores
2. Para i desde 1 hasta 10:
3. Generar un valor aleatorio según una distribución uniforme usando random.random().
4. Si : Calcular .
5. Si : Calcular .
6. Agregar el valor de a valores.
7. Calcular la media muestral de valores usando np.mean(valores) y comprar con la media teórica (54/10).

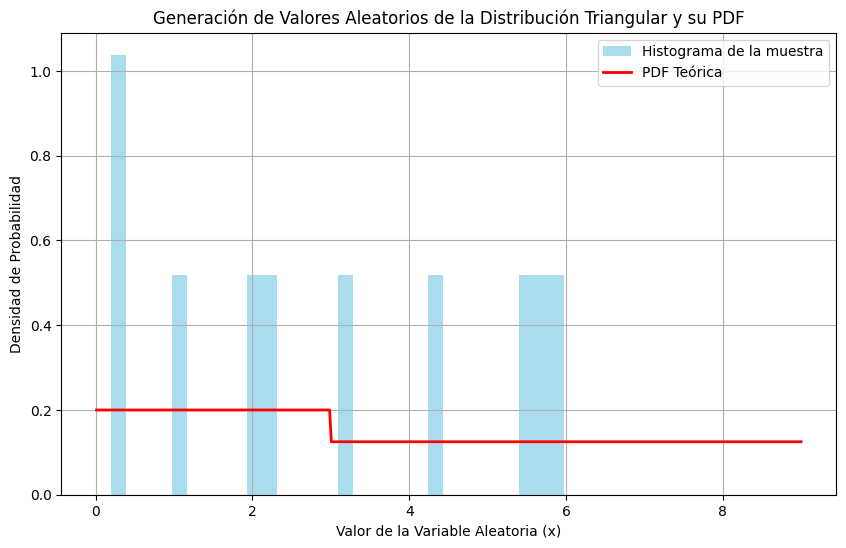
### Gráficas de la función de densidad

A partir del modelo implementado se puede graficar en un histograma la distribución de los valores generados por el algoritmo para la distribución triangular. Para ello se utilizó Python, usando el mismo modelo anteriormente desarrollado, y utilizando la herramienta Matplotlib para graficar un histograma de las variables generadas y compararla con la función de densidad dada (triangular). El código para elaborar estas gráficas con el módulo mencionado se encuentra en el siguiente Google Colab:

[Generacion de variables aleatorias.ipynb](https://colab.research.google.com/drive/18YD8dXChf4wN2HT4ZXNeNt4yEcGXhjQ8#scrollTo=EdDvBHIU64FC&line=1&uniqifier=1)

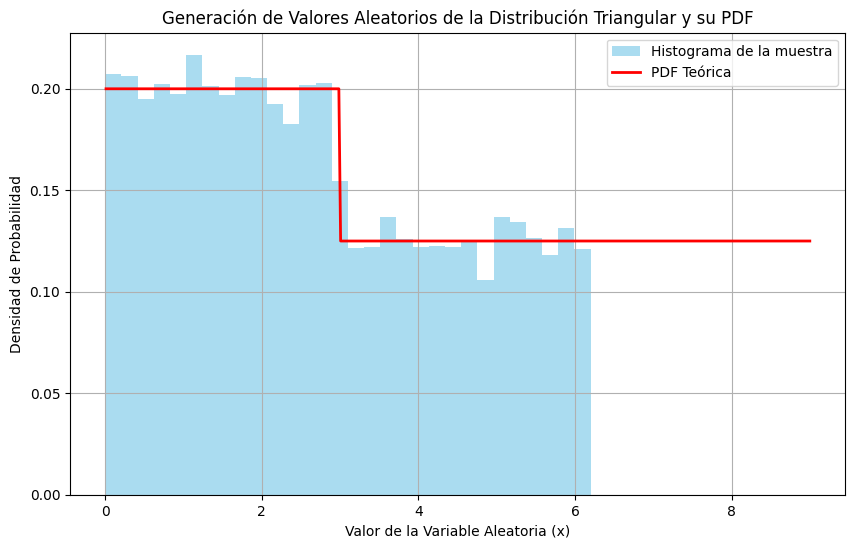
Con el fin de mantener la reproducibilidad del código se ha empleado una semilla para la librería random, de modo que los resultados siempre son los mismos.

Inicialmente se elaboró la gráfica con los 10 valores iniciales, proporcionados por el algoritmo anterior, obteniéndose el siguiente resultado:



Como se puede observar, la distribución no parece seguir la función de distribución dada, pues el histograma no se ajusta bien a la función de densidad del ejercicio. Esto se puede explicar por la cantidad de valores generados, puesto que 10 números aleatorios representa una muestra muy pequeña para acercarse a la distribución esperada (triangular). Sin embargo, también se puede notar que no hay valores más allá del 6, lo cuál se debe a que, como se mencionó anteriormente, hay un problema en el planteamiento de la función de densidad en el enunciado del ejercicio que hace que la función de distribución acumulada sobrepase el límite del 1; y como la variable aleatoria uniforme utilizada, solo genera datos tan grandes como uno, no es posible generar valores más allá del 6, el cual parece ser el valor para que la función de distribución acumulada sea 1.

Con el fin de verificar si la generación de los datos sigue la distribución dada se elaboró una nueva gráfica con más datos, en específico 10000 valores generados por la variable aleatoria, obteniéndose los siguientes resultados:



En esta ocasión se puede observar un mejor acercamiento a la distribución por parte del histograma, generado a partir de los valores aleatorios generados. Esto se debe a que con una mayor cantidad de datos la distribución aplicada se observa de mejor manera, y también va de la mano con la ley de los grandes números, haciendo que la media muestral se acerque cada vez más a la media teórica. Una vez más, es importante resaltar que no hay números aleatorios después de un valor un poco más grande que 6, dado que la variable aleatoria uniforme solamente proporciona números tan grandes como 1, por lo que se vuelve a notar el problema en el planteamiento de la función de distribución del enunciado del ejercicio.

# Problemas asignados al grupo

## Problema 1.3

Este problema consiste en en lo siguiente:

Considere una instalación de servicio conformada de dos servidores tipo A y un servidor tipo B. Asuma que los clientes llegan a la instalación con tiempos entre llegadas que son variables aleatorias exponenciales IID con media de 1 minuto. Al llegar, se determina tipo de cliente 1 o 2 con probabilidades respectivas de 0.7 y 0.30. Un cliente tipo 1 puede ser atendido por cualquier servidor, pero selecciona un servidor tipo A si uno está disponible. Los tiempos de servicio para los clientes tipo 1 son variables aleatorias exponenciales IID con media 0.8 minutos, independiente del servidor utilizado. Los clientes tipo 1 que encuentran todos los servidores ocupados se unen a una cola FIFO para clientes tipo 1. Un cliente tipo 2 requiere servicio simultáneamente de un servidor tipo A y un servidor tipo B. Los tiempos de servicio para clientes tipo 2 se distribuye uniformemente entre 0.5 y 0.7 minutos. Los clientes tipo 2 que al llegar encuentran ambos servidores tipo A ocupados o el servidor tipo B ocupado se unen a una cola FIFO para clientes tipo 2. Al terminar servicio a cualquier cliente, la preferencia se da a un cliente tipo 2 si uno está presente y si un servidor tipo A y el servidor tipo B están libres. De lo contrario, se atiende un cliente tipo 1 si lo hay. Simular la instalación para exactamente 1000 minutos, y evaluar la demora promedio en la cola y el número promedio en cola para cada tipo de cliente. Así mismo evaluó la proporción de tiempo que cada servidor dedica a cada tipo de cliente.

**Planteamiento**

El problema presentado se trata de una instalación de servicio con dos filas y tres servidores: dos del tipo A y uno del tipo B. Los clientes que llegan a esta instalación pueden ser de dos tipos: tipo 1 y tipo 2.

* Los clientes llegan de forma individual y aleatoria, siguiendo una distribución exponencial con media de 1 minuto entre llegadas.
* Al llegar, cada cliente se clasifica como:
  + Tipo 1 con probabilidad 0.7
  + Tipo 2 con probabilidad 0.3
* **Clientes tipo 1**:
* Pueden ser atendidos por cualquier servidor.
* Prefieren un servidor tipo A, si alguno está disponible.
* Si todos los servidores están ocupados, se unen a una cola FIFO exclusiva para clientes tipo 1.
* Su tiempo de servicio se distribuye exponencialmente con una media de 0.8 minutos.
* **Clientes tipo 2:**
* Requieren ser atendidos simultáneamente por un servidor tipo A y el servidor tipo B.
* Si al llegar no hay al menos un servidor A y el servidor B disponible, se unen a una cola FIFO exclusiva para clientes tipo 2.
* Su tiempo de servicio se distribuye uniformemente entre 0.5 y 0.7 minutos.
* **Política de prioridad:**
* Cuando un servidor queda libre, se da prioridad a los clientes tipo 2, siempre que se puedan asignar simultáneamente un servidor A y el B.
* Si no hay condiciones para atender a un cliente tipo 2, entonces se atiende a un cliente tipo 1 (si hay en cola).
* El sistema se debe simular durante exactamente 1000 minutos.

**Definición del Modelo por Eventos**

1. **Variables de Estado del Sistema**

Estas variables describen el estado de la instalación en un instante dado durante la simulación. Su valor se actualiza a medida que ocurren eventos (como llegadas, inicios de servicio y finalizaciones de servicio).

* 1. **Variables generales**

**tiempo\_actual:** Número que representa el instante actual en la simulación (en minutos). Avanza con cada evento procesado.

**eventos\_futuros**: Una cola priorizada que contiene los eventos pendientes, ordenados por el tiempo en que ocurren (p. ej., llegadas, finalización de servicio).

* 1. **Variables relacionadas con las colas**

**cola tipo1**: Lista que almacena los tiempos de llegada (u objetos cliente) de los clientes tipo 1 en espera. Política FIFO.

**cola\_tipo2**: Lista que almacena los tiempos de llegada (u objetos cliente) de los clientes tipo 2 en espera. Política FIFO.

* 1. **Estado de los servidores**

**servidores\_A**: Lista de longitud 2, cada elemento puede ser:

* **None** si el servidor A está libre
* **Un objeto cliente** (con tipo y tiempo de finalización) si está ocupado

**servidor\_B**: Variable que puede ser:

* **None** si el servidor B está libre
* **Un objeto cliente** si está ocupado
  1. **Información del servicio actual**

**clientes\_en\_servicio:** Lista con detalles de los clientes actualmente siendo atendidos (tipo, inicio y fin de servicio, servidores asignados).

**tiempos\_fin\_servicio**: Lista de eventos próximos de fin de servicio para cada cliente en servicio.

**uso\_servidores**: Diccionario que lleva el registro de cuánto tiempo cada servidor A o B ha pasado atendiendo clientes tipo 1 o tipo 2.

uso\_servidores = {

'A1': {'tipo1': 0.0, 'tipo2': 0.0},

'A2': {'tipo1': 0.0, 'tipo2': 0.0},

'B': {'tipo1': 0.0, 'tipo2': 0.0},

}

* 1. **Estadísticas acumuladas**

**tiempo\_espera\_tipo1**: Lista de tiempos de espera individuales para clientes tipo 1

**tiempo\_espera\_tipo2**: Lista de tiempos de espera individuales para clientes tipo 2

**area\_cola\_tipo1**: Acumulador del área bajo la curva de longitud de la cola tipo 1 (para promedio de largo de cola).

**area\_cola\_tipo2**: Lo mismo, pero para la cola tipo 2.

**ult\_actualizacion\_cola**: Último tiempo en que se actualizó el conteo de clientes en cada cola.

Estas variables son la base del sistema de eventos. A partir de aquí, puedes definir:

* Los tipos de eventos (llegada tipo 1, llegada tipo 2, fin de servicio tipo 1, fin de servicio tipo 2).
* El manejador de eventos, que actualiza estas variables cuando un evento ocurre.

1. **Entidades y sus Atributos**

Las entidades son los objetos que fluyen por el sistema (clientes), y que requieren recursos para ser procesados (los servidores). Los atributos describen características individuales de cada entidad y determinan cómo interactúan con el sistema.

1. **Cliente Tipo 1**

**Tiempo de llegada:** Instante en que el cliente tipo 1 ingresa al sistema. Se utiliza para calcular su tiempo de espera en cola. Se almacena en cola\_tipo1.

**Tiempo de servicio requerido:** Generado aleatoriamente al momento de iniciar el servicio, siguiendo una distribución exponencial con media de 0.8 minutos.

**Servidor asignado:** Puede ser cualquiera de los servidores tipo A (preferido) o tipo B (si los A están ocupados).

1. **Cliente Tipo 2**

**Tiempo de llegada:** Instante en que el cliente tipo 2 llega al sistema. Se utiliza para calcular el tiempo de espera. Se almacena en cola\_tipo2.

**Tiempo de servicio requerido:** Generado aleatoriamente al inicio del servicio, siguiendo una distribución uniforme entre 0.5 y 0.7 minutos.

**Servidores asignados:** Requiere simultáneamente un servidor tipo A y el servidor tipo B.

1. **Servidor Tipo A (A1 y A2)**

**Estado:** Ocupado o libre.

**Cliente actual:** Referencia al cliente que está siendo atendido (tipo, inicio y fin de servicio).

**Tiempo de servicio acumulado por tipo:** Registra cuánto tiempo ha pasado atendiendo clientes tipo 1 o tipo 2, para fines estadísticos.

**ID del servidor:** Por ejemplo, "A1" o "A2".

1. **Servidor Tipo B**

**Estado:** Ocupado o libre.

**Cliente actual:** Referencia al cliente tipo 2 que está siendo atendido.

**Tiempo de servicio acumulado por tipo:** Solo acumula tiempo para clientes tipo 2 (ya que tipo 1 no lo usa).

**ID del servidor:** "B"

1. **Eventos**

Los eventos son los puntos en el tiempo en los que el estado del sistema puede cambiar (por ejemplo, llega un cliente o termina un servicio). La simulación se basa en una lista de eventos futuros, que se procesa en orden cronológico.

1. **Llegada de un nuevo cliente (evento: llegada\_cliente)**

* Ocurre cuando un nuevo cliente llega al sistema.
* Se determina aleatoriamente si es tipo 1 (probabilidad 0.7) o tipo 2 (probabilidad 0.3).
* Dependiendo del tipo, se evalúan las siguientes condiciones:

**Si es tipo 1:**

* Si hay algún servidor tipo A libre, se inicia servicio inmediatamente con él.
* Si no hay A libres, pero el servidor B está libre, se inicia servicio con el servidor B.
* Si todos los servidores están ocupados, el cliente se coloca en la cola tipo 1 (FIFO).

**Si es tipo 2:**

* Si hay al menos un servidor A libre y el servidor B libre, se inicia servicio con ambos.
* Si no, el cliente se coloca en la cola tipo 2 (FIFO).
* Se programa la siguiente llegada generando un nuevo tiempo de interarribo (exponencial con media 1 minuto).

1. **Fin de servicio (evento: fin\_servicio)**

* Ocurre cuando termina el servicio a un cliente (tipo 1 o tipo 2).
* El servidor (o servidores) que estaban ocupados se marcan como libres.
* Se registra el tiempo de servicio acumulado por tipo de cliente y servidor.
* Se evalúa si hay clientes esperando en alguna cola:

**Prioridad a tipo 2:**

* Si la cola tipo 2 no está vacía y hay al menos un servidor A libre y B libre, se inicia servicio con ambos a un cliente tipo 2.

**Si no es posible atender tipo 2:**

* Si la cola tipo 1 no está vacía y al menos un servidor A o B está libre, se inicia servicio con el primero disponible a un cliente tipo 1.

1. **Contadores y Acumuladores**

Estas variables recopilan estadísticas clave sobre el rendimiento del sistema durante los 1000 minutos de simulación. Se actualizan a medida que los eventos se procesan.

1. **Clientes Tipo 1**

**total\_espera\_tipo1:** Acumulador del tiempo total de espera en cola de todos los clientes tipo 1.

**clientes\_atendidos\_tipo1:** Contador del número total de clientes tipo 1 que fueron atendidos.

**area\_bajo\_cola\_tipo1**: Acumulador del área bajo la curva del número de clientes en cola tipo 1. Se usa para calcular el promedio de clientes en cola.

**tiempo\_ultimo\_evento\_tipo1**: Marca de tiempo del último evento que afectó la longitud de la cola tipo 1 (para el cálculo incremental del área).

1. **Clientes Tipo 2**

**total\_espera\_tipo2**: Acumulador del tiempo total de espera en cola de todos los clientes tipo 2.

**clientes\_atendidos\_tipo2**: Contador del número total de clientes tipo 2 que fueron atendidos.

**area\_bajo\_cola\_tipo2:** Acumulador del área bajo la curva del número de clientes en cola tipo 2.

**tiempo\_ultimo\_evento\_tipo2**: Marca de tiempo del último evento que afectó la longitud de la cola tipo 2.

1. **Servidores**

**tiempo\_ocupado\_A1\_tipo1, tiempo\_ocupado\_A1\_tipo2**: Tiempo acumulado que el servidor A1 pasó atendiendo clientes tipo 1 y tipo 2, respectivamente.

**tiempo\_ocupado\_A2\_tipo1, tiempo\_ocupado\_A2\_tipo2**: Igual que A1, pero para el servidor A2.

**tiempo\_ocupado\_B\_tipo2**: Tiempo acumulado que el servidor B pasó atendiendo clientes tipo 2 (no atiende clientes tipo 1).

También puede manejarse mediante un diccionario general:

uso\_servidores = {

'A1': {'tipo1': 0.0, 'tipo2': 0.0},

'A2': {'tipo1': 0.0, 'tipo2': 0.0},

'B': {'tipo2': 0.0}

}

Estos contadores y acumuladores permitirán al final de la simulación calcular:

* Tiempo promedio de espera en cola por tipo de cliente.
* Número promedio en cola por tipo de cliente.
* Proporción de utilización de cada servidor por tipo de cliente.

1. **Medidas de Desempeño**

Al finalizar los 1000 minutos de simulación, se calculan las siguientes métricas para evaluar el rendimiento del sistema. Estas medidas se obtienen usando los contadores y acumuladores definidos previamente.

1. **Tiempo de espera promedio en cola por tipo de cliente**

**Clientes tipo 1:**

**tiempo\_espera\_promedio\_tipo1 =**

**clientes\_atendidos\_tipo1 / total\_espera\_tipo1**

Indica el tiempo promedio que un cliente tipo 1 pasa esperando en la cola antes de iniciar su servicio.

**Clientes tipo 2:**

**tiempo\_espera\_promedio\_tipo2 =   
clientes\_atendidos\_tipo2 / total\_espera\_tipo2**

Indica el tiempo promedio que un cliente tipo 2 permanece en la cola hasta que estén disponibles simultáneamente un servidor A y el servidor B.

1. **Número promedio de clientes en cola por tipo**

**Clientes tipo 1:**

**promedio\_cola\_tipo1 = tiempo\_simulacion / area\_bajo\_cola\_tipo1**

**Clientes tipo 2:**

**promedio\_cola\_tipo2= area\_bajo\_cola\_tipo2 / tiempo\_simulacion**

Estas métricas indican cuántos clientes, en promedio, están esperando en la cola en cualquier instante.

1. **Proporción de tiempo ocupado de cada servidor por tipo de cliente**

Para cada servidor (A1, A2, B), se calcula la proporción del tiempo total que estuvo ocupado por cada tipo de cliente:

* **Por ejemplo, para el servidor A1:**

**uso\_A1\_tipo1 = tiempo\_ocupado\_A1\_tipo1​ / tiempo\_simulacion**

**uso\_A1\_tipo2 = tiempo\_ocupado\_A1\_tipo2 / tiempo\_simulacion**

* **Para el servidor B (solo clientes tipo 2):**

**uso\_B\_tipo2 = tiempo\_ocupado\_B\_tipo2​ / tiempo\_simulacion**

Esto permite entender la carga de trabajo de cada servidor y cómo se distribuye entre los tipos de clientes.

**Diagramas de flujo**

**Los diagramas de flujo están en este PDF:** [**Diagramas de flujo problema 1.3.pdf**](https://drive.google.com/file/d/12wIPVTsDo_Lf0hWEbzuKC0JF-UvYSpjP/view?usp=drive_link) **hacer zoom para verlos más de cerca.**

**Desarrollo del simulador**

El código del algoritmo implementado se puede encontrar en el siguiente Google Colab:[Codigo problema 1.3.ipynb](https://colab.research.google.com/drive/1mefByHena7TsvZVyRhmD-AYxNyElhx6q?usp=drive_link)

**​**

**Análisis de resultados**

**Resultados simulación instalación de servicio (semilla 44)**

**Tiempo total de simulación: 1000 minutos**

**Medias de las distribuciones (minutos):**

* Tiempo entre llegadas de clientes: 1.0
* Tiempo de servicio clientes tipo 1: 0.8 (exponencial)
* Tiempo de servicio clientes tipo 2: Uniforme entre 0.5 y 0.7

**Indicadores de Desempeño:**

* Tiempo de espera promedio clientes tipo 1: 0.113 minutos
* Tiempo de espera promedio clientes tipo 2: 0.087 minutos
* Número promedio en cola clientes tipo 1: 0.083
* Número promedio en cola clientes tipo 2: 0.026

**Total de clientes atendidos:**

* Clientes tipo 1: 687
* Clientes tipo 2: 305

**Proporción de tiempo que cada servidor dedicó a cada tipo de cliente:**

**Servidor A1:**

* Tipo 1: 33.86%
* Tipo 2: 13.31%

**Servidor A2:**

* Tipo 1: 20.53%
* Tipo 2: 5.13%

**Servidor B:**

* Tipo 2: 18.44%

**Análisis:**

Los resultados muestran que ambos tipos de clientes experimentan tiempos de espera muy bajos, lo que indica que la instalación tiene suficiente capacidad para manejar la carga de trabajo durante el periodo simulado. Los clientes tipo 2, a pesar de requerir simultáneamente dos servidores (uno tipo A y el tipo B), tienen un tiempo de espera promedio incluso más bajo que los tipo 1. Esto se debe a que:

1. **Política de prioridad:** Cuando hay disponibilidad simultánea de un servidor A y del B, se da prioridad a los clientes tipo 2, lo cual les permite ingresar más rápido al sistema cuando hay oportunidad.
2. **Tiempo de servicio**: Aunque los clientes tipo 2 ocupan más recursos, su tiempo de servicio es corto (máx. 0.7 minutos), y esto ayuda a liberar recursos rápidamente.
3. **Uso de los servidores**: Se observa que el servidor A1 trabajó más que el A2, lo que sugiere que el sistema no balancea perfectamente la carga entre ambos servidores A. El servidor B solo atiende clientes tipo 2, con una utilización del 18.44%, indicando que hay suficiente capacidad ociosa para evitar cuellos de botella.

**​Conclusión**

La instalación opera eficientemente, con baja congestión en las colas y un buen aprovechamiento de los recursos. El sistema de prioridades favorece correctamente a los clientes tipo 2 cuando es posible, sin afectar significativamente la experiencia de los clientes tipo 1. Esto demuestra que el diseño de la política de atención y la cantidad de servidores es adecuado para las tasas de llegada y servicio propuestas. Para evaluar escenarios más exigentes, podría analizarse el sistema con una tasa de llegada mayor o con menos servidores tipo A.

## Problema 1.4

### Identificación del problema

La situación presentada es la siguiente:

*Un teatro utiliza un empleado para vender tiquetes y responder consultas desde las 9 a. m hasta las 5 p. m. Los puestos se adjudican únicamente si el cliente llega al teatro y paga por los tiquetes. Consultas provienen de clientes en persona o de llamadas al teatro y El empleado da prioridad a los clientes en persona. Sin embargo, gracias a un sistema complejo telefónico, las llamadas pueden esperar para ser atendidos según la política FIFO (primero en llegar primero en salir) y no renuncian hasta obtener una respuesta. Los clientes en persona llegan según una distribución exponencial con media de 12 minutos y su tiempo de servicio se distribuye exponencial con media de 6 minutos. Las llamadas ocurren según una distribución exponencial con media de 10 minutos y su tiempo de servicio sigue una distribución exponencial con media de 5 minutos. La primera persona llega a los 2 minutos y la primera llamada a los 3 minutos. Simular este sistema para un día de 8 horas y obtenga el tiempo de espera promedio de cada tipo de cliente.*

### Planteamiento

El problema presentado se trata sobre dos filas (clientes en persona y llamadas telefónicas) de cliente con un único empleado atentiendo, en orden FIFO (primero en llegar primero en salir) para cada fila, pero dándole prioridad a los clientes en persona sobre las llamadas telefónicas. Cada fila tiene llegadas que siguen una distribución exponencial con medias distintas (12 minutos para los clientes en persona y 10 minutos para las llamadas telefónicas) y tiempos de servicio que siguen una distribución exponenicla con medias distintas (6 minutos para los clientes en persona y 5 minutos para las llamadas telefónicas).

### Definición del modelo por eventos

Vamos a realizar un modelo del problema planteado usando la simulación por eventos; para ello comenzaremos definiendo algunas partes importantes:

#### Variables de Estado del Sistema

Son variables que describen la condición actual del sistema en un momento dado durante la simulación. Su valor cambia a medida que ocurren los eventos.

* **tiempo\_actual**: Representa el instante de tiempo actual en la simulación. Avanza a medida que ocurren los eventos.
* **cola\_llamadas**: Una lista que almacena los tiempos de llegada de las llamadas que están esperando ser atendidas. Representa la cola de espera para el servicio de llamadas.
* **cola\_personas**: Una lista que almacena los tiempos de llegada de los clientes en persona que están esperando ser atendidos. Representa la cola de espera para el servicio en persona.
* **empleado\_ocupado**: Una variable booleana que indica si el empleado está actualmente atendiendo a un cliente (True) o está libre (False).
* **tipo\_servicio\_actual**: Una cadena que indica el tipo de servicio que el empleado está prestando actualmente ('persona' o 'llamada'). Es None si el empleado está libre.
* **tiempo\_fin\_servicio**: Un valor numérico que almacena el instante de tiempo en el que el servicio actual (si lo hay) finalizará.

#### Entidades y sus atributos

Las entidades son los objetos que fluyen a través del sistema y son procesados. Los atributos son las características específicas de cada entidad.

* **Cliente (Persona)**:
  + **Tiempo de llegada**: El instante en que el cliente en persona llega al sistema (se almacena en cola\_personas).
  + **Tiempo de servicio requerido**: La duración del servicio que necesita el cliente (generado aleatoriamente al inicio del servicio).
* **Llamada (Cliente por teléfono)**:
  + **Tiempo de llegada**: El instante en que la llamada entra al sistema (se almacena en cola\_llamadas).
  + **Tiempo de servicio requerido**: La duración del servicio que necesita la llamada (generado aleatoriamente al inicio del servicio).
* **Empleado (Servidor)**:
  + **Estado**: Ocupado o libre (representado por la variable de estado del sistema empleado\_ocupado).
  + **Tipo de servicio actual**: El tipo de cliente al que está atendiendo actualmente (representado por la variable de estado del sistema tipo\_servicio\_actual).

Es importante aclarar que los tiempos de llegada de los clientes en persona y de las llamadas telefónicas, al igual que los tiempos de servicio requeridos se calculan utilizando la función generar\_tiempo\_exponencial(media). Esta genera tiempos aleatorios basados en una distribución exponencial tomando como entrada la media de esta distribución. Internamente, utiliza la función random.expovariate() de la librería random de Python, a la que se le proporciona la tasa (inversa de la media). Esta función se encarga de producir números aleatorios donde los valores más pequeños (tiempos cortos) son más probables que los valores grandes, siguiendo así el patrón de una distribución exponencial.

#### Eventos

Son los puntos en el tiempo en los que el estado del sistema puede cambiar. La simulación avanza procesando estos eventos en orden cronológico.

* **Llegada de cliente en persona (llegada\_cliente\_persona)**: Ocurre cuando un nuevo cliente llega al teatro en persona.
* **Llegada de llamada (llegada\_llamada)**: Ocurre cuando una nueva llamada telefónica entra al sistema.
* **Fin de servicio (finalizar\_servicio)**: Ocurre cuando el empleado termina de atender a un cliente (ya sea en persona o por teléfono).

#### Contadores y acumuladores

Son variables que se utilizan para recopilar estadísticas sobre el rendimiento del sistema durante la simulación.

* **total\_espera\_persona**: Un acumulador que suma el tiempo total que todos los clientes en persona han pasado esperando en la cola.
* **clientes\_atendidos\_persona**: Un contador que registra el número total de clientes en persona que han sido atendidos.
* **total\_espera\_llamada**: Un acumulador que suma el tiempo total que todas las llamadas han pasado esperando en la cola.
* **llamadas\_atendidas**: Un contador que registra el número total de llamadas que han sido atendidas.

#### Medidas de desempeño

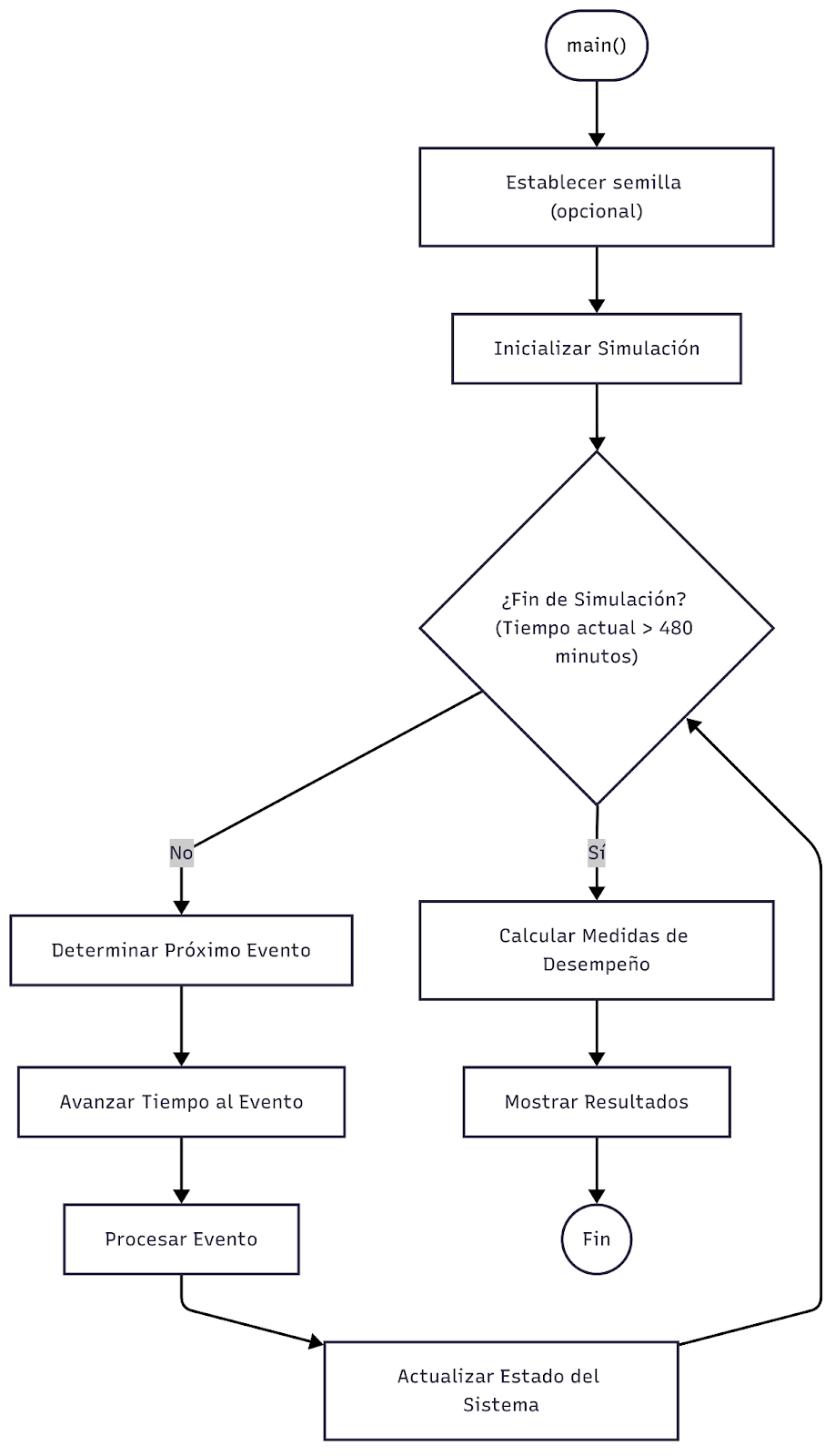
Son métricas que se calculan al final de la simulación utilizando los contadores y acumuladores para evaluar el rendimiento del sistema.

* **Tiempo de espera promedio de clientes en persona**: Calculado dividiendo total\_espera\_persona por clientes\_atendidos\_persona. Indica el tiempo promedio que un cliente en persona debe esperar antes de ser atendido.
* **Tiempo de espera promedio de llamadas**: Calculado dividiendo total\_espera\_llamada por llamadas\_atendidas. Indica el tiempo promedio que una llamada debe esperar antes de ser atendida.

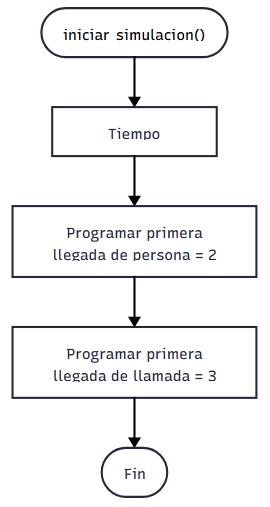
Es importante recalcar que, con el propósito de mantener una uniformidad en la entrega de los resultados se ha empleado también una semilla para la librería random (usando random.seed(44)), de modo que cada vez que se corre la simulación los resultados son siempre los mismos. Para hallar distintos resultados en las medidas de desempeño solamente hace falta eliminar esta línea o cambiar la semilla, de modo que la función para generar tiempos basándose en una distribución exponencial genere números aleatorios distintos.

### Diagramas de flujo

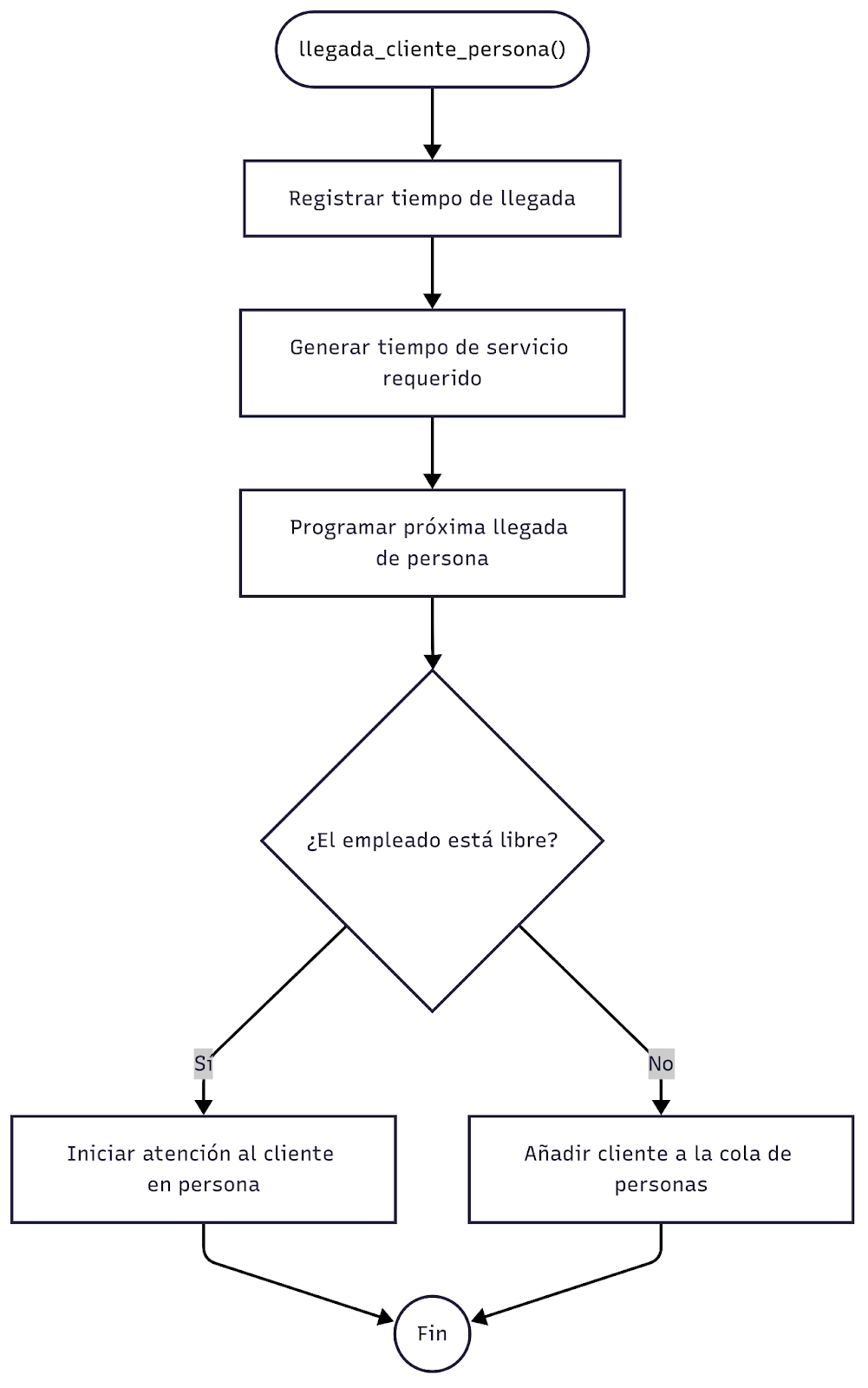
* **Programa principal (main):**



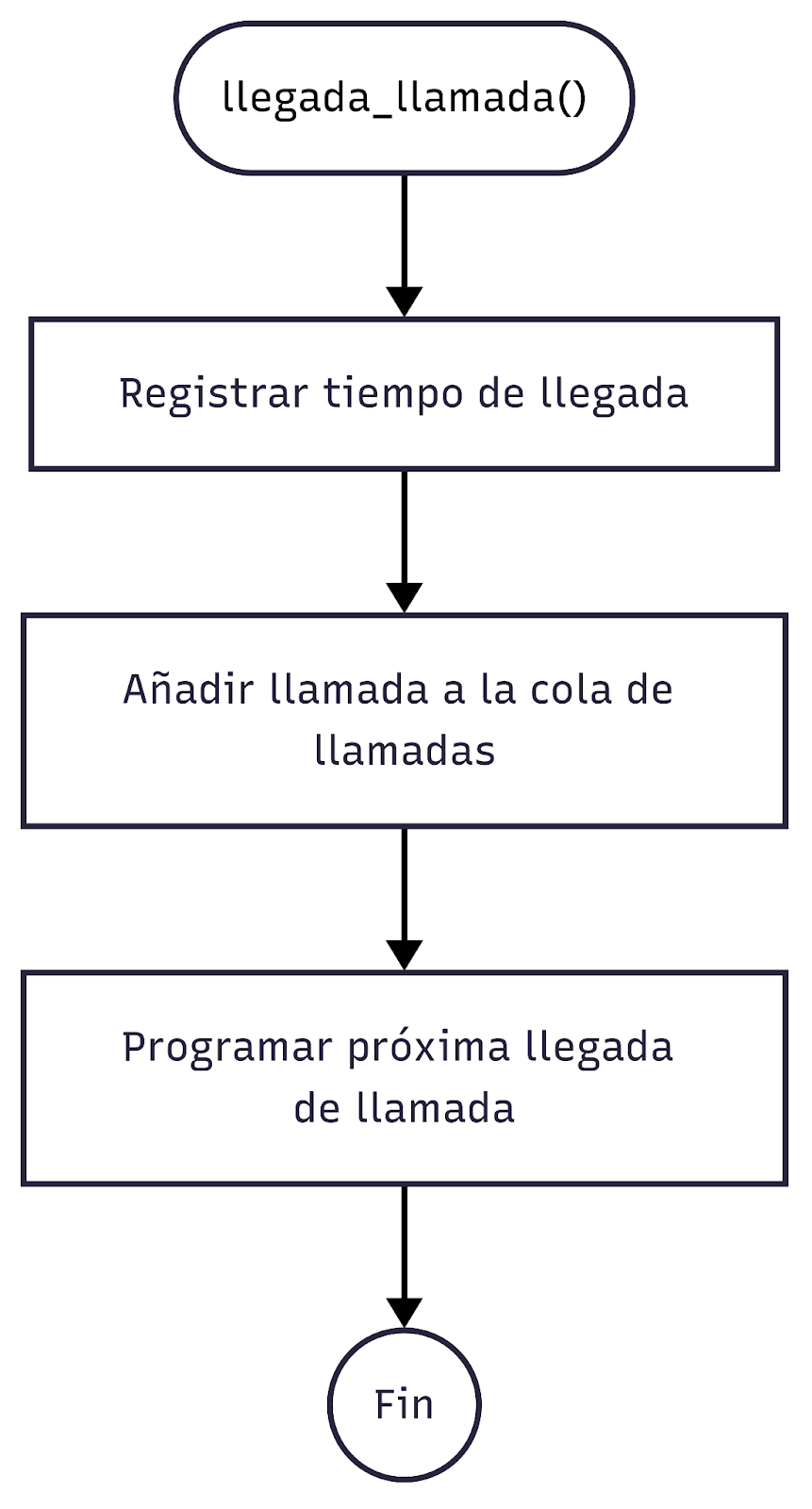
* **iniciar\_simulacion():**



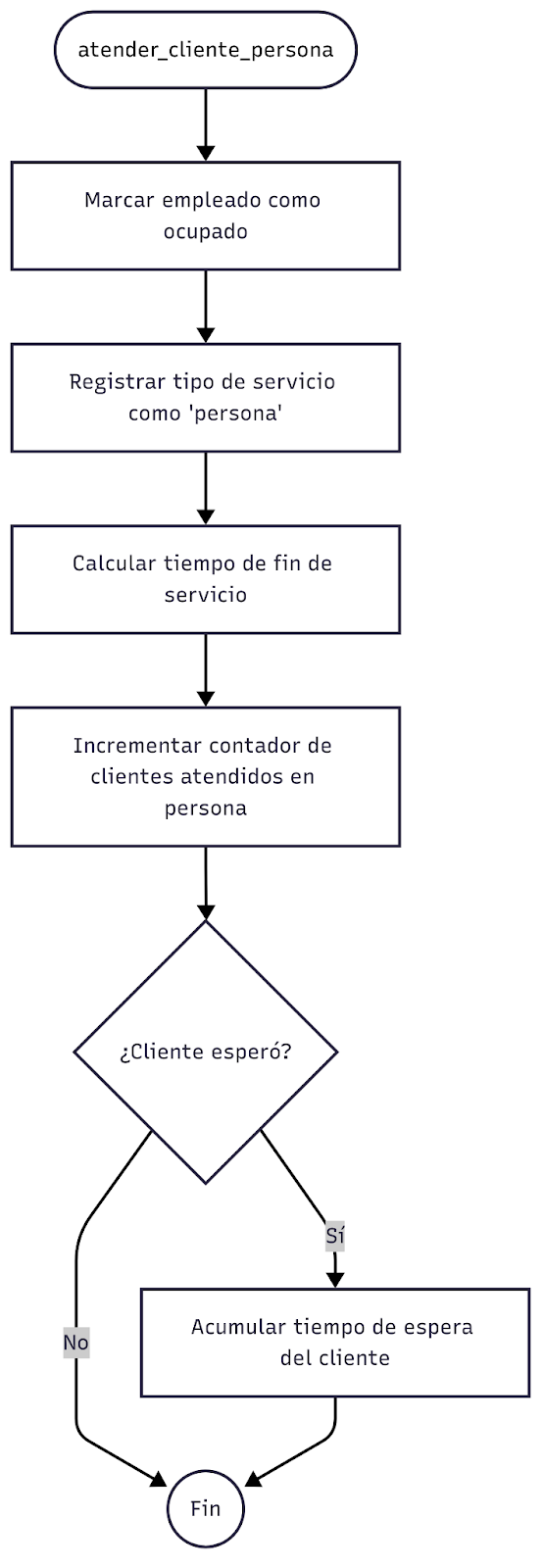
* **llegada\_cliente\_persona():**



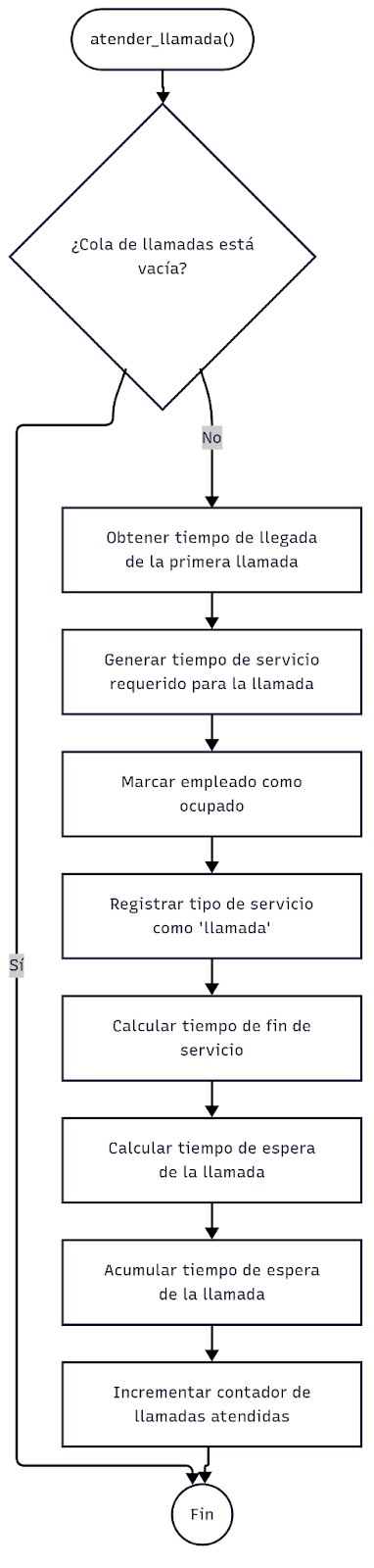
* **llegada\_llamada():**



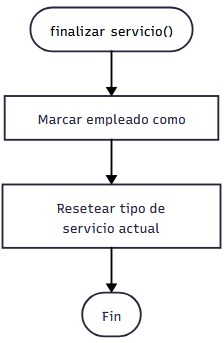
* **atender\_cliente\_persona:**

****

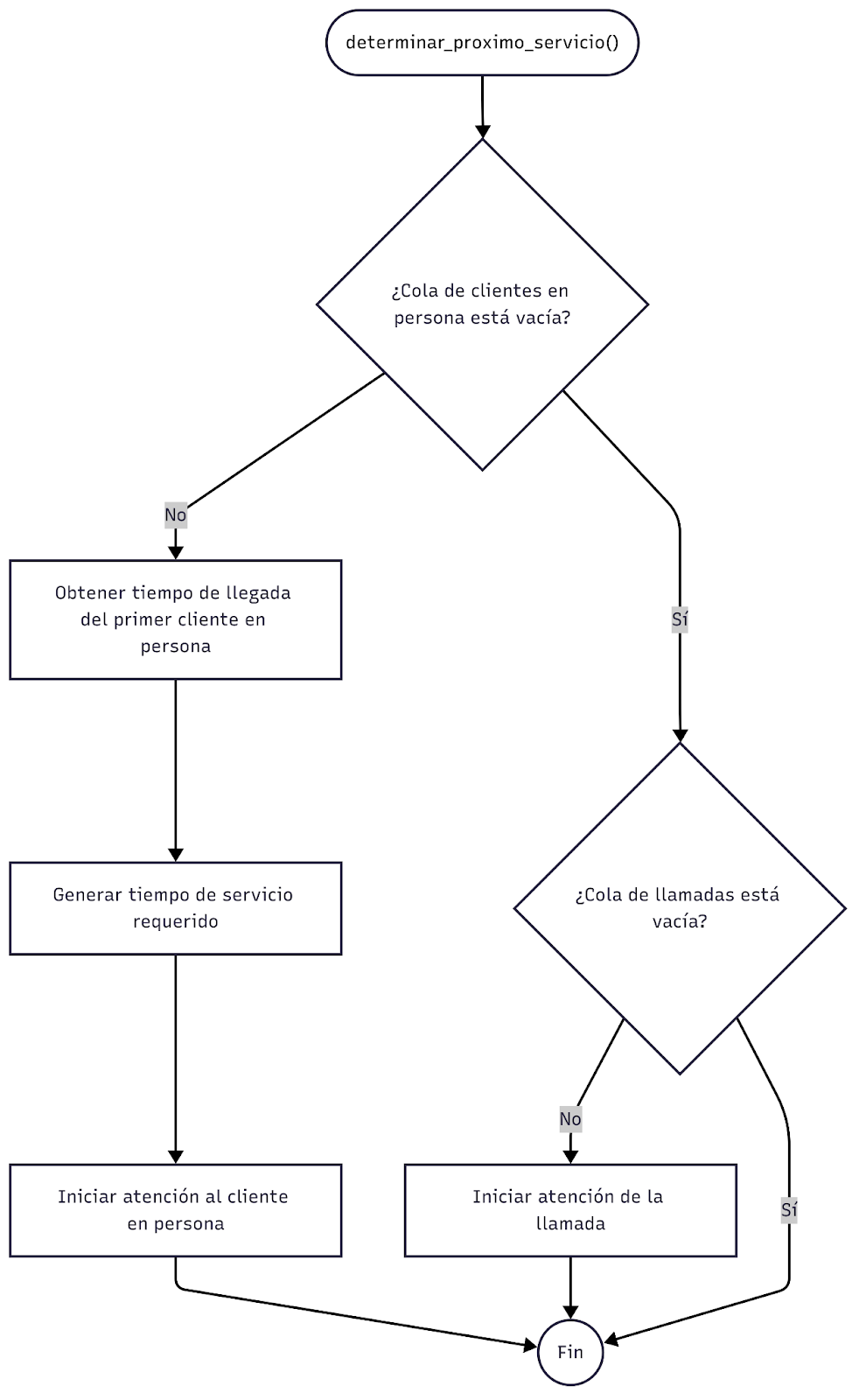
* **atender\_llamada():**

****

* **finalizar\_servicio():**



* **determinar\_proximo\_evento():**



### Desarrollo del simulador

El código del algoritmo implementado se puede encontrar en el siguiente Google Colab:

[problema-1-4-ejercicios-asignados](https://colab.research.google.com/drive/1lbfz0l5AmgzvPZdzqynZdxSESYa0Vf4k?authuser=1#scrollTo=sPtnUpeaAnVi)

### Análisis de resultados

Los resultados obtenidos utilizando la semilla random.seed(44) son los siguientes:

RESULTADOS SIMULACIÓN TEATRO:

Tiempo total de simulación: 8 horas (480 minutos)

Medias de las distribuciones exponenciales (minutos):

- Llegada de cliente en persona: 12

- Servicio a cliente en persona: 6

- Llegada de llamada: 10

- Servicio de llamada: 5

Indicadores de Desempeño:

- Tiempo de espera promedio de clientes en persona: 8.52 minutos

- Tiempo de espera promedio de llamadas: 79.21 minutos

- Total de clientes en persona atendidos: 45

- Total de llamadas atendidas: 47

- Total de servicios completados: 92

De los indicadores de desempeño se puede ver que el tiempo de espera promedio para los clientes en persona es significativamente menor al tiempo de espera promedio para las llamadas, lo cuál se explica fácilmente por el sistema de prioridad del empleado del teatro. Recordemos que las llamadas solo son atendidas cuando no hay clientes en persona en cola, por lo que si hay clientes en la fila del teatro las llamadas se postergan hasta que el empleado termine de atender a todos los clientes en persona. Sin embargo, se puede observar también que el número de llamadas atendidas fue mayor al número de clientes en persona atendidos, lo cuál puede sonar contraproducente al inicio, puesto que las personas que llaman tienen que esperar más en promedio, pero que se explica gracias a los tiempos de servicio de cada tipo de atención; como la media de la distribución exponencial de los tiempos de servicio de las llamadas es menor que la media de los tiempos de servicio de los clientes en persona, el empleado aunque le de prioridad a los clientes físicamente presentes en el teatro, logra atender más llamadas porque son de menor duración.

Es importante mencionar que la simulación fue evaluada con más casos (es decir sin usar la semilla) y se encontraron resultados muy similares, con tiempos de espera promedio menores para los clientes en persona y más llamadas atendidas que clientes en persona. Estos resultados demuestran la buena implementación del modelo, puesto que son acordes al sistema de prioridad del empleado y a los tiempos de llegada y de servicio en promedio para las distribuciones exponenciales planteadas en el enunciado del problema.

## Problema 1.5

Una compañía tiene dos fábricas, A y B, que se encuentran en la misma zona urbana. La compañía organiza un sistema de autobuses entre las dos fábricas, entre la 9 de la mañana y las 5 de la tarde. El autobús parte cada día de la fábrica A. En cada fábrica, el vehículo espera hasta que lo hayan abordado N personas, antes de salir hacia la otra fábrica. El tiempo de recorrido entre las dos fábricas está distribuido normalmente, con una media de 31 minutos y una desviación estándar de 5 minutos. Los pasajeros llegan a la terminal de la fábrica A con una distribución de Poisson a la tasa de 9 por hora, y a la terminal de la fábrica B con una distribución de Poisson y una media de cinco por hora. ¿Cuál debería ser el valor de N para minimizar el tiempo medio de espera por persona? El tiempo de espera no incluye el que se pasa en el autobús.

### Identificación del problema

La compañía tiene un sistema de autobuses entre dos fábricas, Se busca determinar la cantidad N de pasajeros que deberían abordar con el propósito de minimizar el tiempo de promedio de espera de los pasajeros en las terminales de la fábrica.

#### Variables de estado del sistema

* **Reloj de simulación  
  -**t inicia a las 9:00 y termina a las 5:00 (480 min)
* **Parámetros fijos**- Tiempo de viaje μ = 31 min  
  - Desviación σ = 5 min  
  - Horario de operación: 9 h–17 h (480 min)
* **Tasas de llegada**- Terminal A: λ₁ = 9/60 distribuye poisson  
  - Terminal B: λ₂ = 5/60 distribuye poisson
* **Colas  
  -**QA(t), QB(t): número de pasajeros esperando en cada terminal
* **Estado del autobús**-ubicación ∈ {A, B}: ubicación actual  
  -Estado ∈ {“espera”, “viaje”}: si está embarcando o en ruta

#### Entidades y sus atributos

1. **Pasajeros**
   * Terminal de origen (A o B)
   * Tiempo de llegada (arrival time)
   * Tiempo de espera en cola (waiting time = depart time – arrival time)
2. **Autobús**
   * Capacidad mínima para partir: N pasajeros
   * Tiempo de partida (depart\_time)
   * Tiempo de viaje, distribuye normal (31, 5)
   * Tiempo total en ruta
   * Ubicación actual
3. **Terminal**
   * Identificador (A o B)
   * Cola de llegadas pendientes (arrivals\_A, arrivals\_B)
   * Contadores de pasajeros atendidos (served\_A, served\_B)

#### Eventos

1. **Llegada de pasajero a A/B**
   * Se programa según proceso Poisson
   * Incrementa ; si el bus está en “*espera*” y ya hay ≥ N, el bus sale
2. **Salida del autobús de la terminal**
   * Ocurre en *depart\_time*
   * Cambia estado a “viaje” y decrementa la cola en N unidades
3. **Llegada del autobús a la terminal opuesta**
   * Se programa en *depart\_time + travel\_time*
   * Cambia ubicación, estado → “espera”, reinicia conteo de N pasajeros
4. **Fin de jornada**
   * Cuando *depart\_time* > 480 min, se detiene la simulación

#### Contadores y/o acumuladores

* **Total de pasajeros servidos:** *served\_A, served\_B*
* **Pasajeros rezagados al cierre:**

**Sumatoria de tiempos de espera**

* **Área bajo Q(t)**
* **Utilización del autobús**U =
* **Número total de viajes:** trips

#### Medidas de desempeño

1. **Tiempo de espera promedio**
   * **Por terminal:**

**Global:**

1. **Promedio de pasajeros en cola**

**Utilización del autobús**

**Viajes promedio por día**

1. **Pasajeros rezagados totales  
      ​**

### Diagramas de flujo

**Rutina de inicialización**

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

**Rutina del pasajero:**

Imagen que contiene Escala de tiempo

El contenido generado por IA puede ser incorrecto.

**Rutina salida del bus.**

Escala de tiempo

El contenido generado por IA puede ser incorrecto.

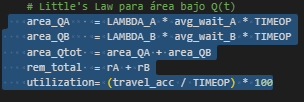
**Rutina llegada del bus**

Imagen que contiene Escala de tiempo

El contenido generado por IA puede ser incorrecto.

### Desarrollo del código en Python (COLAB) [Simulacion\_Autobus.ipynb](https://colab.research.google.com/drive/1WpmSQkk8q5enuxqaJ85QNpGXSFx_Gp5N?usp=sharing)

El código simula los viajes de bus entre fabricas teniendo en cuenta ciertos parámetros iniciales, usamos la distribución de poisson con la inversa para la generación de llegada de pasajeros que en la simulación se pasará como una lista que tendrá en cuenta esto e ira generando llegadas de pasajeros a cada terminal dependiendo de su tasa. Se generan múltiples variables para los resultados como la espera en cada cola de pasajeros, la utilización del bus, el tiempo de partida del bus y su tiempo sin viajar, el promedio de viajes, de espera de pasajeros entre otras variables realizadas en el total de simulaciones repetidas con distintos valores, de 1 a 31 pasajeros. En este código usamos IA principalmente para generar lo siguiente:



Qué son las curvas de tiempo en las colas teniendo en cuenta el número de personas promedio en la cola, la tasa de llegadas de cada terminal y el tiempo medio de cada persona esperando el bus. también nos sirvió como apoyo para hacer un código más limpio y la interpretación de variables y la generación de algunas nuevas como la utilización del bus, para observar qué tan eficiente había sido el bus teniendo en cuenta el número de pasajeros que subían y el tiempo de espera.

### Resultados y Análisis:

Un conjunto de letras blancas en un fondo blanco

El contenido generado por IA puede ser incorrecto.

Gráfico, Gráfico de líneas

El contenido generado por IA puede ser incorrecto.

Gráfico, Gráfico de líneas

El contenido generado por IA puede ser incorrecto.

**Análisis de resultados:**

La simulación nos muestra que con un bus en 480 minutos con todos los parámetros iniciales en este caso el mejor valor es 8 pasajeros. Haciendo múltiples pruebas con el código variando las repeticiones de la simulación este valor entre mayores repeticiones oscila entre 7 a 10 pasajeros ya que entre 10 y 15 los valores de utilización del bus decrementan y los tiempos de espera no siempre mejoran. Las gráficas nos muestran primeramente la comparación entre “Viajes promedio” y “N”, donde observamos que con N=1 el autobús realiza alrededor de 16 viajes por día, esto es muy ineficiente ya que el total de pasajeros atendidos sería muy poco.

Al aumentar N el número de viajes va bajando cuando N está entre [6,10] los valores de viajes por día llegan a 10 y de ahí para adelante es cuando el número de viajes disminuye mucho más al esperar más pasajeros. Sin embargo usando los valores de esta gráfica podemos remitirnos a la de “espera media vs N” donde esta curva empieza alta, luego baja y progresivamente vuelve a subir, cuando N es muy pequeño el valor de espera medio llega a casi 200 min comenzando a disminuir a medida que el valor de pasajeros aumenta, hasta llegar a N = [6,10] donde específicamente entre N = [7,8] su espera media está entre 73 y 75 minutos, ya cuando N > 8 la espera vuelve a aumentar, puesto que el bus se demora más tiempo en llenarse teniendo en cuenta las tasas de llegada de los pasajeros, así que después de los 8, los pasajeros comienzan a verse penalizados con el tiempo de espera de los demás.

En los resultados del código también vemos que la llegada de pasajeros a la terminal B es menor que la de A por lo tanto es normal que la proporción de pasajeros atendidos y pasajeros rezagados sea mala para la terminal A, puesto que el tiempo de espera de pasajeros es menor al tiempo de llegada del bus a una terminal (31 min de media) entonces es coherente que haya más pasajeros rezagados en la terminal A a comparación de la terminal B, aunque se nota en el código que los valores son bastante separados entonces se duda de la fiabilidad en algunos casos

## Problema 1.6

Una facilidad de servicio consiste en dos servidores en serie, cada uno con su propia fila. Un cliente terminando servicio en el servidor 1 procede al servidor dos, mientras que un cliente terminando el servicio en el servidor 2 deja la instalación. Suponga que los clientes llegan al servicio 1 con distribución uniforme entre 1 y 2 minutos y con probabilidad 0.3 es un cliente tipo A. Este tipo de cliente tiene prioridad sobre el cliente tipo B en la cola del servidor 1. El tiempo de servicio en el servidor 1 es exponencial con una media de 1 min. y en el servidor 2 es de 0.8 minutos. El tiempo de desplazamiento entre la salida del servidor 1 y la llegada a la cola del servidor 2 es uniforme entre 0.5 y 2 minutos. Simule este sistema por 15 horas y estime la demora esperada y el número promedio de clientes en cada cola y la utilización de cada servidor.

### Identificación del problema

Una instalación de servicio consta de dos servidores en serie (Servidor 1 y Servidor 2), cada uno con su propia fila. Los clientes llegan al Servidor 1 y, tras completar el servicio, se desplazan al Servidor 2 antes de abandonar el sistema. Existen dos tipos de clientes:

* Tipo A: (prioridad) con probabilidad 0.3 en cada llegada. (en la cola del servidor 1)
* Tipo B: con probabilidad 0,7.

Características de la simulación:

Relaciones estadísticas:

* Llegadas al Servidor 1: distribución uniforme entre 1 y 2 minutos.
* Tiempo de servicio en Servidor 1: exponencial con media de 1 minuto.
* Tiempo de desplazamiento entre servidores: uniforme entre 0.5 y 2 minutos.
* Tiempo de servicio en Servidor 2: exponencial con media de 0.8 minutos.

Objetivo de la simulación:

* Simular 15 horas de operación contínua.
* Estimar:
  + Tiempo medio de demora en cola para los clientes en cada servidor.
  + Número promedio de clientes en cada cola.
  + Utilización de cada servidor.

#### 

#### Variables de estado del sistema

Q1\_A: Número de clientes tipo A en la cola del Servidor 1.

Q1\_B: Número de clientes tipo B en la cola del Servidor 1

Q2: Número de clientes en la cola del Servidor 2

S1\_busy: Variable booleana que indica si el Servidor 1 está ocupado.

S2\_busy: Variable booleana que indica si el Servidor 2 está ocupado.

t: Reloj de la simulación, en minutos.

#### Eventos y Actividades

**Eventos:**

Llegada al Servidor 1: Ocurre cuando llega un nuevo cliente al Servidor 1 ( tipo A|B).

Salida del Servidor 1: Terminación del servicio de un cliente en Servidor 1, lo envía a cola S2.

Llegada al Servidor 2: Evento programado tras desplazamiento (uniforme 0.5 – 2 min).

Salida del Servidor 2: Terminación del servicio en Servidor 2, cliente abandona el sistema.

**Actividades:**

Atención al cliente: Los servidores ocupados atienden a los clientes durante el tiempo especificado.

Espera en cola: Los clientes permanecen en la cola hasta que un servidor esté libre.

#### Contadores y/o acumuladores

| Acumulador | Descripción |
| --- | --- |
| area\_Q1 | , promedio de clientes en cola 1 |
| area\_Q2 | , promedio de clientes en cola 2 |
| t\_busy\_S1 | Tiempo acumulado en que S1\_busy = 1 |
| t\_busy\_S2 | Tiempo acumulado en que S2\_busy = 1 |
| sum\_delay\_S1 | Suma de todas las demoras en cola en Servidor 1 |
| sum\_delay\_S2 | Suma de todas las demoras en cola en Servidor 2 |
| n\_completed\_S1 | Número de clientes servidos en S1 |
| n\_completed\_S2 | Número de clientes servidos en S2 |

#### Medidas de desempeño

1. **Demora media en cola**

* = sum\_delay\_S1 / n\_completed\_S1
* = sum\_delay\_S2 / n\_completed\_S2

1. **Número promedio en cola**

* = area\_Q1 / T
* = area\_Q2 / T

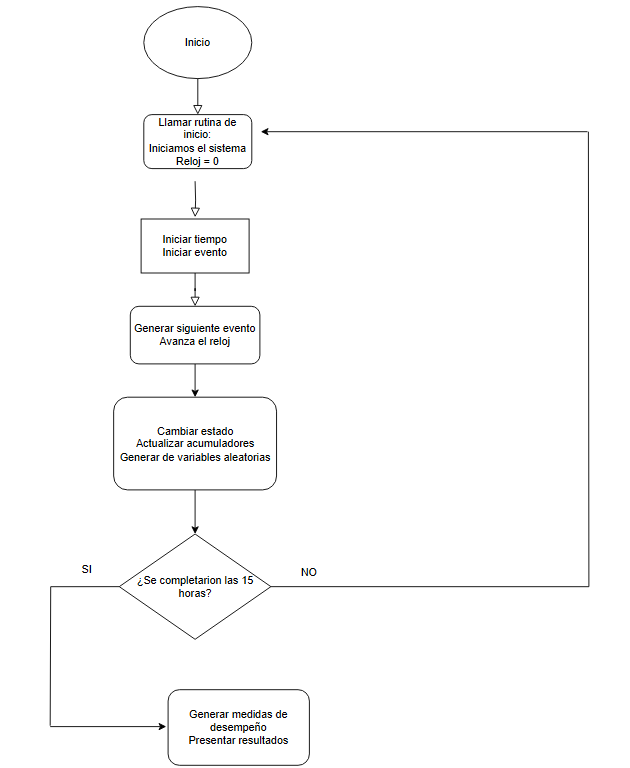
1. **Utilización del servidor**

* ρ₁ = t\_busy\_S1 / T
* ρ₂ = t\_busy\_S2 / T

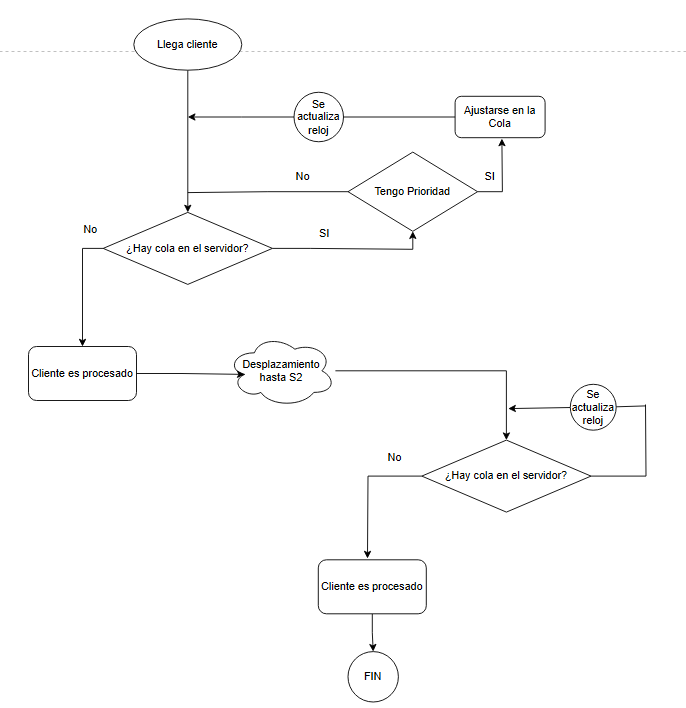
T = 900 minutos es el horizonte de simulación.

**Diagramas de flujo**

**Rutina inicialización**

****

**Rutina de cliente**



**Código en Python de la simulación**

[**Ejercicios Individuales de Simulacion.ipynb**](https://colab.research.google.com/drive/1ZNSFx3meHglwoSIbjKLFlwph-hf_ljgN#scrollTo=SqiSeznXsB-L)

**Análisis y resultados**

Las cifras confirman que el sistema opera en un régimen moderadamente cargado pero estable: la utilización de (S1 ≈0.676) y (S2 ≈0.542) coincide con lo esperado de λ/μ y está por debajo de 1, evitando saturación. El número medio en cola es bajo (≈0.56 clientes) en S1 y (≈0.50 en S2), lo cual produce demoras medias reducidas (≈0.84 min en S1 y ≈0.76 min en S2) frente a los tiempos de servicio (1 min y 0.8 min respectivamente). Además, la prioridad de los clientes tipo A en S1 parece que ayuda a mantener aún menor su tiempo de espera en general. En resumen, los resultados indican un servicio ágil con colas cortas y sin congestión.

# Referencias

1. **Wikipedia.** (s.f.). *Triangular distribution*. Recuperado de<https://en.wikipedia.org/wiki/Triangular_distribution>
2. **Khan Academy.** (s.f.). *Probability density functions*. Recuperado de<https://www.google.com/search?q=https://www.khanacademy.org/math/probability/probability-functions-discrete/probability-density-functions-continuous/a/probability-density-functions>
3. **Wikipedia.** (s.f.). *Inverse transform sampling*. Recuperado de<https://en.wikipedia.org/wiki/Inverse_transform_sampling>
4. **Fiveable.** (s.f.). *Mean of a Continuous Random Variable - (Intro to Probability)*. Recuperado de<https://fiveable.me/key-terms/introduction-probability/mean-of-a-continuous-random-variable>
5. **Investopedia.** (2023, Marzo 2). *Law of Large Numbers (LLN)*. Recuperado de<https://www.investopedia.com/terms/l/lawoflargenumbers.asp> (Investopedia ofrece una explicación accesible de la Ley de los Grandes Números, aunque en un contexto financiero, el principio es el mismo).
6. **Python.org.** (s.f.). *About Python*. Recuperado de<https://www.python.org/about/>
7. **Google Colaboratory.** (s.f.). *Welcome to Colaboratory*. Recuperado de<https://colab.research.google.com/>
8. **NumPy.org.** (s.f.). *What is NumPy?*. Recuperado de<https://numpy.org/doc/stable/user/whatisnumpy.html>
9. **Python Documentation.** (s.f.). *random — Generate pseudo-random numbers*. Recuperado de<https://docs.python.org/3/library/random.html>
10. **Matplotlib Documentation.** (s.f.). *Pyplot tutorial*. Recuperado de<https://matplotlib.org/stable/tutorials/introductory/pyplot.html>
11. **Piecewise uniform distribution.** (s. f.). *Wikipedia*. [Online]. Recuperado de: [https://www.google.com/search?q=https://en.wikipedia.org/wiki/Piecewise\_uniform\_distribution](https://www.google.com/search?q=https://en.wikipedia.org/wiki/Piecewise_uniform_distribution&authuser=1).