U.B.A. FACULTAD DE INGENIERÍA

Departamento de Computación

Modelos y Simulación 7526 - 9519

TRABAJO PRÁCTICO #1

Números al azar y Test estadísticos

Curso: 2019 - 1er Cuatrimestre

Turno: Miércoles

GRUPO N° 1	
Integrantes	**Padrón**
Amurrio, Gastón	93584
Gamarra Silva, Cynthia Marlene	92702
Pinto, Tomás	98757
Fecha de Entrega:	24-04-2019
Fecha de aprobación:	
Calificación:	
Firma de aprobación:	

Observaciones:

Índice

1. Enunciado del trabajo práctico

Trabajo Práctico 1

Modelos y Simulación - 75.26 – 95.19

Consideraciones generales

Debe entregarse un informe explicando el procedimiento utilizado para resolver cada ejercicio, detallando las conclusiones que se solicitan en cada punto, e integrando el código fuente utilizado.

Números al azar

Ejercicio 1

Utilizando Matlab, Octave o Python implementar un Generador Congruencial Lineal (GCL) de módulo 2^{32}, multiplicador 1013904223, incremento de 1664525 y semilla igual a la parte entera de la suma ponderada (0,15-0,25-0,6) de los números de padrón de los integrantes del grupo, ordenados ascendentemente.

- Informar los primeros 5 números de la secuencia.
- Modificar el GCL para que devuelva números al azar entre 0 y 1, y realizar un histograma sobre 100.000 valores generados.

Ejercicio 2

Utilizando el generador de números aleatorios con distribución uniforme [0,1] implementado en el ejercicio 1 y utilizando el método de la transformada inversa genere números pseudoaleatorios con distribución exponencial negativa de media 20.

- Realizar un histograma de 100.000 valores obtenidos.
- Calcular la media y varianza de la distribución obtenida y compararlos con los valores teóricos.

Ejercicio 3

Utilizando el método de Box-Muller genere de números aleatorios con distribución normal standard.

- Realizar un histograma de 100.000 valores obtenidos.
- Calcular la media y varianza de la distribución obtenida y compararlos con los valores teóricos.

Ejercicio 4

Genere 100.000 número aleatorios con distribución Normal de media 40 y desvío estándar 6 utilizando el algoritmo de Aceptación y Rechazo.

- Realizar un histograma de frecuencias relativas con todos los valores obtenidos.
- Comparar, en el mismo gráfico, el histograma realizado en el punto anterior con la distribución normal brindada por Matlab, Octave o Python.
- Calcular la media y la varianza de la distribución obtenida y compararlos con los valores teóricos.

Trabajo Práctico 1

Modelos y Simulación - 75.26 – 95.19

Ejercicio 5

Utilizando el método de la transformada inversa y utilizando el generador de números aleatorios implementado en el ejercicio 1 genere números aleatorios siguiendo la siguiente función de distribución de probabilidad empírica.

Probabilidad	Valor generado
.4	1
.3	2
.12	3
.1	4
0.08	5

Muestre los resultados obtenidos en un histograma.

Ejercicio 6

Utilizando 2 generadores de números al azar, provistos por el lenguaje elegido para resolver el tp, con distribuciones uniformes en [-1,1] genere números aleatorios en un círculo de radio 1 centrado en el origen.
Muestre el resultado en un gráfico de 2 dimensiones.

Test estadísticos

Ejercicio 7

Realizar, sólo gráficamente, un test espectral en 2 y 3 dimensiones al generador conguencial lineal implementado en el ejercicio 1. ¿Cómo se distribuyen espacialmente los puntos obtenidos?

Ejercicio 8

Realizar un test Chi 2 a la distribución empírica implementada en el Ej 5, analizar el resultado indicando si la distribución puede o no ser aceptada.

Ejercicio 9

Al generador congruencial lineal implementado en el ejercicio 1 realizarle un gap test para el intervalo [0,2 - 0,5], analizar el resultado indicando si pasa el test.

Ejercicio 10

Aplicar el test de Kolmogorov-Smirnov al generador de números al azar con distribución normal generado en el ejercicio 3, y analizar el resultado del mismo.
Graficar la distribución acumulada real versus la distribución empírica.

2. Introducción

El trabajo práctico consiste en aplicar conceptos teóricos explicados en clase sobre generación de números aleatorios aplicado a distintos métodos estadísticos utilizados en el medio científico como ser Box Muller, Generador Congruencial Lineal (GCL), Tranformada inversa y tests como Test espectral y Kolmogorov-Smirnov. Los ejercicios están simulados en lenguaje Python.

3. Implementación y resultados

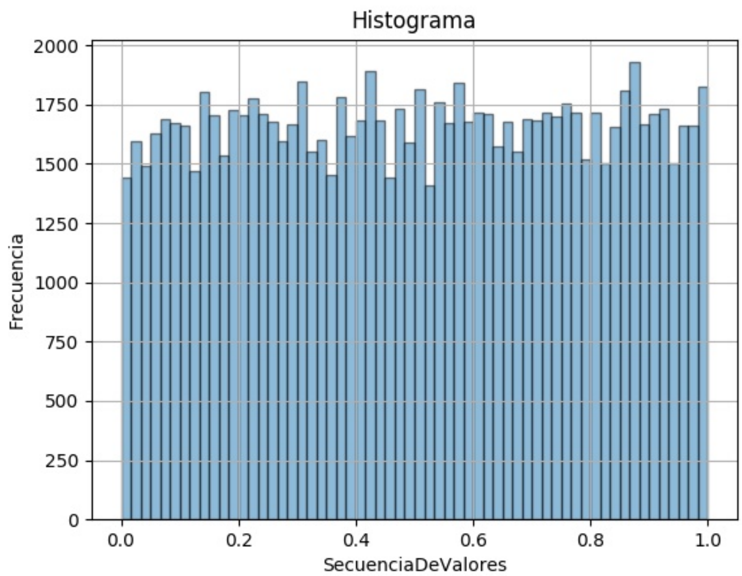
Para cada uno de los ejercicios pedidos se realiza una explicación de cada uno de ellos. Se toma como base teórica lo explicado en clase tanto teórica como clase práctica.

3.1. Ejercicio 1

Para realizar el ejercicio aplicamos lo siguiente:

1. Creamos la secuencia de números utilizando una Generador Congruencial Lineal(GCL). Los datos utilizados para el GCL son:
   - módulo = 2^{32}
   - multiplicador = 1013904223
   - incremento = 1664525
   - semilla = $92702 * 0{,}15 + 93584 * 0{,}25 + 98757 * 0{,}26$
2. Para que de números entre 0 y 1 se divide por su módulo
3. Graficamos el histograma utilizando la secuencia creada.

El resultado de los primeros 5 números de la secuencia: [62978, 383030987L, 2740587618L, 1650525291L, 2470812354L].
El histograma pedido utilizando el método Generador Congruencial Lineal (GCL) donde se grafica para números al azar entre 0 y 1,es el siguiente:



3.2. Ejercicio 2

Para realizar el ejercicio aplicamos lo siguiente:

1. Creamos la secuencia utilizando GCL con los mismos datos del ejercicio anterior.
2. A la secuencia creada le dividimos por su módulo para obtener números entre 0 y 1.
3. Aplicamos la función transformada inversa a la secuencia.
4. Realizamos un histograma con la nueva secuencia.
5. Calculamos la media y varianza de la distribución simulada y teórica.

El histograma pedido utilizando el método de la transformada inversa generado con números pseudo-aleatorios con distribución exponencial negativa de media 20 es el siguiente:

Comparando los resultados simulados y teóricos:

- El valor simulado de la media es 0.0501097366318
- El valor teórico de la media es 0.05
- El valor simulado de la varianza es 0.00250751904958
- El valor teórico de la varianza es 0.0025

Por lo tanto podemos observar que los valores simulados y teóricos son bastantes parecidos.

3.3. Ejercicio 3

Para realizar el ejercicio aplicamos lo siguiente:

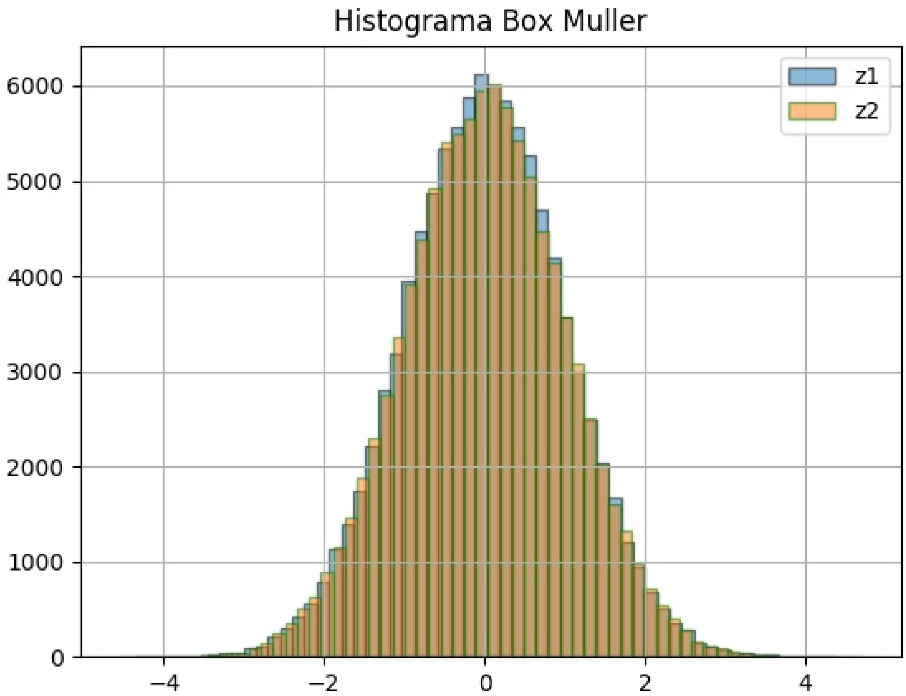
1. Creamos dos listas: u1 y u2 con valores de distribución normal 0, 1.
2. Aplicamos el método de Box Muller a estas dos listas y obtenemos dos listas nuevas z1 y z2.

3. Realizamos histogramas con z1 y z2 y otro histograma con una distribución normal estándar.
4. Calculamos la media y varianza de la distribución simulada y teórica.

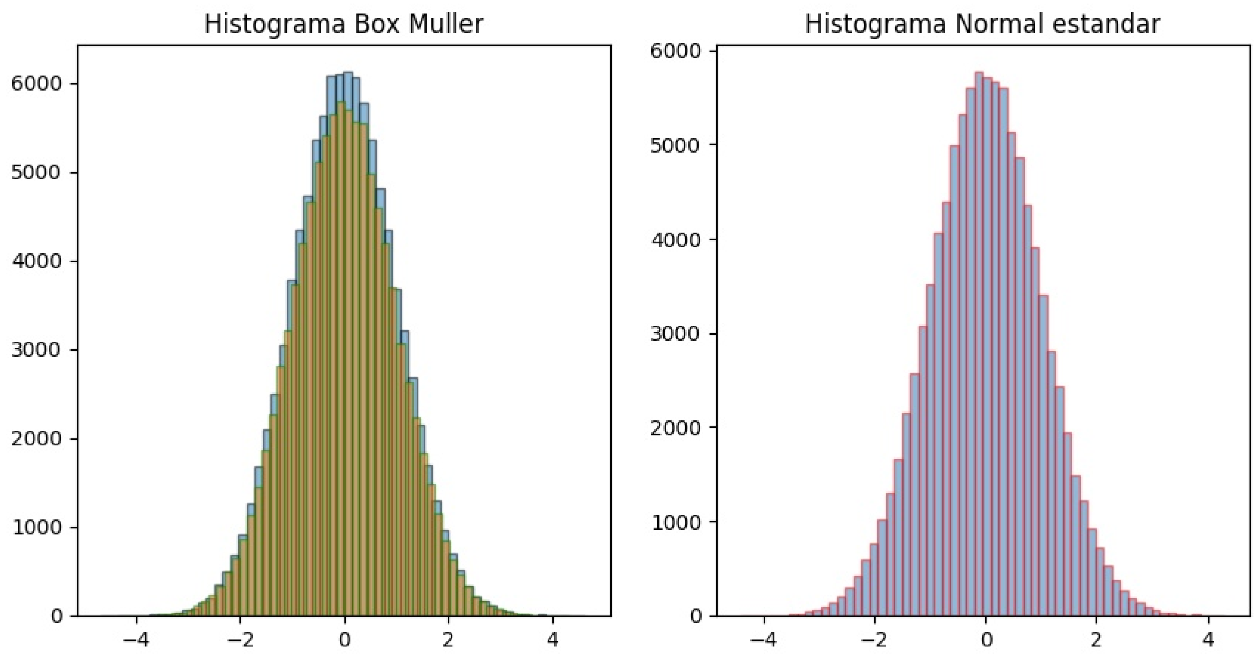
Los resultados que obtenemos son los siguientes:

- El valor simulado de la media z1 es 0.000743926097041
- El valor simulado de la media z2 es -0.0016462929471
- El valor teórico de la media es 0
- El valor simulado de la varianza z1 es 0.998271254875
- El valor simulado de la varianza z2 es 0.996519878651
- El valor teórico de la varianza es 1

El histograma pedido utilizando Box Muller es el siguiente:



Si comparamos con una distribución Normal estándar obtenemos:

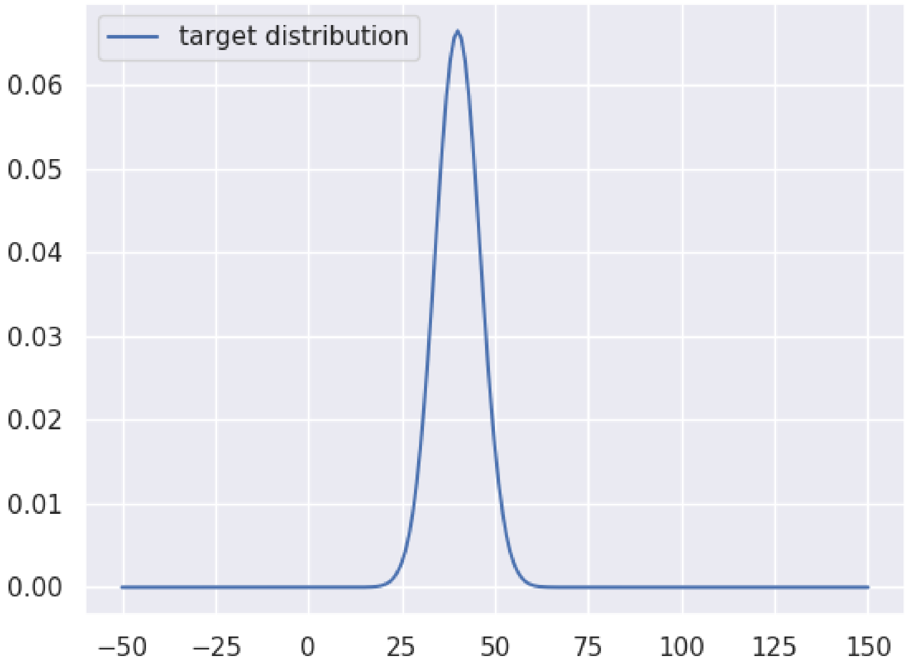


Por lo tanto podemos observar que se comprueba que el método de Box Muller es una distribución normal estándar.

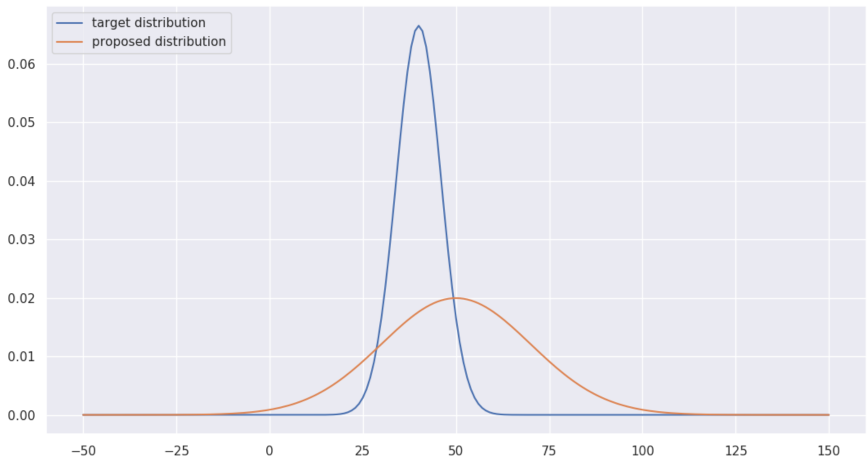
3.4. Ejercicio 4

Seguimos los pasos de esta guia: Rejection Sampling.

La distribucion target $P(x)$ es una $N(40, 6)$



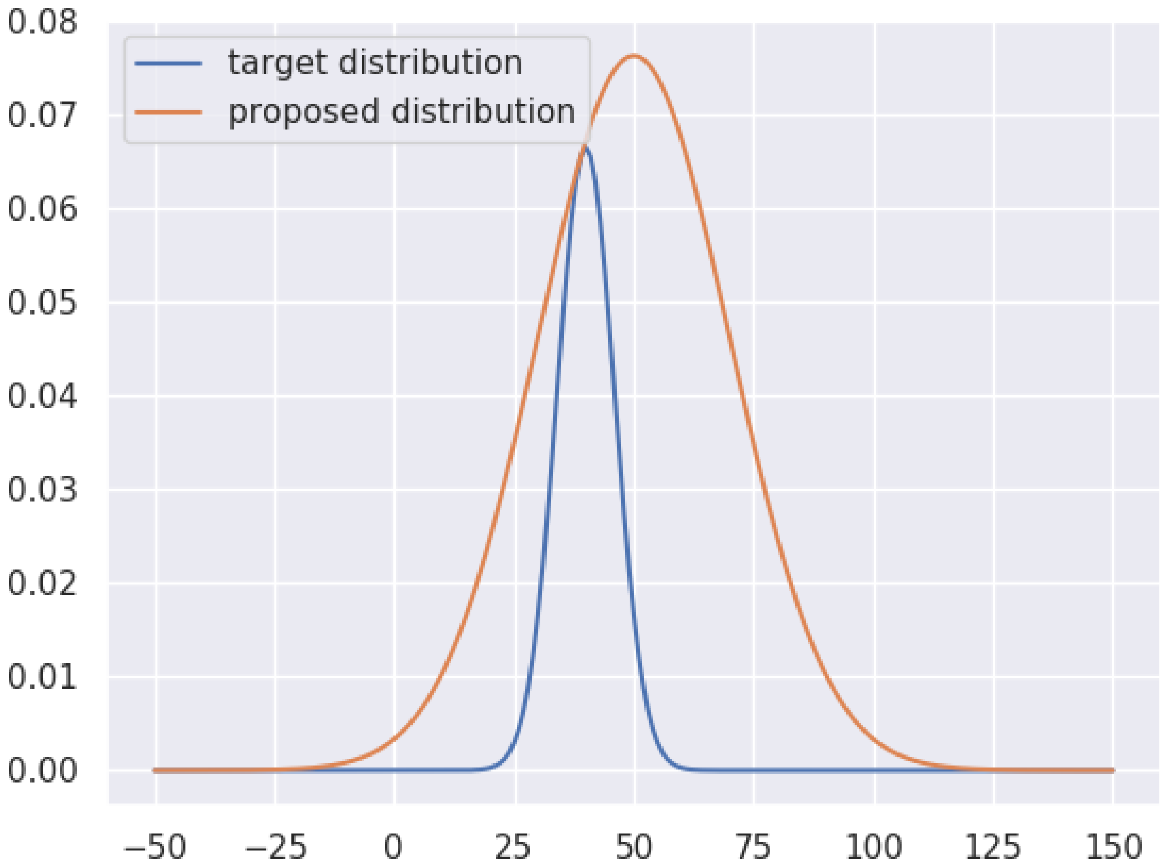
Proponemos $N(50, 30)$ como $Q(x)$



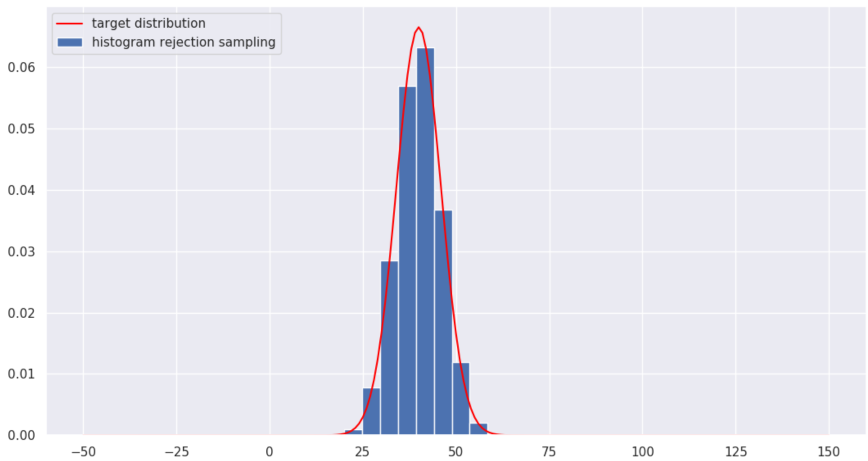
Rejection Sampling va a fallar debido a que la $Q(x)$ propuesta no esta envolviendo a nuestra distribucion target $P(x)$ Para remediar esto, encontramos un factor k que multiplique a $Q(x)$ que es la relacion maxima entre $P(x)$ y $Q(x)$, entonces:

$$k = max(P(x)/Q(x)) forallx$$

Al reescalar..



La distribucion generada mediante rejection sampling es la siguiente:



Este es el grafico de la distribucion real vs el histograma de valores generados por el algoritmo. Como podemos ver, el algoritmo logro generar una aproximacion bastante buena.

La media de la muestra generada es: 40.0, y la varianza: 35.58.

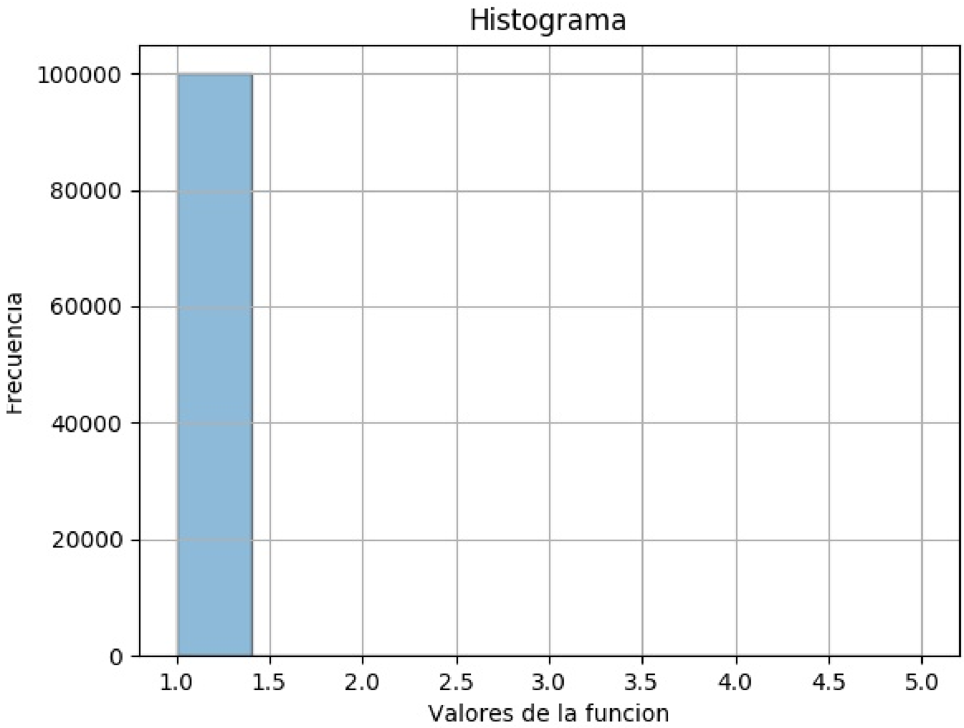
La varianza muestral es mucho mas amplia comparada con la teorica. La media muestral es la misma.

3.5. Ejercicio 5

Para realizar el ejercicio aplicamos lo siguiente:

1. Se genera 100000 valores utilizando GCL.
2. A la secuencia creada le dividimos por su módulo para obtener números entre 0 y 1.
3. Aplicamos la función transformada inversa a la secuencia utilizando la función de distribución de probabilidad empírica.
4. Realizamos un histograma con la nueva secuencia.

El histograma pedido utilizando la función de distribución de probabilidad empírica dada por el enunciado:

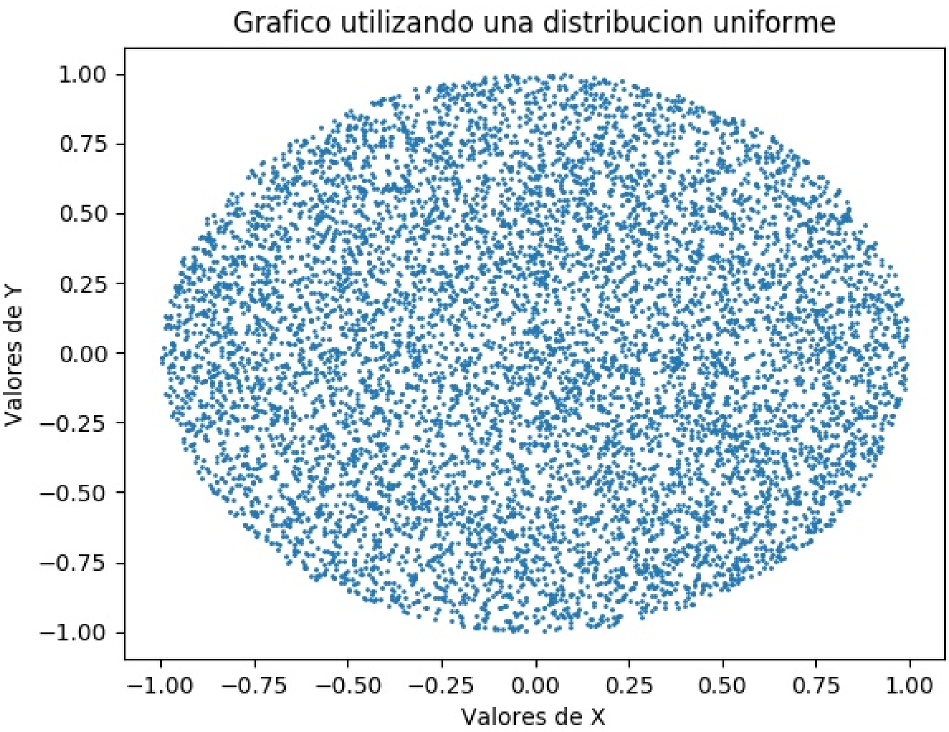


3.6. Ejercicio 6

Para realizar el ejercicio aplicamos lo siguiente:

1. Se genera 2 listas x,y con números aleatorios que provee PYTHON.
2. Con ambas listas x,y creamos dos listas nuevas filtrando con aquellas que cumplen $x^2 + y^2 < 1$.
3. Realizamos un gráfico utlizando estas dos listas nuevas.

   El gráfico pedido utilizando tilizando una distribucion uniforme entre [-1,1] generado con números aleatorios en un círculo de radio 1 centrado en el origen.



3.7. Ejercicio 7

3.8. Ejercicio 8

Utilizando el test Estadístico Chi^2 se aplica el método utilizando los siguientes pasos:

1. Utilizamos la distribución empírica generada en el ejercicio 5). Está distribución se corresponderá a N_i ocurrencias observadas.
2. Obtenemos una distribución uniforme con n = 100.000 muestras generadas
3. Medimos la dispersion de las ocurrencias obervadas N_i respectos de las esperadas $n * p_i$. La dispersión se calcula como:

$$D^2 = \Sigma_{k=1}^{k-1} \frac{(N_i - np_i)^2}{np_i}$$

4. Calculamos $D^2 < t$ para saber si aceptamos la hipótesis con un error del 1 %

   En nuestro caso obtuvimos que se rechaza la hipótesis con la distribución empirica con un error del 1 %

3.9. Ejercicio 9

Se siguieron los pasos del algoritmo explicado aqui: Gap test example.

1. Utilizamos el generador congruencial lineal implementado en el ejercicio 1
2. Asumimos un alpha de 0.05.
3. Definimos la hipotesis:

- H_0: Se asumen independientes los numeros generados por el generador basandonos en las frecuencias de los gaps.
- H_1: No son independientes.

4. Recorremos el array de numeros generados, inicializando un contador en 0.
5. Si nos encontramos con un numero en el intervalo pedido, registramos el valor actual del contador y lo reseteamos.
6. Repetimos los pasos 4 (desde donde registramos el valor incluido en el intervalo) y 5 hasta haber recorrido todos los numeros generados.
7. Una vez que hayamos registrado todos los gaps, por cada longitud de gap contamos su frecuencia y lo registramos en una nueva tabla
8. Agrupamos los intervalos de los gaps. Decidimos hacerlo utilizando los grupos $(0,3)$,$(4,7)$,$(8,11)$, $(12,15)$, $(16,19)$, $(20,23)$, y a cada grupo le asignamos su correspondiente frecuencia.
9. A partir de esto generamos una tabla con la frecuencia relativa de cada longitud de gap . Es decir, para cada grupo de gaps que registramos lo asociamos al valor:

$$\frac{GapLength_i}{\sum_{i=1}^{n} GapLength_i}$$

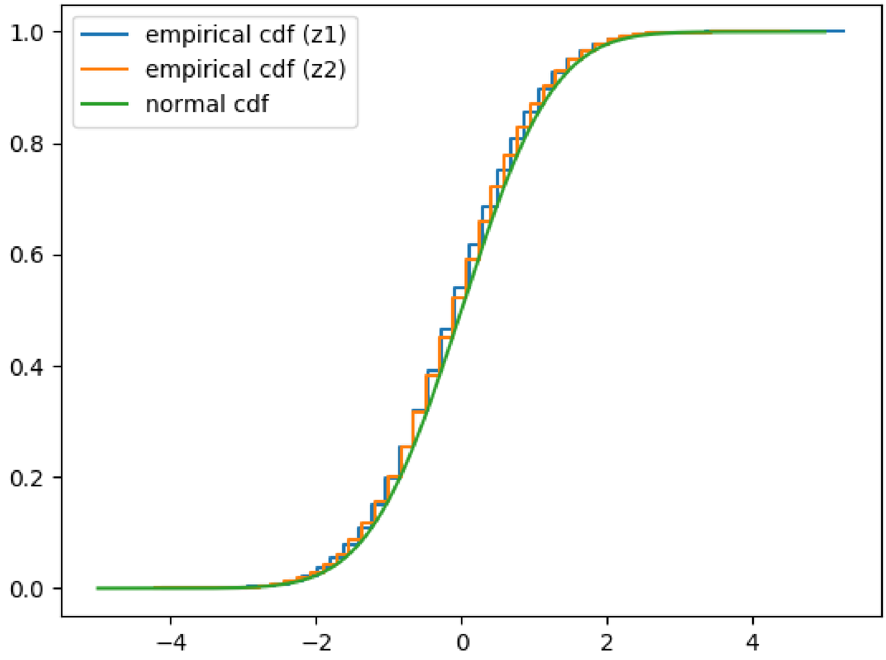
10. Generamos la tabla $Sn(x)$ (la acumulada de la anterior), si a la tabla anterior la llamo A:

$$Sn(x_i) = \sum_{j=1}^{i-1} A_j$$

11. Generamos la tabla $F(x) = 1-0{,}9^{x+1}$ Siendo x el numero mayor de cada grupo de gaps generados anteriormente $(3, 7, 11, ..)$
12. Debemos buscar: $D = Max(Sn_i - F(x_i))$
13. Luego $D_{0,05} = 1{,}36/\sqrt{100000}$
14. Finalmente si $D < D_{0,05}$ aceptamos la hipotesis. No sucedio en nuestro caso por lo que la rechazamos.

3.10. Ejercicio 10

1. Nos apoyamos en scipy.stats, especificamente con la clase kstest
2. Asumimos un nivel de significancia del 0,05 %
3. Ambas muestras pasaron el test de kolmogorov-smirnov



Como podemos observar mediante las distribuciones empiricas, estas dos variables generadas mediante el metodo Box-Muller se aproximan bastante a una normal real.

4. Conclusiones y aclaraciones

El trabajo práctico nos permitió conocer y realizar simulaciones teniendo como base teórica los conceptos explicados en clase . Además, nos permitió conocer herramientas que permiten realizar simulaciones qure son muy utilizadas en el campo científico.

En el ejercicio 3 y 10 si bien podriamos haber trabajado con una unica distribucion generada por el metodo, nos parecio interesante trabajar con ambas ya que son generadas de maneras distintas.

Referencias

[1] Python, Generación de números con distintas distribuciones de probabilidad, https://docs.python.org/3/library/random.html.

[2] Método de Box Muller, https://es.wikipedia.org/wiki/Método_de_Box-Muller.

[3] Probability, Statistics, and Random Processes for Electrical Engineerging,3rd_Ed. Leon-Garcia

A. Código fuente

A.1. Resolución ejercicio 1

```python
#/usr/bin/env/ python
# -*- coding: utf-8 -*-

import matplotlib.pyplot as plt

modulo = 2**32
multiplicador = 1013904223
incremento = 1664525
semilla = int(92702 * 0.15 + 93584 * 0.25 + 98757 * 0.26)
secuencia = [ semilla ]

def GCL( valor ):
        return ( multiplicador * valor + incremento ) % modulo

def cargarSecuencia(secuencia,inicio, fin):
        for i in range(inicio,fin):
                secuencia.append( GCL( secuencia[ i-1 ] ) )

cargarSecuencia(secuencia,1, 5)
print("Primeros 5 numeros de la secuencia: {}".format(secuencia))

#Para que de números entre 0 y 1, divido por su módulo
#Hipótesis: utilizo como semilla el valor: 0.9
secuenciaRango01 = [0.9]

#Cargo la lista de secuencias
cargarSecuencia(secuenciaRango01,1, 100000)

#divido los valores de la lista de secuencias por su modulo
for i in range(0,100000):
        secuenciaRango01[i]= secuenciaRango01[i]/modulo

#histograma
plt.title('Histograma')
plt.xlabel('SecuenciaDeValores')
plt.ylabel('Frecuencia')
plt.hist(secuenciaRango01, bins =60, alpha=0.5, ec='black')
plt.grid(True)
plt.show()
plt.clf()
```

A.2. Resolución ejercicio 2

```python
#/usr/bin/env/ python
# -*- coding: utf-8 -*-

import math
import matplotlib.pyplot as plt
import numpy as np

#Datos del ejercicio anterior
modulo = 2**32
multiplicador = 1013904223
incremento = 1664525
semilla = int(92702 * 0.15 + 93584 * 0.25 + 98757 * 0.26)

secuencia = [semilla]
#Generador del ejercicio anterior, para este en un rango[0,1] se debe dividir por modulo
def GCL( valor ):
        return ( multiplicador * valor + incremento ) % modulo

#Transformada inversa
def transformadaInversa(u):
        return -(float(1)/float(20)) * math.log(1-u)

#Creamos la secuencia utilizando el generador GCL
for i in range(1,100000):
        secuencia.append( GCL( secuencia[ i-1 ] ) )

#Divido los valores de la lista de secuencias por su modulo
for i in range(0,100000):
        secuencia[i]= float(secuencia[i])/float(modulo)

#Aplicamos transformada inversa a la secuencia
for i in range(0,100000):
        secuencia[i]= transformadaInversa(secuencia[i])

#Histograma
plt.title('Histograma Metodo de la transformada inversa')
plt.xlabel('SecuenciaDeValores')
plt.ylabel('Frecuencia')
plt.hist(secuencia, bins =60, alpha=0.5, ec='black')
plt.grid(True)
plt.show()

#Calculo de media
media = np.mean(secuencia)
#Valor simulado de la media
print("El valor simulado de la media es {}".format(media))
#Valor teorico de la media
print("El valor teórico de la media es {}".format(float(1)/float(20)))

#Calculo de varianza
varianza = np.var(secuencia)
print("El valor simulado de la varianza es {}".format(varianza))
print("El valor teórico de la varianza es {}".format(float(1)/float((20))**2))
```

A.3. Resolución ejercicio 3

```python
#/usr/bin/env/ python
# -*- coding: utf-8 -*-

from numpy import random, sqrt, log, sin, cos, pi, mean, var
import matplotlib.pyplot as plt

#Distribución normal
u1 = random.uniform(0,1, 100000)
u2 = random.uniform(0,1, 100000)

#Box muller
z1 = sqrt(-2*log(u1))*cos(2*pi*u2)
z2 = sqrt(-2*log(u1))*sin(2*pi*u2)

#Histogramas
fig, ax = plt.subplots(1,2, figsize=(20, 10))
ax[0].hist(z1, bins =60, alpha=0.5, ec='black', label = "z1")
ax[0].hist(z2, bins =60, alpha=0.5, ec='green', label = "z2")
ax[1].hist(random.normal(0, 1, 100000), bins =60, alpha=0.5, ec='red')
ax[0].title.set_text('Histograma Box Muller')
ax[1].title.set_text('Histograma Normal estandar')
plt.show()

#Calculo de media
#Valor simulado de la media
print("El valor simulado de la media z1 es {}".format(mean(z1)))
print("El valor simulado de la media z2 es {}".format(mean(z2)))
#Valor teorico de la media
print("El valor teórico de la media es {}".format(0))

#Calculo de varianza
print("El valor simulado de la varianza z1 es {}".format(var(z1)))
print("El valor simulado de la varianza z2 es {}".format(var(z2)))
print("El valor teórico de la varianza es {}".format(1))
```

A.4. Resolución ejercicio 4

```
import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plt

def p(x):
    return st.norm.pdf(x, loc=40, scale=6)


def q(x):
    return st.norm.pdf(x, loc=50, scale= 20)


x = np.arange(-50, 151)
k = max(p(x) / q(x))


def rejection_sampling(iter=1000):
    samples = []

    for i in range(iter):
        z = np.random.normal(50, 20)
        u = np.random.uniform(0, k*q(z))

        if u <= p(z):
            samples.append(z)

    return np.array(samples)


if __name__ == '__main__':
    plt.plot(x, p(x), label = 'target distribution')
    plt.plot(x, k*q(x), label = 'proposed distribution')
    plt.legend(loc='upper left')
    plt.show()
    print('Rejection sampling in progress..')
    s = rejection_sampling(iter=100000)
    mean = round(s.mean(), 2)
    var = round(s.var(), 2)
    print('La media de la muestra generada es: '+str(mean)+', y la varianza: '+str(
    var))
    plt.plot(x, p(x), label = 'target distribution', color = 'red')
    plt.hist(s, label = 'histogram rejection sampling', normed=True)
    plt.legend(loc='upper left')
    plt.show()
```

A.5. Resolución ejercicio 5

```
#/usr/bin/env/ python
# -*- coding: utf-8 -*-
import math
import numpy as np
import matplotlib.pyplot as plt

## datos y GCL del ejercicio 1
modulo = 2**32
multiplicador = 1013904223
incremento = 1664525
semilla = int(92702 * 0.15 + 93584 * 0.25 + 98757 * 0.26)
secuencia = [ semilla ]

def GCL( valor ):
        return ( multiplicador * valor + incremento ) % modulo

def funcionInversa( valoresFuncion, secuencia ):
        for nro in secuencia:
            if (nro >= 0 and nro < 0.4):
                valoresFuncion.append(1)
            elif (nro >= 0.4 and nro < 0.7):
                valoresFuncion.append(2)
            elif (nro >= 0.7 and nro < 0.82):
                valoresFuncion.append(3)
            elif (nro >= 0.82 and nro < 0.92):
                valoresFuncion.append(4)
            else:
                valoresFuncion.append(5)

        return valoresFuncion

def modularSecuencia(secuencia):
        for i in range(1,100000):
                secuencia[i] = secuencia[i]/modulo
        return secuencia

def generarGCL(secuencia):
        for i in range(1,100000):
                secuencia.append( GCL( secuencia[ i-1 ] ) )
        return secuencia

def crearHistograma(valores):
        plt.title('Histograma')
        plt.xlabel('Valores de la funcion')
        plt.ylabel('Frecuencia')
        plt.hist(valores, bins = 10, alpha=0.5, ec='black')
        plt.grid(True)
        plt.show()

def correrEjercicio():
        ## genero 100000 valores utilizando GCL
        secuenciaGCL = generarGCL(secuencia)
        ## divido por su modulo para tener valores (0,1)
        secuenciaModulada = modularSecuencia(secuenciaGCL)
        valoresFuncion = []
        #La función inversa de la Función de distribución Empírica es:
        valores = funcionInversa( valoresFuncion, secuenciaModulada )
        #histograma
        #crearHistograma(valores)
        return valores

```

```
correrEjercicio()
```

A.6. Resolución ejercicio 6

```
#/usr/bin/env/ python
# -*- coding: utf-8 -*-

import matplotlib.pyplot as plt
import random

#Generador de números aleatorios que provee PYTHON
def aleatorio():
        return random.uniform(-1,1)

listaDeValores1=[]
listaDeValores2=[]

#Genero 1000 valores(por ejemplo)
for i in range(0,10000):
        x = aleatorio()
        y = aleatorio()
        if ( x**2 + y**2) < 1:
                listaDeValores1.append(x)
                listaDeValores2.append(y)

#Gráfico
plt.title('Grafico utilizando una distribucion uniforme')
plt.plot(listaDeValores1,listaDeValores2,'o',markersize=1)
plt.xlabel('Valores de X')
plt.ylabel('Valores de Y')
plt.show()
```

A.7. Resolución ejercicio 7

```python
#/usr/bin/env/ python
# -*- coding: utf-8 -*-

import matplotlib.pyplot as plt

modulo = 2**32
multiplicador = 1013904223
incremento = 1664525
semilla = int(92702 * 0.15 + 93584 * 0.25 + 98757 * 0.26)
secuencia = [ semilla ]

def GCL( valor ):
        return ( multiplicador * valor + incremento ) % modulo

def cargarSecuencia(secuencia,inicio, fin):
        for i in range(inicio,fin):
                secuencia.append( GCL( secuencia[ i-1 ] ) )


#Cargo la lista de secuencias
cargarSecuencia(secuencia,1, 100000)

#Gráfico en dos dimensiones
plt.specgram(secuencia, NFFT=256, Fs=2, Fc=0,noverlap=128)

#Gráfico en tres dimensiones

plt.show()
```

A.8. Resolución ejercicio 8

```python
#/usr/bin/env/ python
# -*- coding: utf-8 -*-
import scipy.stats as stats
from numpy import random
from TPEj5 import correrEjercicio

#Tomamos los resultados del ejercicio 5
valoresObservados = correrEjercicio()
nMUestras = 100000

esperados = []
cant_valores = 100000
dispersionCuadrado = 0
uniforme = random.uniform(0,1,100000)

#Generamos la muestra esperada n*p_i
for i in range(cant_valores):
    esperado = nMUestras * float(uniforme[i])
    esperados.append(float(esperado))

#Medimos la dispersion de las ocurrencias obervadas(N_i) respectos de las esperadas n
    *p_i
for i in range(1,cant_valores):
    dispersionCuadrado += ((valoresObservados [i] - esperados[i]) ** 2) / esperados[i
    ]

#Calculamos los grados de libertad de la distribucion Chi-2

if (dispersionCuadrado < t):
        print(' Se acepta la hipotesis con la distribucion empirica con un error del
    1% ')
else:
        print('Se rechaza la hipotesis con la distribucion empirica con un error del
    1% ')
```

A.9. Resolución ejercicio 9

```python

#Generador Ej1

import math
import numpy

modulo = 2**32
multiplicador = 1013904223
incremento = 1664525
semilla = int(92702 * 0.15 + 93584 * 0.25 + 98757 * 0.26)
secuencia = [ semilla ]

def GCL( valor ):
        return ( multiplicador * valor + incremento ) % modulo

for i in range(1,5):
        secuencia.append( GCL( secuencia[ i-1 ] ) )

secuenciaRango01 = [0.9]

for i in range(1,100000):
        secuenciaRango01.append( GCL( secuenciaRango01[i-1]) )

for i in range(0,100000):
        secuenciaRango01[i]= secuenciaRango01[i]/modulo

#Gap test
from collections import Counter

# Intervalo (enunciado)
a = 0.2
b = 0.5

# Cuento cada cuantos numeros aparece un numero de este intervalo y lo registro en un
    array
# Repito hasta recorrer todos los numeros generados

gaps = []

actual_gap = 0

for i in range(0, len(secuenciaRango01)):
  numero_generado = secuenciaRango01[i]
  if a <= numero_generado <= b:
    gaps.append(actual_gap)
    actual_gap = 0
  else:
    actual_gap += 1

frecuencias_gaps = Counter(gaps)

#el maximo gap es 23, separo en bins de 3
bins = [(0,3) , (4,7), (8,11), (12,15), (16,19), (20,23)]
bins_ocurrencias = {(0,3) : 0, (4,7): 0, (8,11): 0, (12,15): 0, (16,19): 0, (20,23):
    0}
# por cada gap en frecuencias_gap, le sumo su resultado al bin correspondiente
for gap in frecuencias_gaps:
  for interval in bins:
    start = interval[0]
    finish = interval[1]
    if start <= gap <= finish:
```

```python
      bins_ocurrencias[interval] += frecuencias_gaps[gap]

#Testeo que este for ande
assert bins_ocurrencias[(0,3)] == frecuencias_gaps[0] + frecuencias_gaps[1] + frecuencias_gaps[2] + frecuencias_gaps[3]
assert bins_ocurrencias[(4,7)] == frecuencias_gaps[4] + frecuencias_gaps[5] + frecuencias_gaps[6] + frecuencias_gaps[7]

#Ahora calculo las frecuencias relativas de cada bin de gaps

total = sum(bins_ocurrencias.values())
#calculo las frecuencias relativas
bins_frecuencias_relativas = {k: v/total for k, v in bins_ocurrencias.items()}

#calculo las frecuencias relativas acumuladas
bins_frecuencias_relativas_acumuladas = {}
bins_frecuencias_relativas_acumuladas[(0,3)] = bins_frecuencias_relativas[(0,3)]
bins_frecuencias_relativas_acumuladas[(4,7)] = bins_frecuencias_relativas[(4,7)] + bins_frecuencias_relativas_acumuladas[(0,3)]
bins_frecuencias_relativas_acumuladas[(8,11)] = bins_frecuencias_relativas[(8,11)] + bins_frecuencias_relativas_acumuladas[(4,7)]
bins_frecuencias_relativas_acumuladas[(12,15)] = bins_frecuencias_relativas[(12,15)] + bins_frecuencias_relativas_acumuladas[(8,11)]
bins_frecuencias_relativas_acumuladas[(16,19)] = bins_frecuencias_relativas[(16,19)] + bins_frecuencias_relativas_acumuladas[(12,15)]
bins_frecuencias_relativas_acumuladas[(20,23)] = bins_frecuencias_relativas[(20,23)] + bins_frecuencias_relativas_acumuladas[(16,19)]

#testeo que esto este acumulando bien
assert bins_frecuencias_relativas_acumuladas[(20,23)] == 1

#Calculo FX de cada bin (1 - (0.9)**x+1) siendo x el segundo valor de la tupla (por ej para (0,3) es 1 - (0.9)**4)
FX_bins = {}
for interval in bins:
  finish = interval[1]
  FX_bins[interval] = 1 - ((0.9)**(finish + 1))

#Ahora resto los valores de bins_frecuencias_relativas_acumuladas a FX_bins
res = {}
for k in FX_bins.keys():
  res[k] = bins_frecuencias_relativas_acumuladas[k] - FX_bins[k]
#Falta seguir los ultimos pasos que hace este chabon: https://www.youtube.com/watch?v=xh-4i0v-0yk

#Step 4
max_value = max(res.values())

#Step 5 asumo alpha 0.05

n = 100000

D = 1.36/math.sqrt(n)
if max_value > D:
  print('La muestra es rechazada por el GAP test')
else:
  print('La hipotesis no es rechazada')
```

A.10. Resolución ejercicio 10

```python
#Distribucion generada en el ejercicio 3

#/usr/bin/env/ python
# -*- coding: utf-8 -*-
import numpy as np
from numpy import random, sqrt, log, sin, cos, pi
import matplotlib.pyplot as plt
import scipy.stats
import statsmodels.api as sm # recommended import according to the docs

#Distribución uniforme
u1 = random.uniform(0,1, 100000)
u2 = random.uniform(0,1, 100000)

#Box muller transformation
z1 = sqrt(-2*log(u1))*cos(2*pi*u2)
z2 = sqrt(-2*log(u1))*sin(2*pi*u2)

#Ejercicio 10:

#Aplico el test a las dos distribuciones generadas


test1 = scipy.stats.kstest(z1, 'norm')
test2 = scipy.stats.kstest(z2, 'norm')

#Hipotesis nula (de igualdad): Si no la rechazamos podemos decir que es igual a una normal
#Hipotesis alternativa (de diferencias): Si rechazamos la hipotesis nula decimos que hay diferencias con la distribucion normal
#Con un nivel de significancia del 0,05%

if test1.pvalue >= 0.05:
        print('La variable z1 pasa el test de kolmogorov-smirnov para un nivel de significancia del 0.05%, por lo tanto no podemos rechazar la hipotesis nula y la distribucion de esta variable es igual a la distribucion normal')
else:
        print('La variable z1 no pasa el test de kolmogorov-smirnov y afirmamos que tiene diferencias con la distribucion normal')
if test2.pvalue >= 0.05:
        print('La variable z2 pasa el test de kolmogorov-smirnov para un nivel de significancia del 0.05%, por lo tanto no podemos rechazar la hipotesis nula y la distribucion de esta variable es igual a la distribucion normal')
else:
        print('La variable z2 no pasa el test de kolmogorov-smirnov y afirmamos que tiene diferencias con la distribucion normal')


#Grafico de la empirica de z1
sample = z1
ecdf = sm.distributions.ECDF(sample)
x = np.linspace(min(sample), max(sample))
y = ecdf(x)
plt.step(x, y, label = 'empirical cdf (z1)')

#Grafico de la empirica de z2
sample = z2
ecdf = sm.distributions.ECDF(sample)
x = np.linspace(min(sample), max(sample))
y = ecdf(x)
plt.step(x, y, label = 'empirical cdf (z2)')
```

```python

#Grafico la acumulada de la normal
x = np.linspace(-5, 5, 5000)
mu = 0
sigma = 1
y_cdf = scipy.stats.norm.cdf(x, mu, sigma) # the normal cdf
plt.plot(x, y_cdf, label='normal cdf')


plt.legend()
plt.show()
```