

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

Índice	1
1. Enunciado del trabajo práctico	2
2. Introducción	4
3. Implementación y resultados	4
3.1. Ejercicio 1	4
3.2. Ejercicio 2	4
3.3. Ejercicio 3	5
3.4. Ejercicio 4	7
3.5. Ejercicio 5	9
3.6. Ejercicio 6	9
3.7. Ejercicio 7	10
3.8. Ejercicio 8	10
3.9. Ejercicio 9	10
3.10. Ejercicio 10	11
4. Conclusiones y aclaraciones	13
Referencias	13
A. Código fuente	14
A.1. Resolución ejercicio 1	14
A.2. Resolución ejercicio 2	15
A.3. Resolución ejercicio 3	16
A.4. Resolución ejercicio 4	17
A.5. Resolución ejercicio 5	18
A.6. Resolución ejercicio 6	20
A.7. Resolución ejercicio 7	21
A.8. Resolución ejercicio 8	22
A.9. Resolución ejercicio 9	23
A.10. Resolución ejercicio 10	25

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 números 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 congruencial 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), Transformada inversa y tests como Test espectral y Kolmogorov-Smirnov. Los ejercicios están simulados en lenguaje Python.

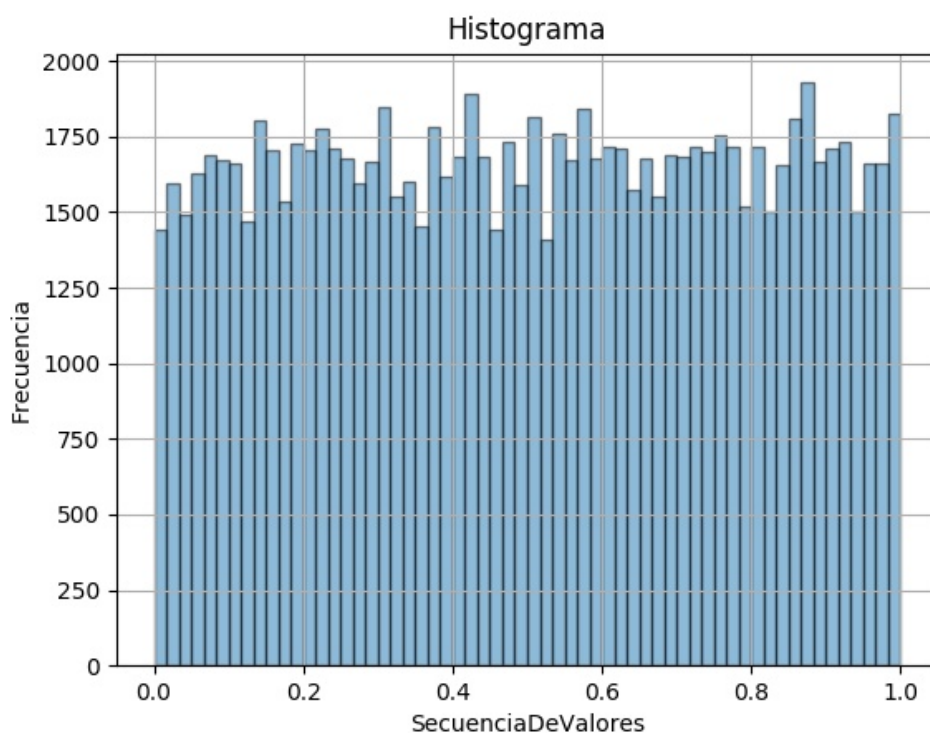
3. Implementación y resultados

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

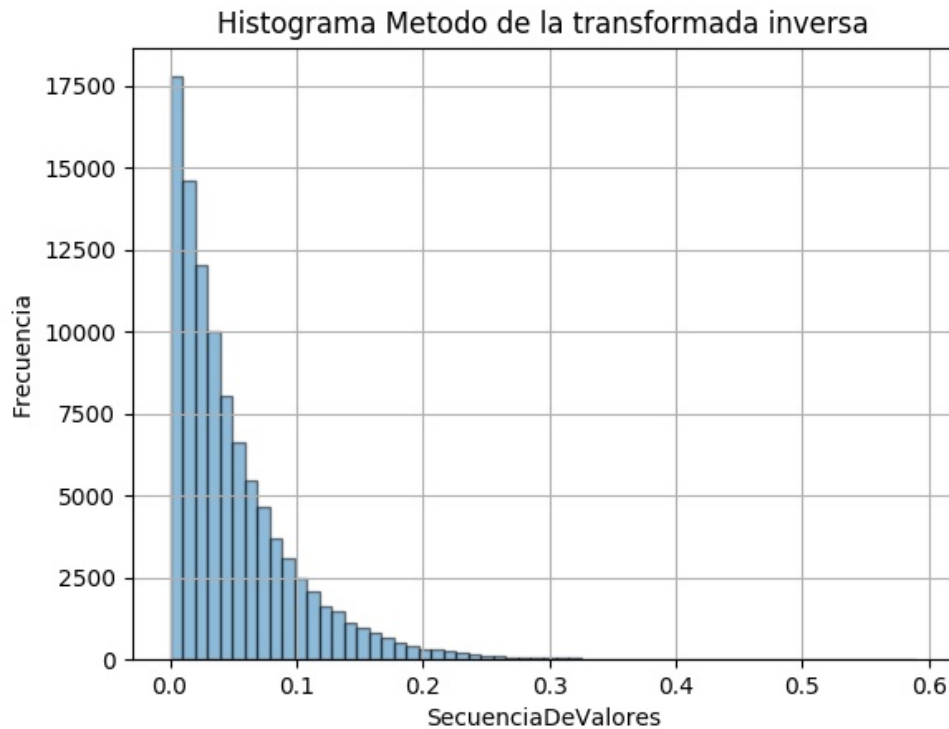
El resultado de los primeros 5 números de la secuencia: [62978, 383030987L, 2740587618L, 1650525291L, 2470812354L]

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

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

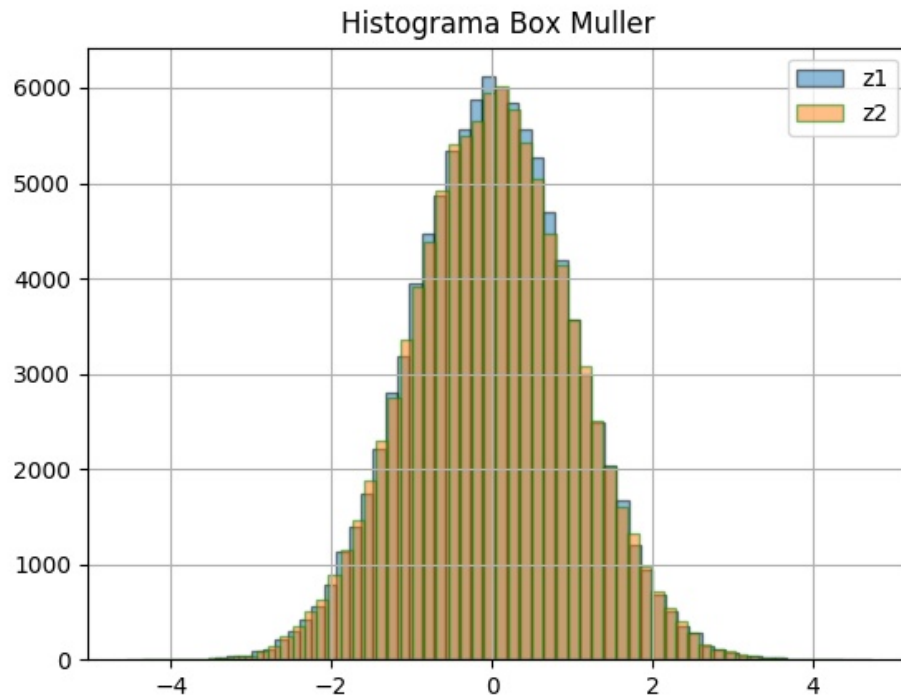
Por lo tanto podemos observar que los valores simulados y teóricos son bastante parecidos.

3.3. Ejercicio 3

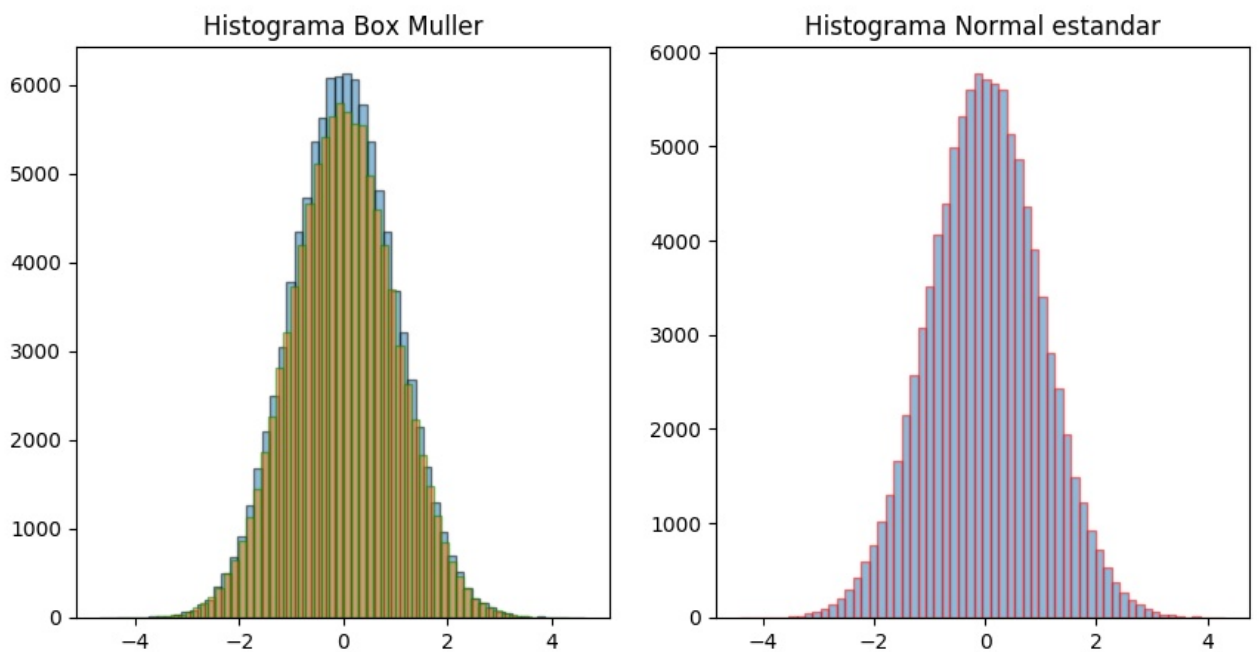
Los resultados que obtenemos son los siguientes:

- El valor simulado de la media z_1 es 0.000743926097041
- El valor simulado de la media z_2 es -0.0016462929471
- El valor teórico de la media es 0
- El valor simulado de la varianza z_1 es 0.998271254875
- El valor simulado de la varianza z_2 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:

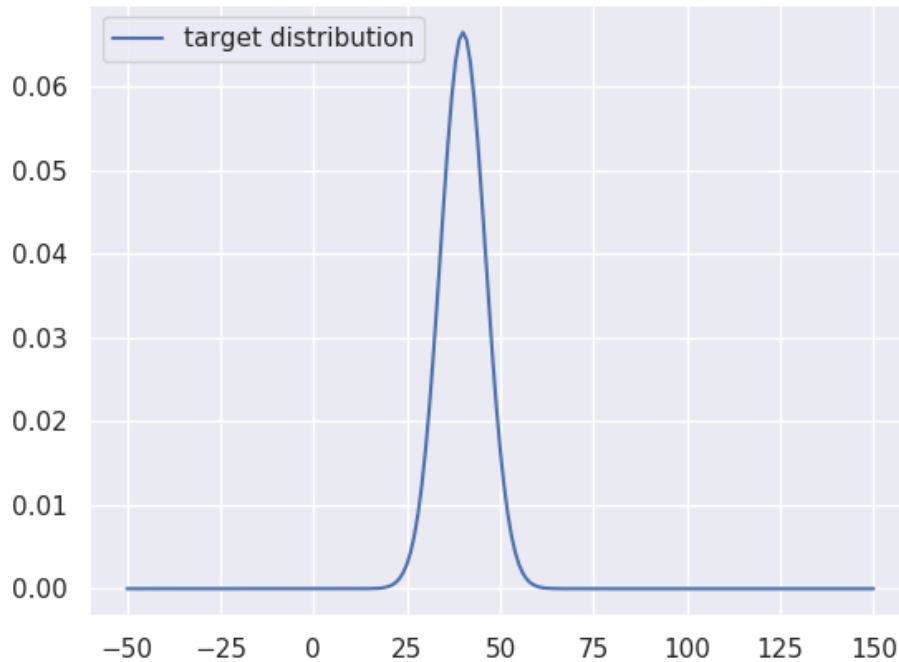


Por lo tanto podemos observar que se comprueba que el método de Box Muller es una distribución normal estándar.

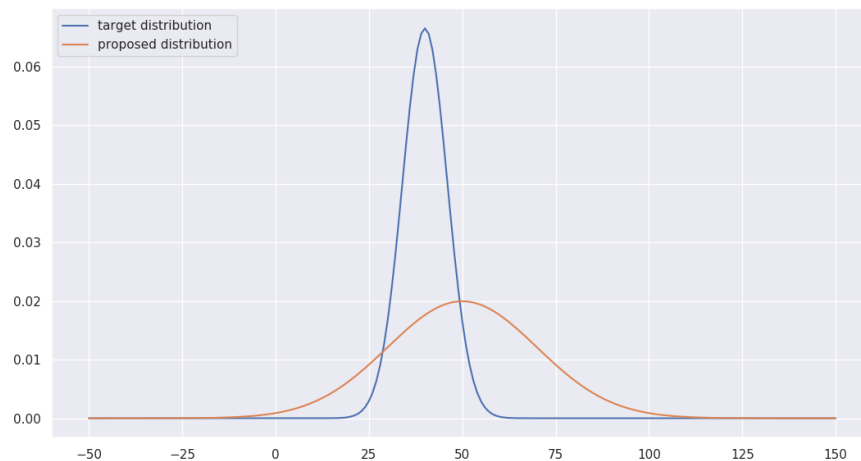
3.4. Ejercicio 4

Seguimos los pasos de esta guía: Rejection Sampling.

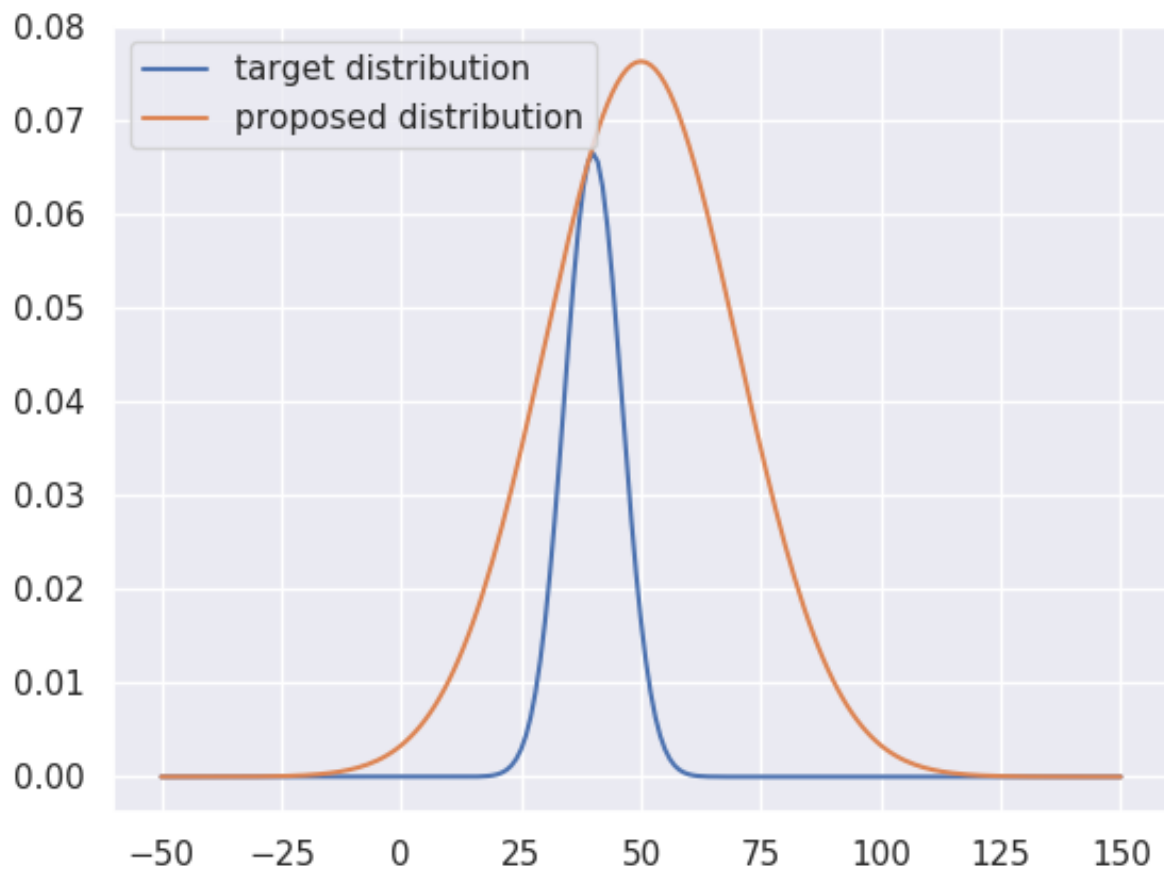
La distribución target $P(x)$ es una $N(40, 6)$



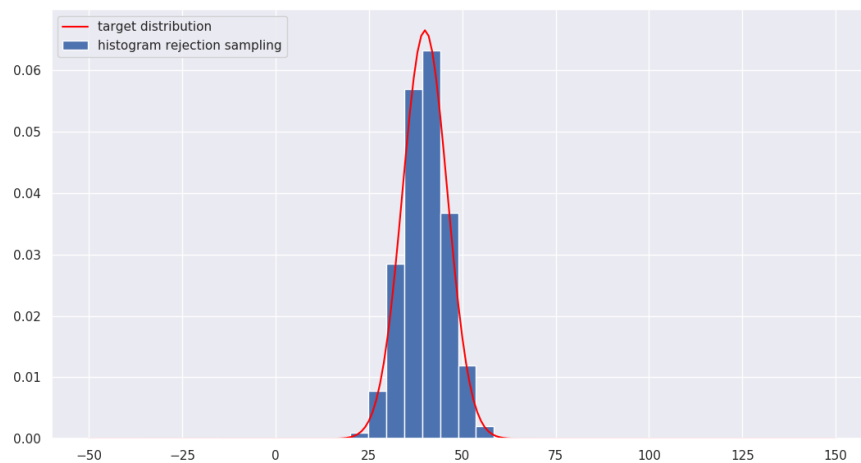
Proponemos $N(50, 30)$ como $Q(x)$



Rejection Sampling va a fallar debido a que la $Q(x)$ propuesta no está envolviendo a nuestra distribución target $P(x)$. Para remediar esto, encontramos un factor k que multiplique a $Q(x)$ que es la relación máxima entre $P(x)$ y $Q(x)$, entonces: $k = \max(P(x)/Q(x))$ for all x . Al reescalar..



La distribución generada mediante rejection sampling es la siguiente:



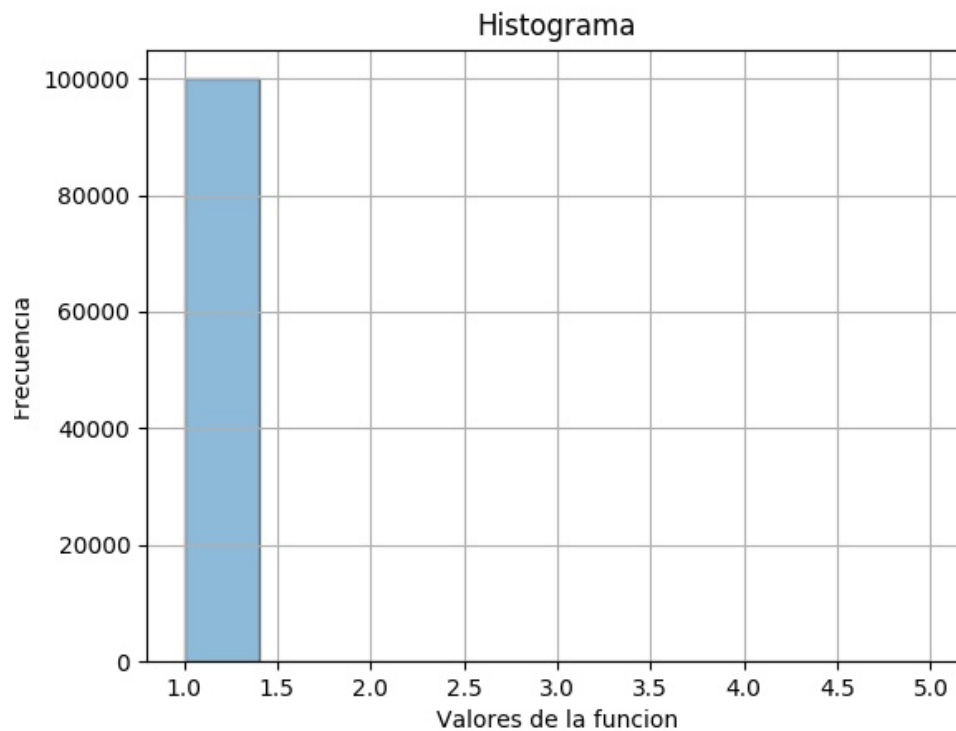
Este es el gráfico de la distribución real vs el histograma de valores generados por el algoritmo. Como podemos ver, el algoritmo logra generar una aproximación bastante buena.

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.

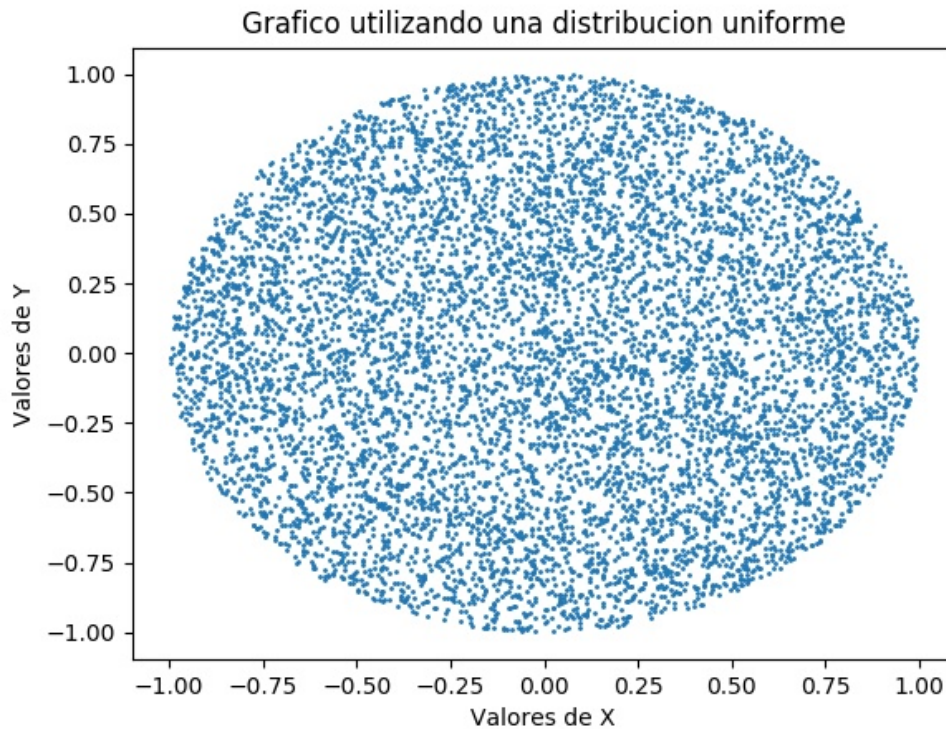
3.5. Ejercicio 5

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



3.6. Ejercicio 6

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 χ^2 se aplica el método utilizando los siguientes pasos:

1. Utilizamos la distribución empírica generada en el ejercicio 5). Esta 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:

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

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:

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

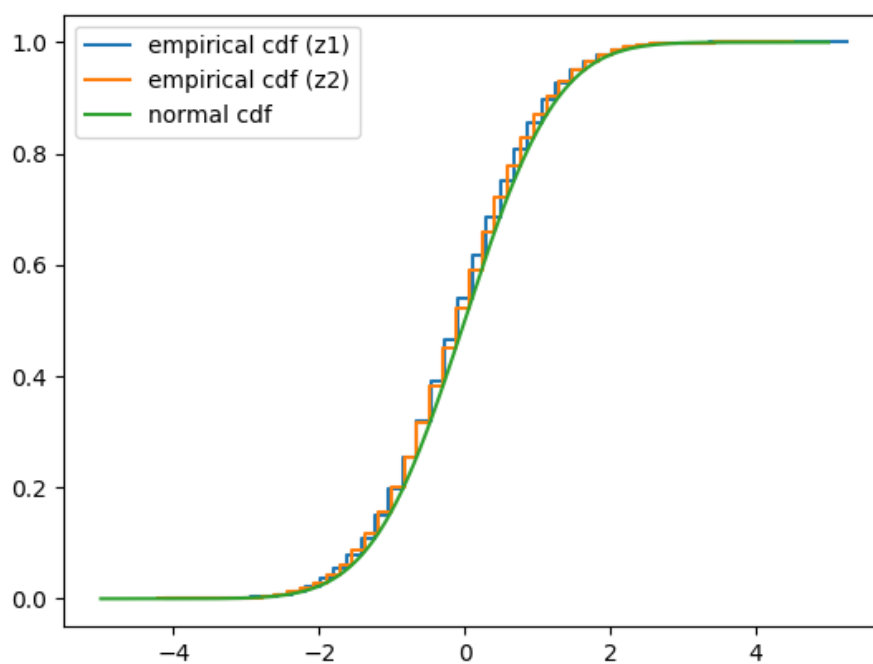
10. Generamos la tabla $Sn(x)$ (la acumulada de la anterior), si a la tabla anterior la llamo A:

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

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 que 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 Engineering, 3rd Ed. Leon-Garcia

A. Código fuente

A.1. Resolución ejercicio 1

```
1  #/usr/bin/env/ python
2  # -*- coding: utf-8 -*-
3
4  import matplotlib.pyplot as plt
5
6  modulo = 2**32
7  multiplicador = 1013904223
8  incremento = 1664525
9  semilla = int(92702 * 0.15 + 93584 * 0.25 + 98757 * 0.26)
10 secuencia = [ semilla ]
11
12 def GCL( valor ):
13     return ( multiplicador * valor + incremento ) % modulo
14
15 def cargarSecuencia(secuencia,inicio, fin):
16     for i in range(inicio,fin):
17         secuencia.append( GCL( secuencia[ i-1 ] ) )
18
19 cargarSecuencia(secuencia,1, 5)
20 print("Primeros 5 numeros de la secuencia: {}".format(secuencia))
21
22 #Para que de números entre 0 y 1, dividido por su módulo
23 #Hipótesis: utilizo como semilla el valor: 0.9
24 secuenciaRango01 = [0.9]
25
26 #Cargo la lista de secuencias
27 cargarSecuencia(secuenciaRango01,1, 100000)
28
29 #divido los valores de la lista de secuencias por su modulo
30 for i in range(0,100000):
31     secuenciaRango01[i]= secuenciaRango01[i]/modulo
32
33 #histograma
34 plt.title('Histograma')
35 plt.xlabel('SecuenciaDeValores')
36 plt.ylabel('Frecuencia')
37 plt.hist(secuenciaRango01, bins =60, alpha=0.5, ec='black')
38 plt.grid(True)
39 plt.show()
40 plt.clf()
```

A.2. Resolución ejercicio 2

```

1  #/usr/bin/env/ python
2  # -*- coding: utf-8 -*-
3
4  import math
5  import matplotlib.pyplot as plt
6  import numpy as np
7
8  #Datos del ejercicio anterior
9  modulo = 2**32
10 multiplicador = 1013904223
11 incremento = 1664525
12 semilla = int(92702 * 0.15 + 93584 * 0.25 + 98757 * 0.26)
13
14 secuencia = [semilla]
15 #Generador del ejercicio anterior, para este en un rango[0,1] se debe dividir por
    modulo
16 def GCL( valor ):
17     return ( multiplicador * valor + incremento ) % modulo
18
19 #Transformada inversa
20 def transformadaInversa(u):
21     return -(float(1)/float(20)) * math.log(1-u)
22
23 #Creamos la secuencia utilizando el generador GCL
24 for i in range(1,100000):
25     secuencia.append( GCL( secuencia[ i-1 ] ) )
26
27 #Divido los valores de la lista de secuencias por su modulo
28 for i in range(0,100000):
29     secuencia[i]= float(secuencia[i])/float(modulo)
30
31 #Aplicamos transformada inversa a la secuencia
32 for i in range(0,100000):
33     secuencia[i]= transformadaInversa(secuencia[i])
34
35 #Histograma
36 plt.title('Histograma Metodo de la transformada inversa')
37 plt.xlabel('SecuenciaDeValores')
38 plt.ylabel('Frecuencia')
39 plt.hist(secuencia, bins =60, alpha=0.5, ec='black')
40 plt.grid(True)
41 plt.show()
42
43 #Calculo de media
44 media = np.mean(secuencia)
45 #Valor simulado de la media
46 print("El valor simulado de la media es {}".format(media))
47 #Valor teorico de la media
48 print("El valor teórico de la media es {}".format(float(1)/float(20)))
49
50 #Calculo de varianza
51 varianza = np.var(secuencia)
52 print("El valor simulado de la varianza es {}".format(varianza))
53 print("El valor teórico de la varianza es {}".format(float(1)/float((20)**2)))

```


A.3. Resolución ejercicio 3

```
1  #/usr/bin/env/ python
2  # -*- coding: utf-8 -*-
3
4  from numpy import random, sqrt, log, sin, cos, pi, mean, var
5  import matplotlib.pyplot as plt
6
7  #Distribución normal
8  u1 = random.uniform(0,1, 100000)
9  u2 = random.uniform(0,1, 100000)
10
11 #Box muller
12 z1 = sqrt(-2*log(u1))*cos(2*pi*u2)
13 z2 = sqrt(-2*log(u1))*sin(2*pi*u2)
14
15 #Histogramas
16 fig, ax = plt.subplots(1,2, figsize=(20, 10))
17 ax[0].hist(z1, bins =60, alpha=0.5, ec='black', label = "z1")
18 ax[0].hist(z2, bins =60, alpha=0.5, ec='green', label = "z2")
19 ax[1].hist(random.normal(0, 1, 100000), bins =60, alpha=0.5, ec='red')
20 ax[0].title.set_text('Histograma Box Muller')
21 ax[1].title.set_text('Histograma Normal estandar')
22 plt.show()
23
24 #Calculo de media
25 #Valor simulado de la media
26 print("El valor simulado de la media z1 es {}".format(mean(z1)))
27 print("El valor simulado de la media z2 es {}".format(mean(z2)))
28 #Valor teorico de la media
29 print("El valor teórico de la media es {}".format(0))
30
31 #Calculo de varianza
32 print("El valor simulado de la varianza z1 es {}".format(var(z1)))
33 print("El valor simulado de la varianza z2 es {}".format(var(z2)))
34 print("El valor teórico de la varianza es {}".format(1))
```

A.4. Resolución ejercicio 4

```
1  import numpy as np
2  import scipy.stats as st
3  import matplotlib.pyplot as plt
4
5  def p(x):
6      return st.norm.pdf(x, loc=40, scale=6)
7
8
9  def q(x):
10     return st.norm.pdf(x, loc=50, scale= 20)
11
12
13  x = np.arange(-50, 151)
14  k = max(p(x) / q(x))
15
16
17  def rejection_sampling(iter=1000):
18     samples = []
19
20     for i in range(iter):
21         z = np.random.normal(50, 20)
22         u = np.random.uniform(0, k*q(z))
23
24         if u <= p(z):
25             samples.append(z)
26
27     return np.array(samples)
28
29
30  if __name__ == '__main__':
31     plt.plot(x, p(x), label = 'target distribution')
32     plt.plot(x, k*q(x), label = 'proposed distribution')
33     plt.legend(loc='upper left')
34     plt.show()
35     print('Rejection sampling in progress..')
36     s = rejection_sampling(iter=100000)
37     mean = round(s.mean(), 2)
38     var = round(s.var(), 2)
39     print('La media de la muestra generada es: '+str(mean)+' , y la varianza: '+str(
40         var))
41     plt.plot(x, p(x), label = 'target distribution', color = 'red')
42     plt.hist(s, label = 'histogram rejection sampling', normed=True)
43     plt.legend(loc='upper left')
44     plt.show()
```

A.5. Resolución ejercicio 5

```

1  #/usr/bin/env/ python
2  # -*- coding: utf-8 -*-
3  import math
4  import numpy as np
5  import matplotlib.pyplot as plt
6
7  ## datos y GCL del ejercicio 1
8  modulo = 2**32
9  multiplicador = 1013904223
10 incremento = 1664525
11 semilla = int(92702 * 0.15 + 93584 * 0.25 + 98757 * 0.26)
12 secuencia = [ semilla ]
13
14 def GCL( valor ):
15     return ( multiplicador * valor + incremento ) % modulo
16
17 def funcionInversa( valoresFuncion, secuencia ):
18     for nro in secuencia:
19         if (nro >= 0 and nro < 0.4):
20             valoresFuncion.append(1)
21         elif (nro >= 0.4 and nro < 0.7):
22             valoresFuncion.append(2)
23         elif (nro >= 0.7 and nro < 0.82):
24             valoresFuncion.append(3)
25         elif (nro >= 0.82 and nro < 0.92):
26             valoresFuncion.append(4)
27         else:
28             valoresFuncion.append(5)
29
30     return valoresFuncion
31
32 def modularSecuencia(secuencia):
33     for i in range(1,100000):
34         secuencia[i] = secuencia[i]/modulo
35     return secuencia
36
37 def generarGCL(secuencia):
38     for i in range(1,100000):
39         secuencia.append( GCL( secuencia[ i-1 ] ) )
40     return secuencia
41
42 def crearHistograma(valores):
43     plt.title('Histograma')
44     plt.xlabel('Valores de la funcion')
45     plt.ylabel('Frecuencia')
46     plt.hist(valores, bins = 10, alpha=0.5, ec='black')
47     plt.grid(True)
48     plt.show()
49
50 def correrEjercicio():
51     ## genero 100000 valores utilizando GCL
52     secuenciaGCL = generarGCL(secuencia)
53     ## dividido por su modulo para tener valores (0,1)
54     secuenciaModulada = modularSecuencia(secuenciaGCL)
55     valoresFuncion = []
56     #La función inversa de la Función de distribución Empírica es:
57     valores = funcionInversa( valoresFuncion, secuenciaModulada )
58     #histograma
59     #crearHistograma(valores)
60     return valores
61

```

62 `correrEjercicio()`

A.6. Resolución ejercicio 6

```
1  #/usr/bin/env/ python
2  # -*- coding: utf-8 -*-
3
4  import matplotlib.pyplot as plt
5  import random
6
7  #Generador de números aleatorios que provee PYTHON
8  def aleatorio():
9      return random.uniform(-1,1)
10
11  listaDeValores1=[]
12  listaDeValores2=[]
13
14  #Genero 1000 valores(por ejemplo)
15  for i in range(0,10000):
16      x = aleatorio()
17      y = aleatorio()
18      if ( x**2 + y**2) < 1:
19          listaDeValores1.append(x)
20          listaDeValores2.append(y)
21
22  #Gráfico
23  plt.title('Grafico utilizando una distribucion uniforme')
24  plt.plot(listaDeValores1,listaDeValores2,'o',markersize=1)
25  plt.xlabel('Valores de X')
26  plt.ylabel('Valores de Y')
27  plt.show()
```

A.7. Resolución ejercicio 7

```
1  #/usr/bin/env/ python
2  # -*- coding: utf-8 -*-
3
4  import matplotlib.pyplot as plt
5
6  modulo = 2**32
7  multiplicador = 1013904223
8  incremento = 1664525
9  semilla = int(92702 * 0.15 + 93584 * 0.25 + 98757 * 0.26)
10 secuencia = [ semilla ]
11
12 def GCL( valor ):
13     return ( multiplicador * valor + incremento ) % modulo
14
15 def cargarSecuencia(secuencia,inicio, fin):
16     for i in range(inicio,fin):
17         secuencia.append( GCL( secuencia[ i-1 ] ) )
18
19
20 #Cargo la lista de secuencias
21 cargarSecuencia(secuencia,1, 100000)
22
23 #Gráfico en dos dimensiones
24 plt.specgram(secuencia, NFFT=256, Fs=2, Fc=0,noverlap=128)
25
26 #Gráfico en tres dimensiones
27
28 plt.show()
```

A.8. Resolución ejercicio 8

```
1  #/usr/bin/env/ python
2  # -*- coding: utf-8 -*-
3  import scipy.stats as stats
4  from numpy import random
5  from TPEj5 import correrEjercicio
6
7  #Tomamos los resultados del ejercicio 5
8  valoresObservados = correrEjercicio()
9  nMUestras = 100000
10
11  esperados = []
12  cant_valores = 100000
13  dispersionCuadrado = 0
14  uniforme = random.uniform(0,1,100000)
15
16  #Generamos la muestra esperada n*p_i
17  for i in range(cant_valores):
18      esperado = nMUestras * float(uniforme[i])
19      esperados.append(float(esperado))
20
21  #Medimos la dispersion de las ocurrencias observadas(N_i) respecto de las esperadas n
    *p_i
22  for i in range(1,cant_valores):
23      dispersionCuadrado += ((valoresObservados [i] - esperados[i]) ** 2) / esperados[i]
24
25  #Calculamos los grados de libertad de la distribucion Chi-2
26
27  if (dispersionCuadrado < t):
28      print(' Se acepta la hipotesis con la distribucion empirica con un error del
        1% ')
29  else:
30      print('Se rechaza la hipotesis con la distribucion empirica con un error del
        1% ')
```

A.9. Resolución ejercicio 9

```

1
2 #Generador Ej1
3
4 import math
5 import numpy
6
7 modulo = 2**32
8 multiplicador = 1013904223
9 incremento = 1664525
10 semilla = int(92702 * 0.15 + 93584 * 0.25 + 98757 * 0.26)
11 secuencia = [ semilla ]
12
13 def GCL( valor ):
14     return ( multiplicador * valor + incremento ) % modulo
15
16 for i in range(1,5):
17     secuencia.append( GCL( secuencia[ i-1 ] ) )
18
19 secuenciaRango01 = [0.9]
20
21 for i in range(1,100000):
22     secuenciaRango01.append( GCL( secuenciaRango01[i-1]) )
23
24 for i in range(0,100000):
25     secuenciaRango01[i]= secuenciaRango01[i]/modulo
26
27 #Gap test
28 from collections import Counter
29
30 # Intervalo (enunciado)
31 a = 0.2
32 b = 0.5
33
34 # Cuento cada cuantos numeros aparece un numero de este intervalo y lo registro en un
    array
35 # Repito hasta recorrer todos los numeros generados
36
37 gaps = []
38
39 actual_gap = 0
40
41 for i in range(0, len(secuenciaRango01)):
42     numero_generado = secuenciaRango01[i]
43     if a <= numero_generado <= b:
44         gaps.append(actual_gap)
45         actual_gap = 0
46     else:
47         actual_gap += 1
48
49 frecuencias_gaps = Counter(gaps)
50
51 #el maximo gap es 23, separo en bins de 3
52 bins = [(0,3) , (4,7), (8,11), (12,15), (16,19), (20,23)]
53 bins_ocurrencias = {(0,3) : 0, (4,7): 0, (8,11): 0, (12,15): 0, (16,19): 0, (20,23):
    0}
54 # por cada gap en frecuencias_gap, le sumo su resultado al bin correspondiente
55 for gap in frecuencias_gaps:
56     for interval in bins:
57         start = interval[0]
58         finish = interval[1]
59         if start <= gap <= finish:

```



```

60     bins_ocurrencias[interval] += frecuencias_gaps[gap]
61
62 #Testeo que este for ande
63 assert bins_ocurrencias[(0,3)] == frecuencias_gaps[0] + frecuencias_gaps[1] +
    frecuencias_gaps[2] + frecuencias_gaps[3]
64 assert bins_ocurrencias[(4,7)] == frecuencias_gaps[4] + frecuencias_gaps[5] +
    frecuencias_gaps[6] + frecuencias_gaps[7]
65
66 #Ahora calculo las frecuencias relativas de cada bin de gaps
67
68 total = sum(bins_ocurrencias.values())
69 #calculo las frecuencias relativas
70 bins_frecuencias_relativas = {k: v/total for k, v in bins_ocurrencias.items()}
71
72 #calculo las frecuencias relativas acumuladas
73 bins_frecuencias_relativas_acumuladas = {}
74 bins_frecuencias_relativas_acumuladas[(0,3)] = bins_frecuencias_relativas[(0,3)]
75 bins_frecuencias_relativas_acumuladas[(4,7)] = bins_frecuencias_relativas[(4,7)] +
    bins_frecuencias_relativas_acumuladas[(0,3)]
76 bins_frecuencias_relativas_acumuladas[(8,11)] = bins_frecuencias_relativas[(8,11)] +
    bins_frecuencias_relativas_acumuladas[(4,7)]
77 bins_frecuencias_relativas_acumuladas[(12,15)] = bins_frecuencias_relativas[(12,15)]
    + bins_frecuencias_relativas_acumuladas[(8,11)]
78 bins_frecuencias_relativas_acumuladas[(16,19)] = bins_frecuencias_relativas[(16,19)]
    + bins_frecuencias_relativas_acumuladas[(12,15)]
79 bins_frecuencias_relativas_acumuladas[(20,23)] = bins_frecuencias_relativas[(20,23)]
    + bins_frecuencias_relativas_acumuladas[(16,19)]
80
81 #testeo que esto este acumulando bien
82 assert bins_frecuencias_relativas_acumuladas[(20,23)] == 1
83
84 #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)
85 FX_bins = {}
86 for interval in bins:
87     finish = interval[1]
88     FX_bins[interval] = 1 - ((0.9)**(finish + 1))
89
90 #Ahora resto los valores de bins_frecuencias_relativas_acumuladas a FX_bins
91 res = {}
92 for k in FX_bins.keys():
93     res[k] = bins_frecuencias_relativas_acumuladas[k] - FX_bins[k]
94 #Falta seguir los ultimos pasos que hace este chabon: https://www.youtube.com/watch?v=xh-4i0v-0yk
95
96 #Step 4
97 max_value = max(res.values())
98
99 #Step 5 asumo alpha 0.05
100
101 n = 100000
102
103 D = 1.36/math.sqrt(n)
104 if max_value > D:
105     print('La muestra es rechazada por el GAP test')
106 else:
107     print('La hipotesis no es rechazada')

```

A.10. Resolución ejercicio 10

```

1  #Distribucion generada en el ejercicio 3
2
3  #/usr/bin/env/ python
4  # -*- coding: utf-8 -*-
5  import numpy as np
6  from numpy import random, sqrt, log, sin, cos, pi
7  import matplotlib.pyplot as plt
8  import scipy.stats
9  import statsmodels.api as sm # recommended import according to the docs
10
11 #Distribución uniforme
12 u1 = random.uniform(0,1, 100000)
13 u2 = random.uniform(0,1, 100000)
14
15 #Box muller transformation
16 z1 = sqrt(-2*log(u1))*cos(2*pi*u2)
17 z2 = sqrt(-2*log(u1))*sin(2*pi*u2)
18
19 #Ejercicio 10:
20
21 #Aplico el test a las dos distribuciones generadas
22
23
24 test1 = scipy.stats.kstest(z1, 'norm')
25 test2 = scipy.stats.kstest(z2, 'norm')
26
27 #Hipotesis nula (de igualdad): Si no la rechazamos podemos decir que es igual a una
   normal
28 #Hipotesis alternativa (de diferencias): Si rechazamos la hipotesis nula decimos que
   hay diferencias con la distribucion normal
29 #Con un nivel de significancia del 0,05%
30
31 if test1.pvalue >= 0.05:
32     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')
33 else:
34     print('La variable z1 no pasa el test de kolmogorov-smirnov y afirmamos que
       tiene diferencias con la distribucion normal')
35 if test2.pvalue >= 0.05:
36     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')
37 else:
38     print('La variable z2 nopasa el test de kolmogorov-smirnov y afirmamos que
       tiene diferencias con la distribucion normal')
39
40
41 #Grafico de la empirica de z1
42 sample = z1
43 ecdf = sm.distributions.ECDF(sample)
44 x = np.linspace(min(sample), max(sample))
45 y = ecdf(x)
46 plt.step(x, y, label = 'empirical cdf (z1)')
47
48 #Grafico de la empirica de z2
49 sample = z2
50 ecdf = sm.distributions.ECDF(sample)
51 x = np.linspace(min(sample), max(sample))
52 y = ecdf(x)
53 plt.step(x, y, label = 'empirical cdf (z2)')

```

```
54
55 #Grafico la acumulada de la normal
56 x = np.linspace(-5, 5, 5000)
57 mu = 0
58 sigma = 1
59 y_cdf = scipy.stats.norm.cdf(x, mu, sigma) # the normal cdf
60 plt.plot(x, y_cdf, label='normal cdf')
61
62
63 plt.legend()
64 plt.show()
```