

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	5
3.3. Ejercicio 3	5
3.4. Ejercicio 4	7
3.5. Ejercicio 5	11
3.6. Ejercicio 6	12
3.7. Ejercicio 7	13
3.8. Ejercicio 8	15
3.9. Ejercicio 9	15
3.10. Ejercicio 10	15
4. Conclusiones y aclaraciones	17
Referencias	17
A. Código fuente	18
A.1. Resolución ejercicio 1	18
A.2. Resolución ejercicio 2	19
A.3. Resolución ejercicio 3	20
A.4. Resolución ejercicio 4	21
A.5. Resolución ejercicio 5	23
A.6. Resolución ejercicio 6	25
A.7. Resolución ejercicio 7	26
A.8. Resolución ejercicio 8	27
A.9. Resolución ejercicio 9	28
A.10. Resolución ejercicio 10	30

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.

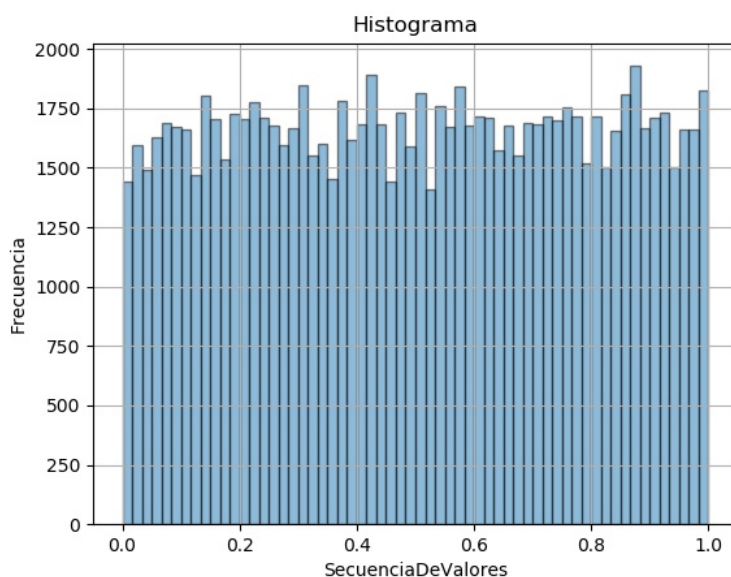
3.1. Ejercicio 1

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:

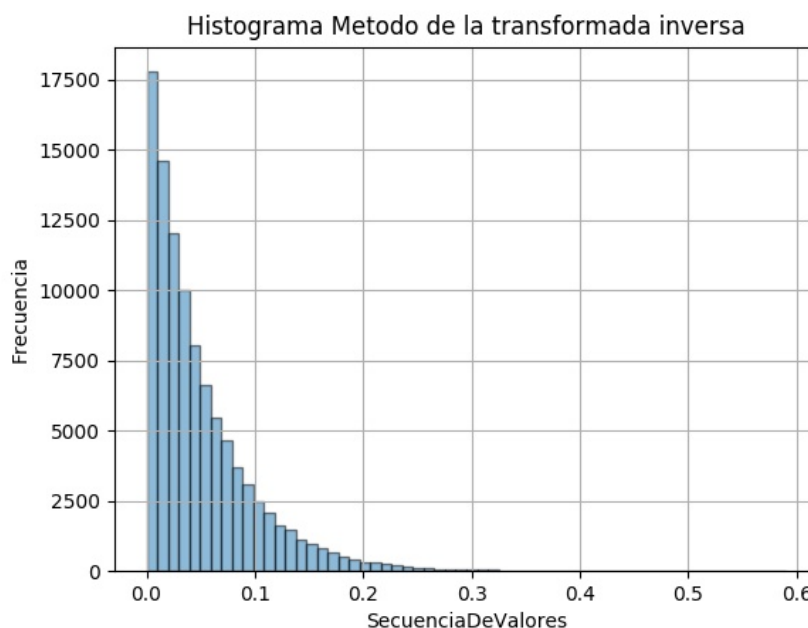


3.2. Ejercicio 2

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 bastante 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.

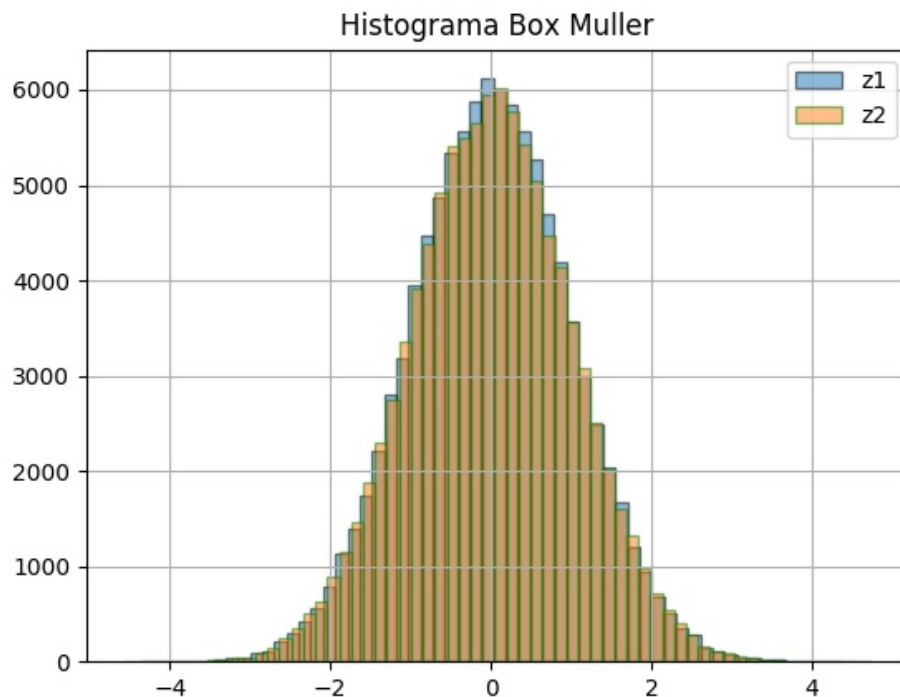
3. Realizamos histogramas con $z1$ y $z2$ y otro histograma con una distribución normal estándar.

4. Calculamos la media y varianza de la distribución simulada y teórica.

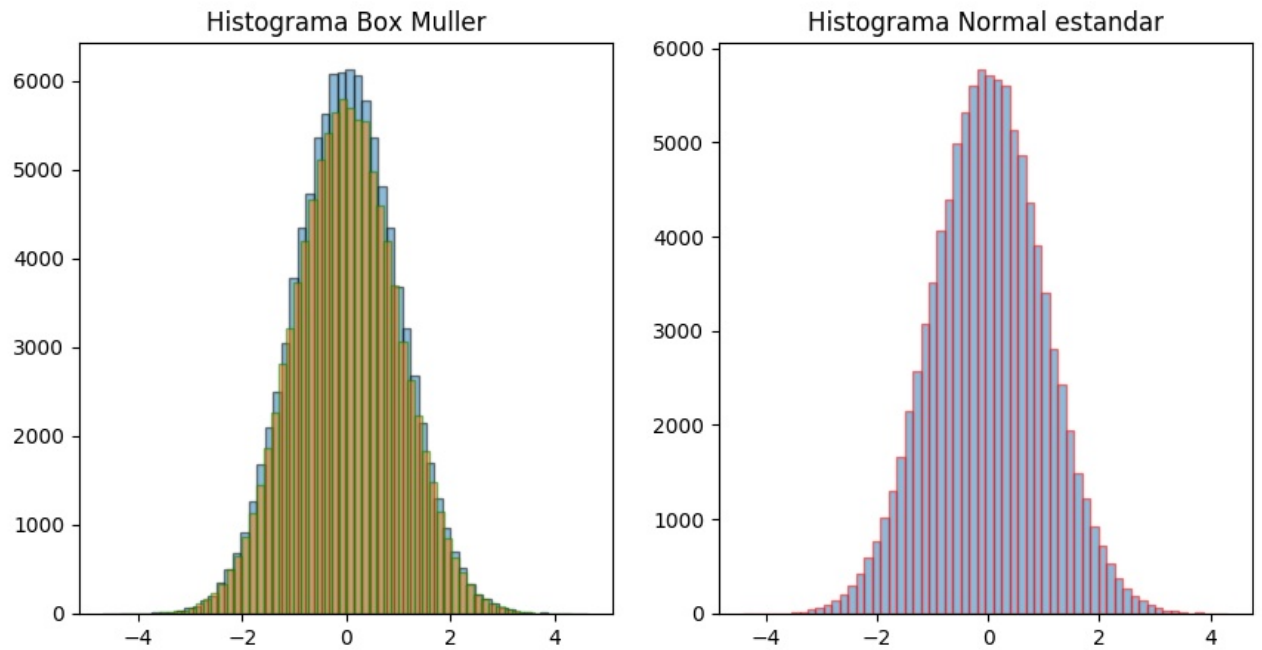
Los resultados que obtenemos son los siguientes:

- 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:

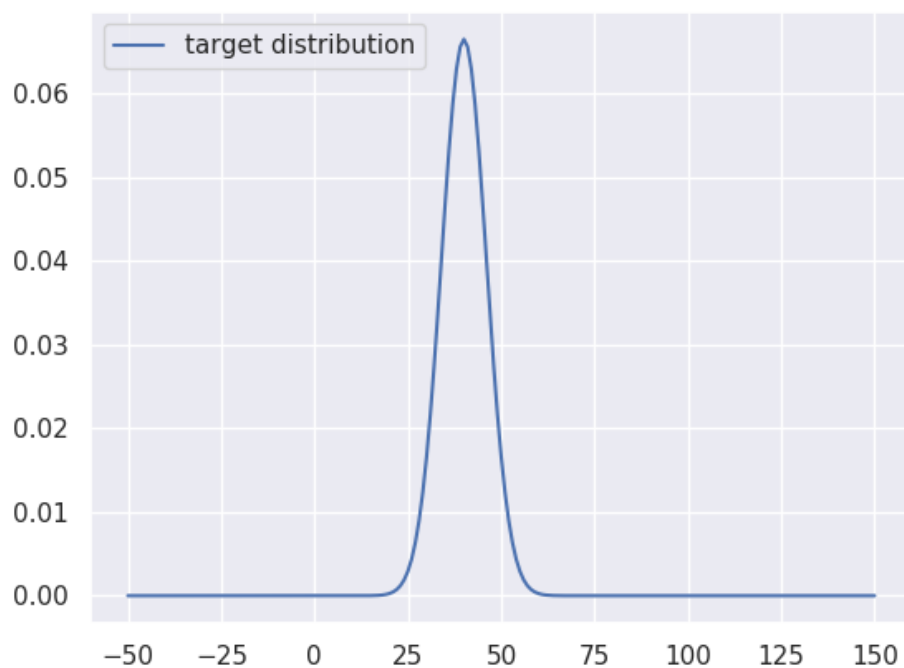


Por lo tanto podemos observar que se comprueba que el método de Box Muller es una distribución normal estándar.

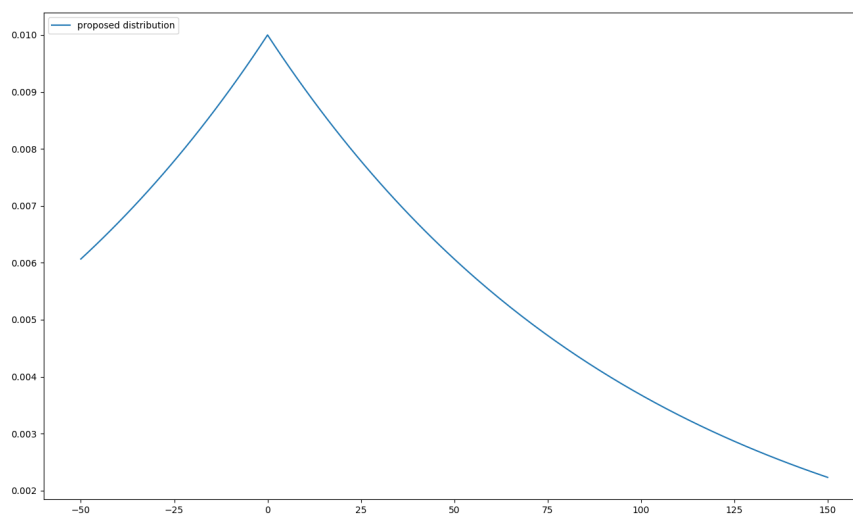
3.4. Ejercicio 4

Seguimos los pasos de esta guía: Rejection Sampling.

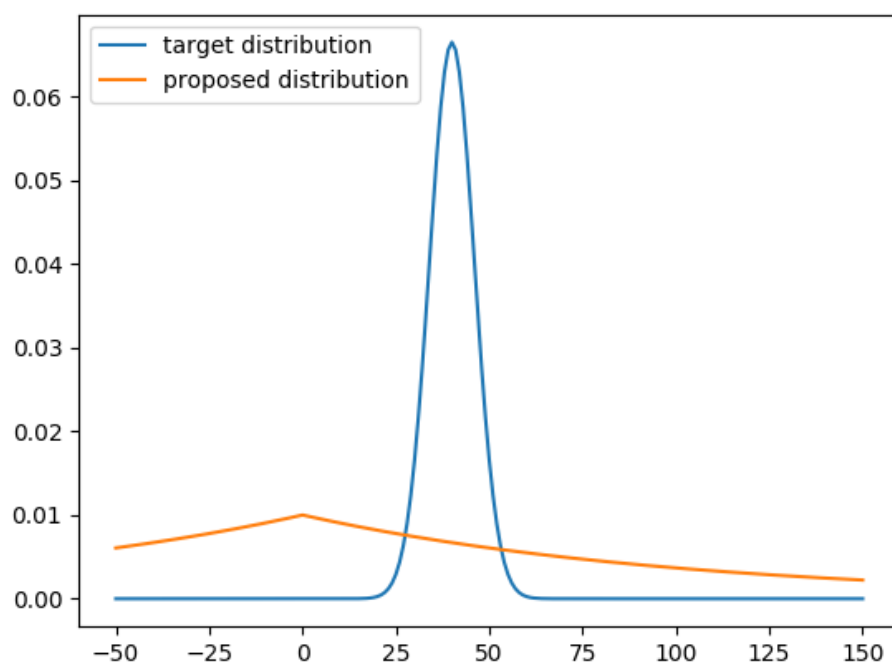
La distribución target $P(x)$ es una $N(40, 6)$



Proponemos $Exp(0,01)$ como $Q(x)$



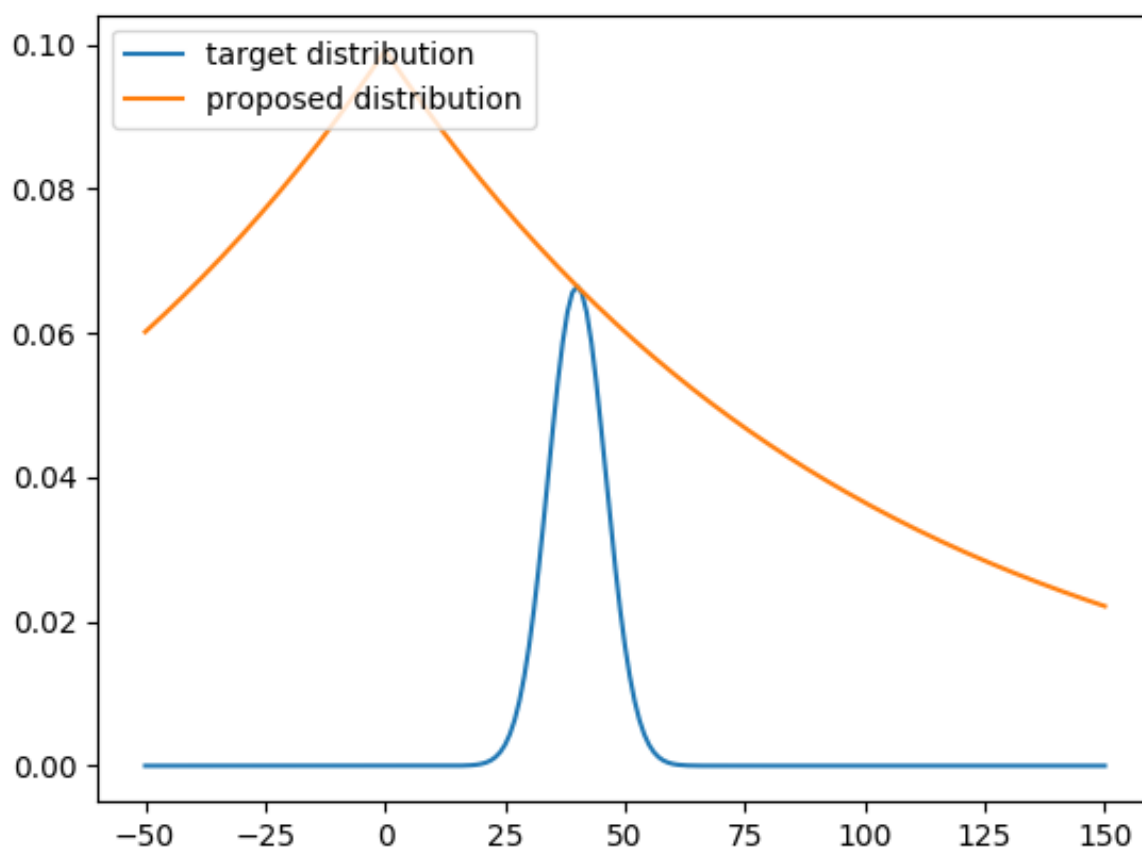
Rejection Sampling va a fallar debido a que la $Q(x)$ propuesta no esta envolviendo a nuestra distribución target $P(x)$



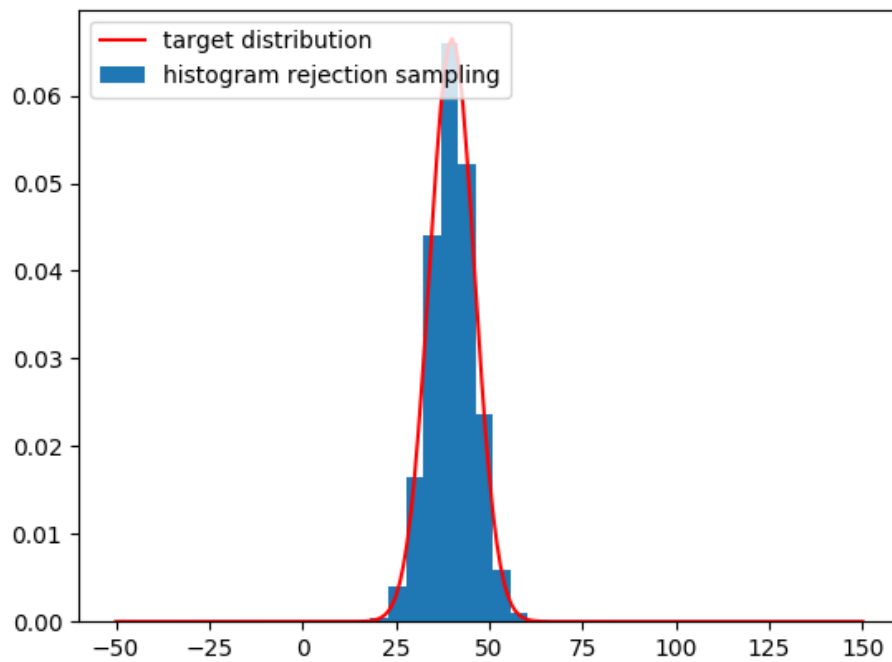
Para remediar esto, encontramos un factor k que multiplique a $Q(x)$ que es la relación máxima entre $P(x)$ y $Q(x)$, entonces:

$$k = \max(P(x)/Q(x)) \text{ for all } x$$

Al reescalar..



La distribución 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 se encuentra alrededor de 40, y la varianza cerca de 36.

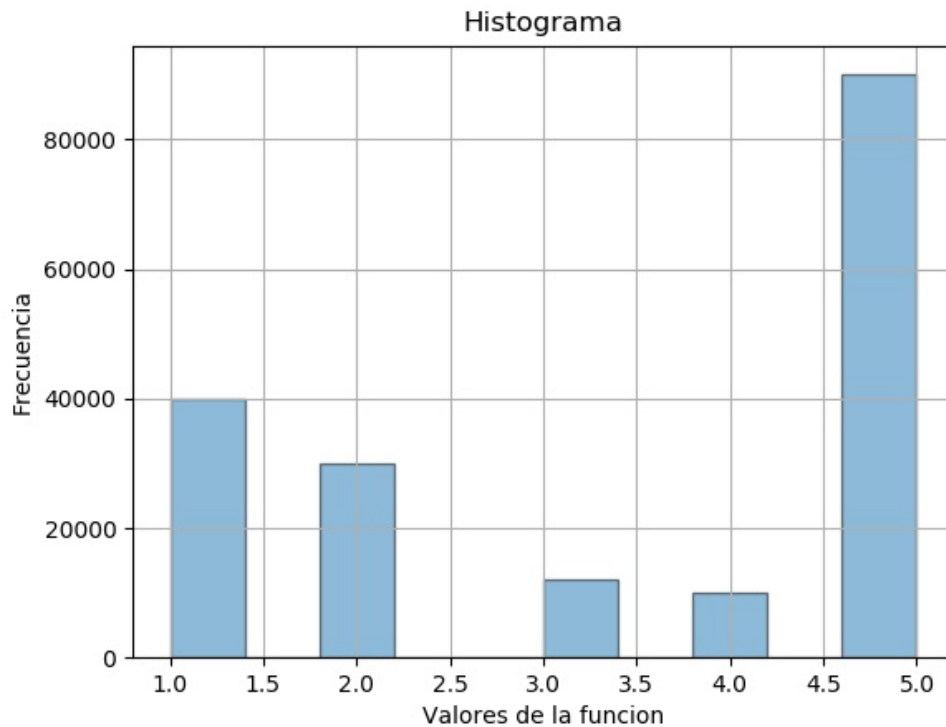
La varianza muestral es mucho mas amplia comparada con la teorica. La media muestral es la misma.

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:

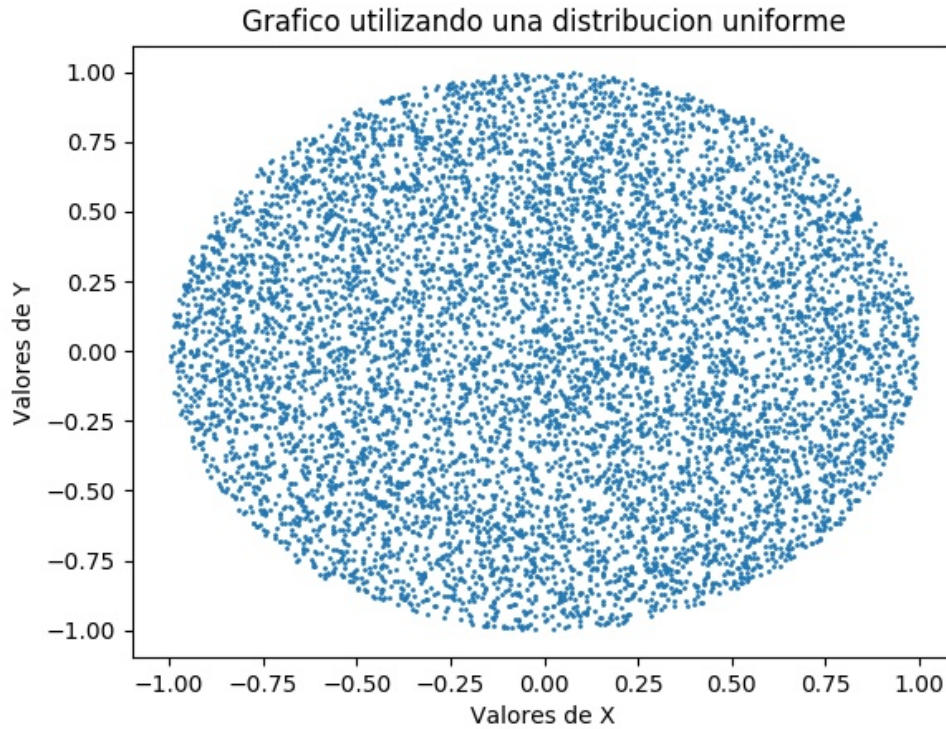


3.6. Ejercicio 6

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 utilizando 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.

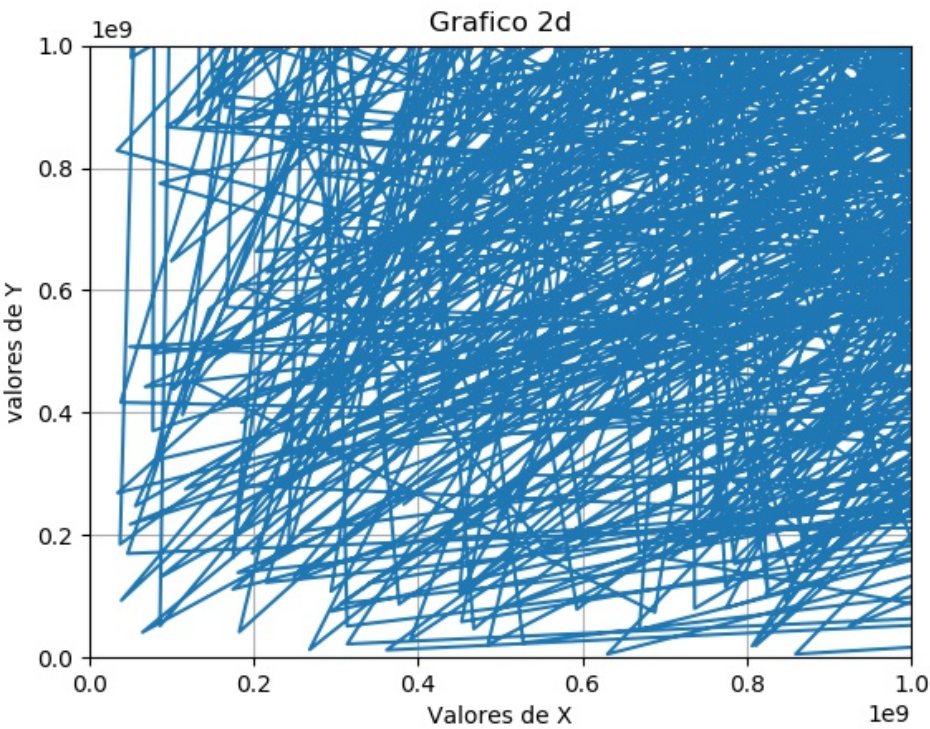


3.7. Ejercicio 7

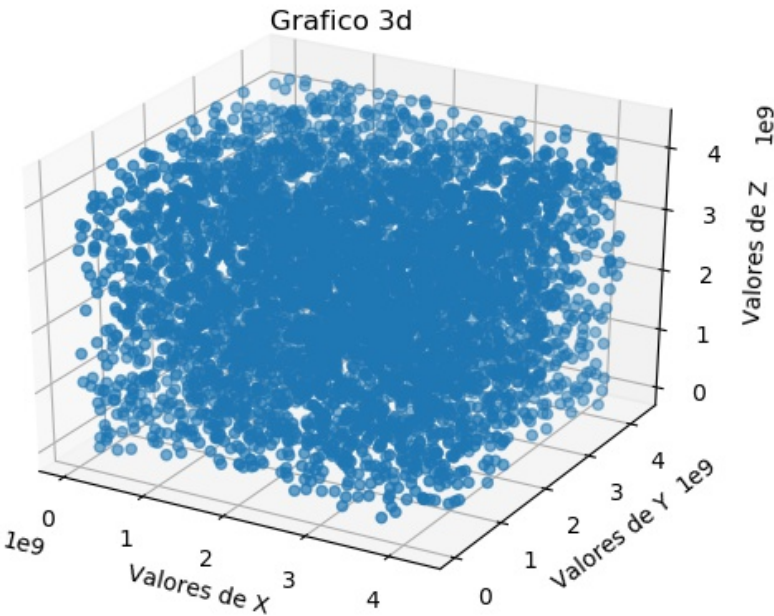
Para realizar el ejercicio aplicamos lo siguiente:

1. Se genera 3 listas x, y, z utilizando el generador GCL.
2. Con las listas x, y, z creamos 3 listas nuevas filtrando con aquellas que cumplen:
 - $GCL(i), GCL(i + 3), GCL(i + 6), \dots$
 - $GCL(i + 1), GCL(i + 4), GCL(i + 7), \dots$
 - $GCL(i + 2), GCL(i + 5), GCL(i + 8), \dots$
3. Realizamos un gráfico utilizando estas dos listas nuevas.

El gráfico en 2D es



El grafico en 3D es:



3.8. Ejercicio 8

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

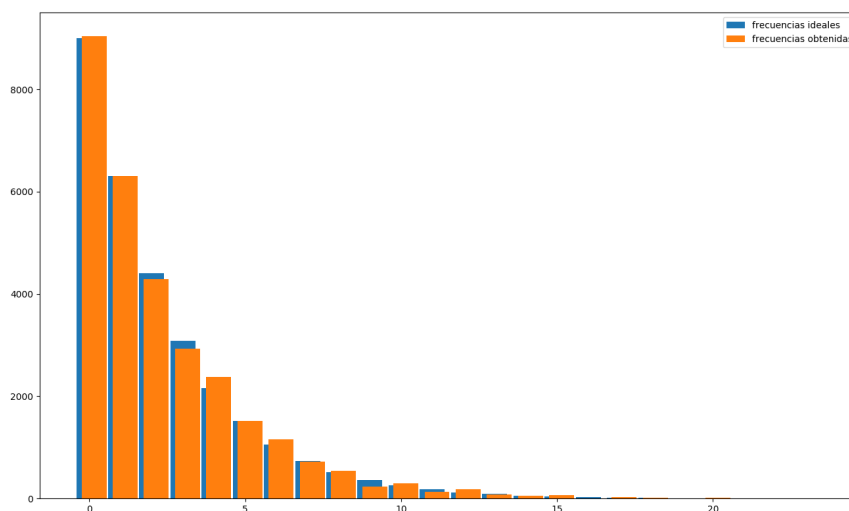
1. Utilizamos la distribución empírica generada en el ejercicio 5). Se obtuvo los datos de la gráfica del histograma creado, éstas serían las frecuencias observadas y son también los valores observados.
2. Los valores esperados lo obtenemos del enunciado.
3. Utilizando la biblioteca de Python *chisquare* aplicamos el test comparando las frecuencias reales contra las esperadas. Los resultados que se encontraron fueron:
 - El valor de probabilidad es 0.000471795221251
 - El valor del estadístico es 20.1250583333

En nuestro caso obtuvimos que se acepta la hipótesis con la distribución empírica con un error del 1

3.9. Ejercicio 9

Se siguieron los pasos del algoritmo explicado en The Art of Computing Program Volumen 2 de D.E.Knuth, página 62

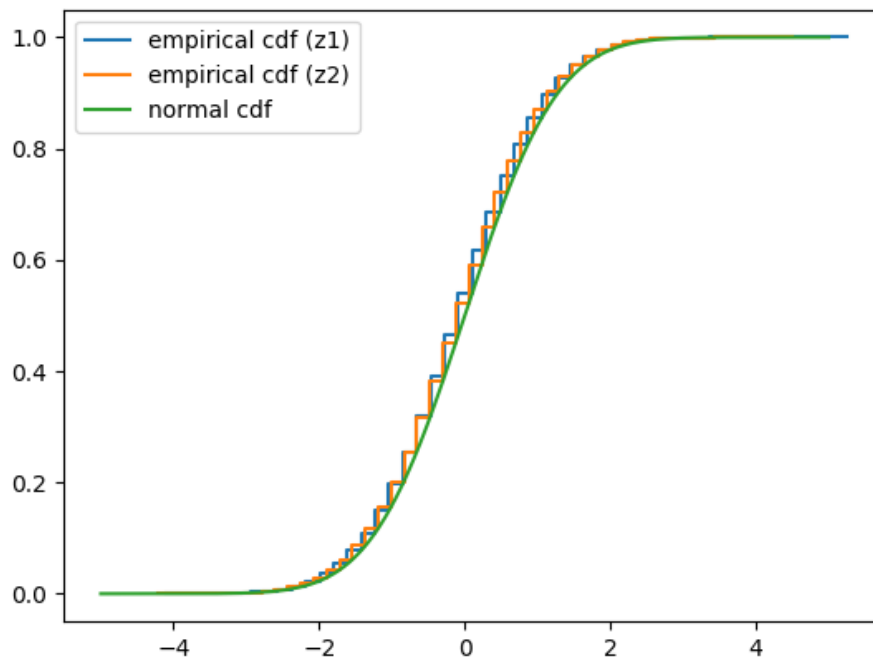
Realizamos un gráfico para comparar las frecuencias de gaps obtenidas y las esperadas:



Graficamente se espera que pase el test, y así sucedió con un nivel de significación 0.05

3.10. Ejercicio 10

1. Nos apoyamos en *scipy.stats*, específicamente 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 empíricas, estas dos variables generadas mediante el método 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 podíamos haber trabajado con una única distribución generada por el método, nos pareció 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
- [4] Graficar en python, https://matplotlib.org/api/_as_gen/mpl_toolkits.mplot3d.axes3d.Axes3D.html

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 * 0.6)
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(20)) * math.log(1-u) ##modificado
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(20)))
49
50 #Calculo de varianza
51 varianza = np.var(secuencia)
52 print("El valor simulado de la varianza es {}".format(varianza))
53 lamdaCuadrado = float(1)/float(20**2)
54 print("El valor teórico de la varianza es {}".format(float(1)/lamdaCuadrado))

```

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  import math
5
6  def p(x):
7      #target
8      return st.norm.pdf(x, loc=40, scale=6)
9
10
11 def q(x):
12     #proposed
13     y = np.array([])
14     for number in x:
15         n = abs(number)
16         lambd = 0.01
17         res = (math.e**(-n * lambd)) * lambd
18         y = np.append(y, res)
19     return y
20
21 def q2(n):
22     #proposed
23     #misma funcion que q(x) pero como la tuve que implementar a mano, la funcion
    anterior
24     #no funcionaba si tenia que recibir un solo numero, como se hace en
    rejection_sampling
25     lambd = 0.01
26     res = (math.e**(-n * lambd)) * lambd
27     return res
28
29 x = np.arange(-50, 151)
30 k = max(p(x) / q(x))
31
32 def rejection_sampling(iter=1000):
33     samples = []
34
35     for i in range(iter):
36         z = np.random.exponential(float(1)/0.01) #aca es 1/lambda porque recibe la
        media
37
38         u = np.random.uniform(0, k*q2(z))
39
40         if u <= p(z):
41             samples.append(z)
42
43     return np.array(samples)
44
45
46 if __name__ == '__main__':
47     plt.plot(x, p(x), label = 'target distribution')
48     plt.plot(x, k*q(x), label = 'proposed distribution')
49     plt.legend(loc='upper left')
50     plt.show()
51     print('Rejection sampling in progress..')
52     s = rejection_sampling(iter=100000)
53     mean = round(s.mean(), 2)
54     var = round(s.var(), 2)
55     print('La media de la muestra generada es: '+str(mean)+' y la varianza: '+str(
        var))
56     plt.plot(x, p(x), label = 'target distribution', color = 'red')
57     plt.hist(s, label = 'histogram rejection sampling', normed=True)

```

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

```


62

63 `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 math
5  import numpy as np
6  import matplotlib.pyplot as plt
7  import matplotlib.colors
8  from matplotlib.pyplot import plot
9  from mpl_toolkits.mplot3d import axes3d
10
11
12  modulo = 2**32
13  multiplicador = 1013904223
14  incremento = 1664525
15  semilla = int(92702 * 0.15 + 93584 * 0.25 + 98757 * 0.26)
16  secuencia = [ semilla ]
17
18  def GCL( valor ):
19      return ( multiplicador * valor + incremento ) % modulo
20
21  x=[]
22  y=[]
23  z=[]
24
25  k=0
26  valor=semilla
27  for i in range(15000):
28      valor=GCL(valor)
29      #print(valor)
30      if(k==0):
31          x.append(valor)
32          k=1
33      else:
34          if(k==1):
35              y.append(valor)
36              k=2
37          else:
38              z.append(valor)
39              k=0
40
41  #grafico 3d
42  fig = plt.figure()
43  ax = fig.add_subplot(111, projection='3d')
44  ax.scatter(x,y,z,marker='o')
45  plt.title("Grafico 3d")
46  ax.set_xlabel('Valores de X')
47  ax.set_ylabel('Valores de Y')
48  ax.set_zlabel('Valores de Z')
49  plt.show()
50
51  #grafico 2d
52  fig=plt.figure()
53  plt.xlim(1,10000000000)
54  plt.ylim(1,10000000000)
55  plot(x,y)
56  plt.title('Grafico 2d')
57  plt.xlabel('Valores de X')
58  plt.ylabel('valores de Y')
59  plt.grid(True)
60  plt.show()

```

A.8. Resolución ejercicio 8

```
1  #!/usr/bin/env/ python
2  # -*- coding: utf-8 -*-
3  from scipy import stats as sp
4  from numpy import random
5
6  #Tomamos los resultados del ejercicio 5
7  valoresObservados=[39440.,30081.,12106.,10317.,8093.]
8  #Tomamos los valores esperados por a distribución
9  valoresEsperados=[40000.0, 30000.0, 12000.0, 10000.0, 8000.0]
10
11 estadistica, pvalue = sp.chisquare(valoresObservados, valoresEsperados)
12 print(estadistica)
13 print(pvalue)
14 if (pvalue < 0.01):
15     print(' Se acepta la hipotesis con la distribucion empirica con un error
16         menor del 1% ')
17 else:
18     print('Se rechaza la hipotesis con la distribucion empirica con un error
19         mayor del 1% ')
```

A.9. Resolución ejercicio 9

```

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

```

```
61     b = 0.5
62
63     p = b - a
64     gap_prob = p * (1 - p)**gap
65     return gap_prob
66
67
68     frecuencias_ideales = []
69     frecuencias_obtenidas = []
70     gaps = []
71     for gap in frecuencias_gaps:
72         gaps.append(gap)
73         frecuencias_ideales.append(proba_ideal(gap) * total)
74         frecuencias_obtenidas.append(frecuencias_gaps[gap])
75
76     #esto es para mostrar mejor el grafico
77     gaps_corridos = []
78     for gap in gaps:
79         gaps_corridos.append(gap + 0.15)
80
81     plt.bar(gaps, frecuencias_ideales, label='frecuencias ideales')
82     plt.bar(gaps_corridos, frecuencias_obtenidas, label='frecuencias obtenidas')
83     plt.legend()
84     plt.show()
85     result = scipy.stats.chisquare(frecuencias_obtenidas, frecuencias_ideales)
86     if result.pvalue < 0.05:
87         print('Pasa el gap test')
```

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()
```