Métodos Montecarlo Fing 2022 - Entrega 5 - Números aleatorios

Autor: Carlos M. Martinez, mayo 2022.

Email: carlosm@fing.edu.uy, carlos@cagnazzo.uy

Ejercicio 8.1

Parte a:

Elegir al menos dos fuentes de números aleatorios disponibles en Internet (sitio o tabla con valores). Explicar cómo funcionan, como se accede a los números, y que características tienen.

Fuente (1) QRNG Service de la Universidad de Berlín Humboldt

Link: Sitio web de la Universidad Humboldt de Berlín, QRNG

El servicio QRNG de la Universidad Humboldt de Berín QRNGBER es un servicio de generación de números aleatorios basado en un fenómeno físico, el tiempo de llegada de fotones a un sensor. ESte fenómeno tiene características cuánticas que hacen que naturalmente sea un fenómeno probabilistico.

Según los autores los números aleatorios generados son de buena calidad y el equipo puede construirse relativamente a bajo costo.

El acceso a QRNGBER se realiza via web. El servicio requiere un registro previo pero no tiene costo. Le pueden bajar archivos de 1MB, 15MB y 100MB con bytes generados aleatoriamente. Los autores aseguran que ningun archivo bajado se repite, y de hecho tienen un contador en la página que muestra cuantos petabytes de "random data" llevan entregados.

También los autores hacen disponible algunas utilidades y librerías que permiten acceder al servicio programáticamente sin bajar manualmente archivos. Lamentablemente no se encuentra una documentación del API de esta librería que permita por ejemplo escribir un cliente Python nativo.

Fuente (2) RAND Book of 1 Million Random Numbers

En 1947 la RAND Corporation, como respuesta a la creciente demanda de realización de simulaciones publicó un libro conteniendo una tabla de un millón de dígitos generados aleatoriamente. Para generar la primera edición de esta tabla construyeron un dispositivo que utilizaba una rueda de ruleta preparada de tal forma que el resultado de cada jugada podía ser leído por una computadora electrónica Wiki1. Durante muchos años esta tabla fue referencia obligada de expertos de diferentes areas de la ciencia.

La lectura de los dígitos de la ruleta tenía sesgos significativos que fueron corregidos previamente a la producción de la tabla publicada.

En 2001 RAND publicó una nueva edición de este libro, la cual está disponible gratuitamente a través de la web de RAND en A Million Random Digits with 100,000 Normal Deviates.

La utilización de la tabla implica los siguientes pasos.

- Seleccionar una posición de arranque.
  - La tabla contiene 20.000 líneas de 50 dígitos cada una, agrupados en "palabras" de 5 dígitos. La versión en papel del libro tiene 400 páginas.
  - En la época en la que se utilizaba el libro impreso en papel lo que se hacía era abrir el libro en una página al azar, de esa página elegir un grupo de 5 digitos al azar.

In [ ]:
```
import random
import math
import pandas
from IPython.core.display import HTML
random.seed()

import cm2c.fing.mmc.volumen as mmc
import cm2c.fing.mmc.utils as mmcutils
reloj_ppal = mmcutils.timeit()
mmc.version()
```

Out[ ]:
```
'Volúmenes en R^N MMC v0.1.1 - Carlos Martinez marzo 2022'
```

Parte b:

En base a este análisis, elegir una de las fuentes, fundamentar la selección, y modificar el ejercicio 3.1, parte a (visto en la sesión 3) para que emplee dichos números aleatorios (en lugar de los generados por bibliotecas como hasta el momento). Comparar si la salida obtenida es consistente o no con la obtenida en los experimentos de la parte a del ejercicio 3.1.

Resolución

Selecciono la fuente de números aleatorios de QRNG. Voy a utilizar un archivo bajado de 15MB generado el 21 de mayo de 2022 a las 12.30 UTC-3.

La forma de obtener números al azar de este archivo será de la siguiente forma:

- abrir el archivo desde python en formato binario
- cargar el archivo de 15MB en memoria (para lograr mejor velocidad) en un array
- seleccionar un indice de partida utilizando el generador pseudo aleatorio del sistema
- leer el array considerandolo una estructura circular generando numeros aleatorios en el intervalo [0,255]

In [ ]:
```
class QRNG_UBerlin:
    qrng_data_file = "rnd/sampledata-15MB.bin"
```

```
    def __init__(self):
        with open(self.qrng_data_file, "rb") as file:
            self.qrng_data = file.read()

        self._ix = random.randint(0, len(self.qrng_data)-1 )

    def _sampleone(self):
        """
        Devuelve una muestra considerando el archivo como una estructura circular.
        """
        r = self.qrng_data[self._ix]
        self._ix = self._ix + 1
        if self._ix == len(self.qrng_data):
            self._ix = 0
        return r
    # end def

    def sample(self, n):
        """
        Devuelve n muestras del archivo fuente aleatorio.
        """
        r = []
        for x in range(n):
            r.append(self._sampleone())
        return r
    # end def

    def uniform(self, a, b):
        """
        Al igual que random.uniform devuelve un float entre a y b.
        """
        k = 1

        p = self._sampleone()

        # self._ix = (self._ix + p ) % len(self.qrng_data)

        s = self.sample(k)

        sc = 1/255

        # rbase = s[3]*sc + s[2]*(sc**2) + s[1]*(sc**3) + s[0]*(sc**4)
        rbase = 0
        for i in range(k):
            rbase = rbase + s[i]*(sc**(k-i))

        r = a + rbase*(b-a)

        return r
    # end def
# end class

qrng = QRNG_UBerlin()
print(qrng.sample(10))

print(qrng.uniform(0,1))
```

```
[36, 229, 96, 148, 250, 123, 134, 0, 138, 185]
0.9215686274509803
```

In [ ]:
```
import random
import math
import tabulate
import time
random.seed()

def puntoDentroVolumen(punto, restricciones=True):
    """
    Devuelve 0 o 1 si un punto esta fuera o dentro de un cierto volumen.
    Si restricciones es "false", el volumen es la hiperesfera en R6
    """

    # Para que este dentro del volumen tiene que estar dentro de la esfera
    # y ademas cumplir con las restricciones adicionales

    dentro = 1
    fuera = 0

    # chequeo 1 : dentro de esfera

    d = math.sqrt(
        (punto[0]-0.45)**2 +
        (punto[1]-0.5)**2 +
        (punto[2]-0.6)**2 +
        (punto[3]-0.6)**2 +
        (punto[4]-0.5)**2 +
        (punto[5]-0.45)**2
    )

    # si la distancia es mayor al radio, esta fuera
    if (d>=0.35) :
        return fuera

    if restricciones:
        # restriccion 1
        if 3*punto[0] + 7*punto[3] > 5:
            return fuera
        # restriccion 2
        if punto[2]+punto[3] > 1:
            return fuera
        # restriccion 3
        if punto[0]-punto[1]-punto[4]+punto[5] < 0:
            return fuera
    else:
        return dentro

    return dentro
# end fun punto dentro del volumen

mmc.sortearPuntoRN(6, qrng.uniform)
```

Out[ ]:
```
[0.2823529411764706,
 0.2823529411764706,
 0.5607843137254902,
 0.792156862745098,
 0.06274509803921569,
 0.6274509803921569]
```

In [ ]:
```
VolHEAnalitico = math.pi**3*(0.35**6)/6
```

```
# Caclulo del volumen de la hiperesfera por MMC
(VolR, VarVolR, S, execTime) = mmc.MetodoMonteCarlo(10**5, lambda x: puntoDentroVolume

print("Volumen hiper esfera por MMC = {:e}, Varianza = {:e}".format(VolR, VarVolR))

print(" ")
print("Volumen hiper esfera analitico = {:e}, diferencia MMC - analitico = {:.3f}%".fo
```

```
Volumen hiper esfera por MMC = 9.590000e-03, Varianza = 9.498127e-08

Volumen hiper esfera analitico = 9.499629e-03, diferencia MMC - analitico = -0.942%
```

Comparación entre calculo con pseudo-random y QRNG del volumen de la hiperesfera

En esta seccion corremos MMC para calcular el volumen de la hiperesfera en R^6 para diferentes tamanos de muestra, usando tanto numeros generados con el pseudo-random como con los numeros bajados del servicio QRNG.

El segundo parámetro booleano en "False" indica a la función "puntoDentroVolumen" que no debe aplicar las restricciones adicionales.

In [ ]:
```
table = [ ['N', 'Vol HE (analitico)',
           'S (Pseudo)', 'Vol HE (Pseudo)', 'Var. (Pseudo)', 'Dif% (Pseudo)','T (s) (
           'S (qrng)', 'Vol HE (qrng)', 'Var. (qrng)', 'Dif% (qrng)', 'T (s) (qrng)'
        ], ]

for n in [4, 5, 5, 6, 6, 7]:
    (VolR, VarVolR, S, execTime) = mmc.MetodoMonteCarloParalelo(10**n, 8, lambda x: pu
    qrng2 = QRNG_UBerlin()
    (VolRq, VarVolRq, Sq, execTimeq) = mmc.MetodoMonteCarloParalelo(10**n, 8, lambda x
    table.append(
        [
            10**n, "{:3e}".format(VolHEAnalitico),
            S, "{:3e}".format(VolR), "{:3e}".format(VarVolR), "{:.3f}%".format((VolHEA
            Sq, "{:3e}".format(VolRq), "{:3e}".format(VarVolRq), "{:.3f}%".format((Vol
        ]
     )


tabulate.tabulate(table, tablefmt='html')
```

Out[ ]:

N	Vol HE (analitico)	S (Pseudo)	Vol HE (Pseudo)	Var. (Pseudo)	Dif% (Pseudo)	T (s) (Pseudo)	S (qrng)	Vol HE (qrng)	Var. (qr
10000	9.499629e-03	96	9.600000e-03	9.508791e-07	-1.046%	0.059338	88	8.800000e-03	8.72343
100000	9.499629e-03	1048	1.048000e-02	1.037027e-07	-9.355%	0.049514	816	8.160000e-03	8.09349
100000	9.499629e-03	992	9.920000e-03	9.821692e-08	-4.238%	0.048696	992	9.920000e-03	9.82169
1000000	9.499629e-03	9200	9.200000e-03	9.115369e-09	3.257%	0.392603	9304	9.304000e-03	9.21744
1000000	9.499629e-03	9656	9.656000e-03	9.562771e-09	-1.619%	0.386925	9456	9.456000e-03	9.36659
10000000	9.499629e-03	94248	9.424800e-03	9.335974e-10	0.794%	4.355111	93896	9.389600e-03	9.30143

Comparación entre calculo con pseudo-random y QRNG del volumen con restricciones.

En esta seccion corremos MMC para calcular el volumen de la hiperesfera restringida en R^6 para diferentes tamanos de muestra, usando tanto numeros generados con el pseudo-random como con los numeros bajados del servicio QRNG.

El segundo parámetro booleano en "True" indica a la función "puntoDentroVolumen" que no debe aplicar las restricciones adicionales.

In [ ]:

```
table = [ ['N',
           'S (Pseudo)', 'Vol HE (Pseudo)', 'Var. (Pseudo)','Spread (pseudo)', 'T (s)
           'S (qrng)', 'Vol HE (qrng)', 'Var. (qrng)', 'Spread (qrng)','T (s) (qrng)'
        ], ]

for n in [5, 5, 6, 6, 7]:
    (VolR, VarVolR, S, execTime) = mmc.MetodoMonteCarloParalelo(10**n, 8, lambda x: pu
    (ipl, ipu) = mmc.intConfianzaAC(S, 10**n, 0.05)
    ic_rel = (ipu-ipl)*100/VolR
    #
    qrng = QRNG_UBerlin()
    (VolRq, VarVolRq, Sq, execTimeq) = mmc.MetodoMonteCarloParalelo(10**n, 8, lambda x
    (ipql, ipqu) = mmc.intConfianzaAC(Sq, 10**n, 0.05)
    ic_relq = (ipqu-ipql)*100/VolRq
    #
    table.append(
        [
            10**n,
            S, "{:3e}".format(VolR), "{:3e}".format(VarVolR), "{:.3f}%".format(ic_rel)
            Sq, "{:3e}".format(VolRq), "{:3e}".format(VarVolRq), "{:.3f}%".format(ic_r
        ]
     )
## end for


tabulate.tabulate(table, tablefmt='html')
```

```
/tmp/ipykernel_18537/853280894.py:14: RuntimeWarning: divide by zero encountered in d
ouble_scalars
  ic_relq = (ipqu-ipql)*100/VolRq
```

Out[ ]:

N	S (Pseudo)	Vol HE (Pseudo)	Var. (Pseudo)	Spread (pseudo)	T (s) (Pseudo)	S (qrng)	Vol HE (qrng)	Var. (qrng)	
100000	24	2.400000e-04	2.399448e-09	83.141%	0.054522	0	0.000000e+00	0.000000e+00	
100000	40	4.000000e-04	3.998440e-09	63.434%	0.049569	16	1.600000e-04	1.599760e-09	1
1000000	288	2.880000e-04	2.879173e-10	23.172%	0.406516	264	2.640000e-04	2.639306e-10	
1000000	320	3.200000e-04	3.198979e-10	21.975%	0.407966	320	3.200000e-04	3.198979e-10	
10000000	3040	3.040000e-04	3.039076e-11	7.111%	3.911319	2872	2.872000e-04	2.871175e-11	

Conlusiones

Los resultados obtenidos el generador de números aleatorios de Python y el obtenido del servicio QRNG son consistentes. Sin embargo para la mayor parte de las corridas el resultado obtenido con el generador pseudo-random de Python es de igual calidad y es bastante más rapido.

Seguramente la forma en la que utilizo el stream de bytes aleatorios para generar un float entre (0 y 1) no es la mejor.

Datos adicionales y referencias

Información acerca del software y hardware utilizados

Software:

- Python 3.8.10 corriendo en Windows WSL2 (Windows Subsystem for Linux)
- Jupyter Notebook

Librerias:

- scipy norm
- pathos multiprocessing (para paralelizar ejecuciones)

Hardware:

- PC Windows 11, con WSL2
- CPU Intel Core i5 10400F (6 cores, 12 threads de hardware)
- 16 GB de RAM

In [ ]:
```
print(f"%% FIN - tiempo total de ejecución {reloj_ppal.lap():.3f}s")
```

```
%% FIN - tiempo total de ejecución 49.923s
```

Código de las funciones desarrolladas

Adjunto en el archivo "*aleatorios.py.pdf*".