Prueba Práctica - Números Pseudoaleatorios

Por: S. Peñaranda J.

Realizar un método que permita generar números pseudoaleatorios pro medio de los cuadrados medios

Se buscar encontrar la eficiencia de la generación de números pseudo-aletorios a través de los métodos de cuadrados medios y congruencia lineal, para ello se debe seguir el siguiente proceso:

1. A traves de la misma api generar una semilla diferente.
2. Encontrar el numero de iteraciones hasta que se repita uno de sus datos.
3. Generar 100 simulaciones con diferentes semillas.
4. Generar un histograma con el resultado obtenidos por cada método.
5. Agregar sus conclusiones, opiniones y recomendaciones

Método de cuadrados medios

In [1]:
```
from tabulate import tabulate
from prettytable import PrettyTable

##Creamos las variables y pedimos los datos necesarios
iteraciones= int(input("Interaciones: "))
x0= int(input("Semilla: "))
digitos=int(input("Digitos: "))
xn=0
dobleXn=0
longitud=8
ui=0
rn=0

numeroDividir = ''
numeroDividir = numeroDividir.ljust(digitos,'0')
numeroDividir='1'+numeroDividir
print(numeroDividir)

##Creamos la tabla e iniciamos el proceso
tabla = PrettyTable()
tabla.field_names =['Iteracion','Xn','Xn*Xn','Ui','Rn']
x_0=x0
for i in range(0,iteraciones):
    xn=x_0
    dobleXn=pow(xn,2)
    longitud=(len(str(dobleXn)))
    medio1=((longitud/2)-(digitos/2))
    medio2=((longitud/2)+(digitos/2))
    ui=(int(str(dobleXn)[int(medio1):int(medio2)]))

    if len(str(ui))<digitos:
        medio1=((longitud/2)-(digitos/2))
        medio2=((longitud/2)+(digitos/2)+1)
        ui=(int(str(dobleXn)[int(medio1):int(medio2)]))

    rn=(ui/int(numeroDividir))
    x_0=ui
    tabla.add_row([i,xn,dobleXn,ui,rn])
print(tabla)
```

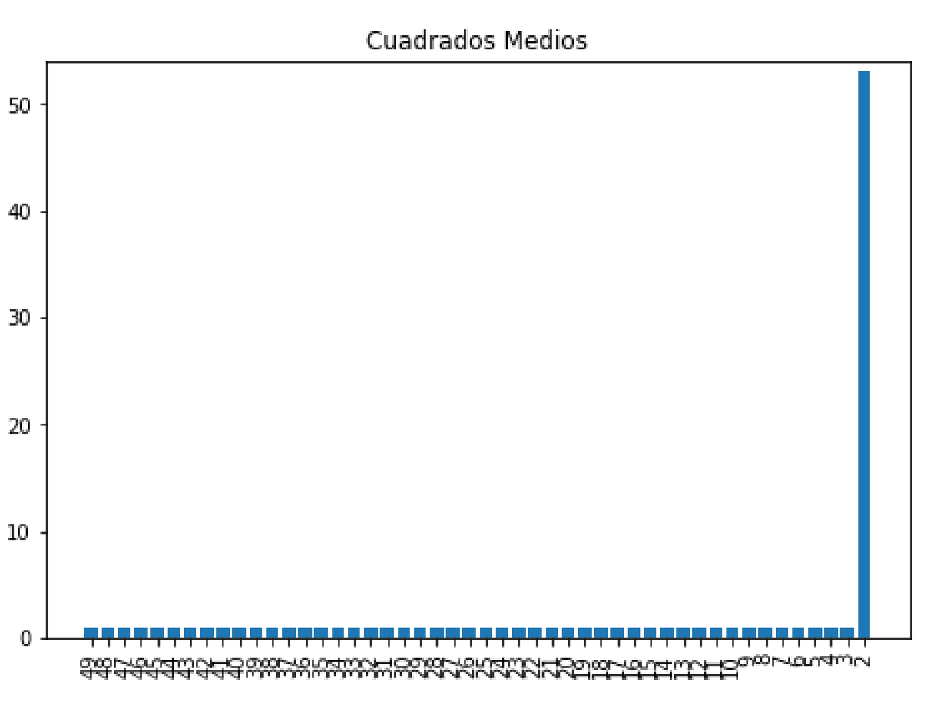
```
Interaciones: 1
Semilla: 7543
Digitos: 4
10000
+-----------+------+----------+------+--------+
| Iteracion |  Xn  |  Xn*Xn   |  Ui  |   Rn   |
+-----------+------+----------+------+--------+
|     0     | 7543 | 56896849 | 8968 | 0.8968 |
+-----------+------+----------+------+--------+
```

In [8]:
```
repeticion = []
for k in range(0,100):
    iteraciones = 1
    uid = []
    bandera = True
    while bandera:
        xn=x_0
        dobleXn=pow(xn,2)
        longitud=(len(str(dobleXn)))
        medio1=((longitud/2)-(digitos/2))
        medio2=((longitud/2)+(digitos/2))
        ui=(int(str(dobleXn)[int(medio1):int(medio2)]))
        if len(str(ui))<digitos:
            medio1=((longitud/2)-(digitos/2))
            medio2=((longitud/2)+(digitos/2)+1)
            ui=(int(str(dobleXn)[int(medio1):int(medio2)]))
        rn=(ui/int(numeroDividir))
        x_0=ui
        if( iteraciones == 1 ):
            uid.append(ui)
            iteraciones = iteraciones + 1
        else:
            for x in range(0,len(uid)):
                if(uid[x]==ui):
                    repeticion.append(iteraciones)
                    x_0 = int(uid[0])
                    bandera = False
            uid.append(ui)
            iteraciones = iteraciones + 1
print(repeticion)
```

```
[49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21,
20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
```

In [9]:
```
import pylab as pl
import numpy as np
def contarElementosLista(lista):

    return {i:lista.count(i) for i in lista}

lista = repeticion

resultado=contarElementosLista(lista)
print(resultado)

X = np.arange(len(resultado))
fig = pl.figure()
ax = fig.add_axes([0,0,1,1])
pl.bar(X, resultado.values())
pl.xticks(X, resultado.keys(), rotation = 90)
pl.title("Cuadrados Medios")
ymax = max(resultado.values()) + 1
pl.ylim(0, ymax)
pl.show()
```

```
{49: 1, 48: 1, 47: 1, 46: 1, 45: 1, 44: 1, 43: 1, 42: 1, 41: 1, 40: 1, 39: 1, 38: 1, 37: 1, 36: 1, 35: 1, 34: 1, 33:
1, 32: 1, 31: 1, 30: 1, 29: 1, 28: 1, 27: 1, 26: 1, 25: 1, 24: 1, 23: 1, 22: 1, 21: 1, 20: 1, 19: 1, 18: 1, 17: 1, 1
6: 1, 15: 1, 14: 1, 13: 1, 12: 1, 11: 1, 10: 1, 9: 1, 8: 1, 7: 1, 6: 1, 5: 1, 4: 1, 3: 1, 2: 53}
```



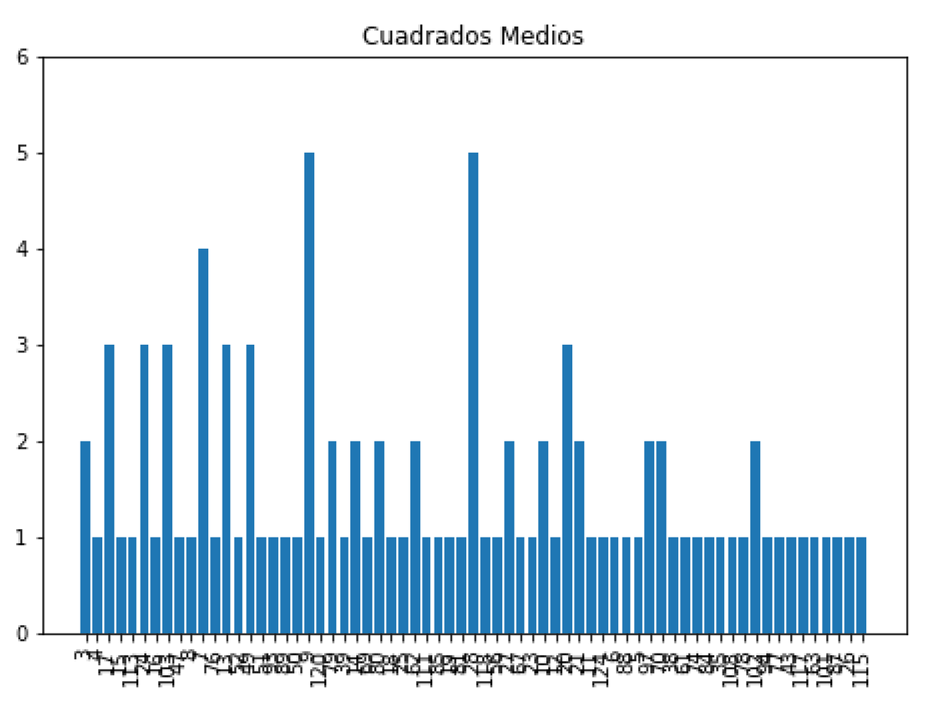
Alterando la nueva semilla de otra forma

In [10]:
```
repeticionL = []
for k in range(0,100):
    iteraciones = 1
    uid = []
    bandera = True
    while bandera:
        xn=x_0
        dobleXn=pow(xn,2)
        longitud=(len(str(dobleXn)))
        medio1=((longitud/2)-(digitos/2))
        medio2=((longitud/2)+(digitos/2))
        ui=(int(str(dobleXn)[int(medio1):int(medio2)]))
        if len(str(ui))<digitos:
            medio1=((longitud/2)-(digitos/2))
            medio2=((longitud/2)+(digitos/2)+1)
            ui=(int(str(dobleXn)[int(medio1):int(medio2)]))
        rn=(ui/int(numeroDividir))
        x_0=ui
        if( iteraciones == 1 ):
            uid.append(ui)
            iteraciones = iteraciones + 1
        else:
            for x in range(0,len(uid)):
                if(uid[x]==ui):
                    repeticionL.append(iteraciones)
                    x_0 = int(uid[0])+10-3
                    bandera = False
        uid.append(ui)
        iteraciones = iteraciones + 1
print(repeticionL)
```

```
[3, 3, 4, 17, 15, 113, 24, 16, 103, 47, 8, 7, 76, 13, 52, 49, 51, 93, 89, 50, 9, 120, 79, 7, 13, 39, 14, 69, 80, 79,
18, 17, 7, 25, 9, 62, 111, 85, 19, 81, 9, 28, 28, 103, 118, 56, 27, 67, 23, 10, 12, 20, 21, 11, 14, 28, 124, 6, 88,
5, 9, 97, 24, 24, 70, 62, 49, 80, 38, 103, 49, 61, 74, 84, 21, 17, 35, 108, 78, 102, 94, 77, 27, 43, 97, 102, 20, 7,
28, 13, 117, 70, 20, 63, 9, 10, 101, 87, 28, 26, 115]
```

In [11]:
```
import pylab as pl
import numpy as np
def contarElementosLista(lista):

    return {i:lista.count(i) for i in lista}

lista = repeticionL
print(resultado)

resultado=contarElementosLista(lista)

X = np.arange(len(resultado))
fig = pl.figure()
ax = fig.add_axes([0,0,1,1])
pl.bar(X, resultado.values(), align='center')
pl.xticks(X, resultado.keys(), rotation = 90)
pl.title("Cuadrados Medios")
ymax = max(resultado.values()) + 1
pl.ylim(0, ymax)
pl.show()
```

```
{49: 1, 48: 1, 47: 1, 46: 1, 45: 1, 44: 1, 43: 1, 42: 1, 41: 1, 40: 1, 39: 1, 38: 1, 37: 1, 36: 1, 35: 1, 34: 1, 33:
1, 32: 1, 31: 1, 30: 1, 29: 1, 28: 1, 27: 1, 26: 1, 25: 1, 24: 1, 23: 1, 22: 1, 21: 1, 20: 1, 19: 1, 18: 1, 17: 1, 1
6: 1, 15: 1, 14: 1, 13: 1, 12: 1, 11: 1, 10: 1, 9: 1, 8: 1, 7: 1, 6: 1, 5: 1, 4: 1, 3: 1, 2: 53}
```



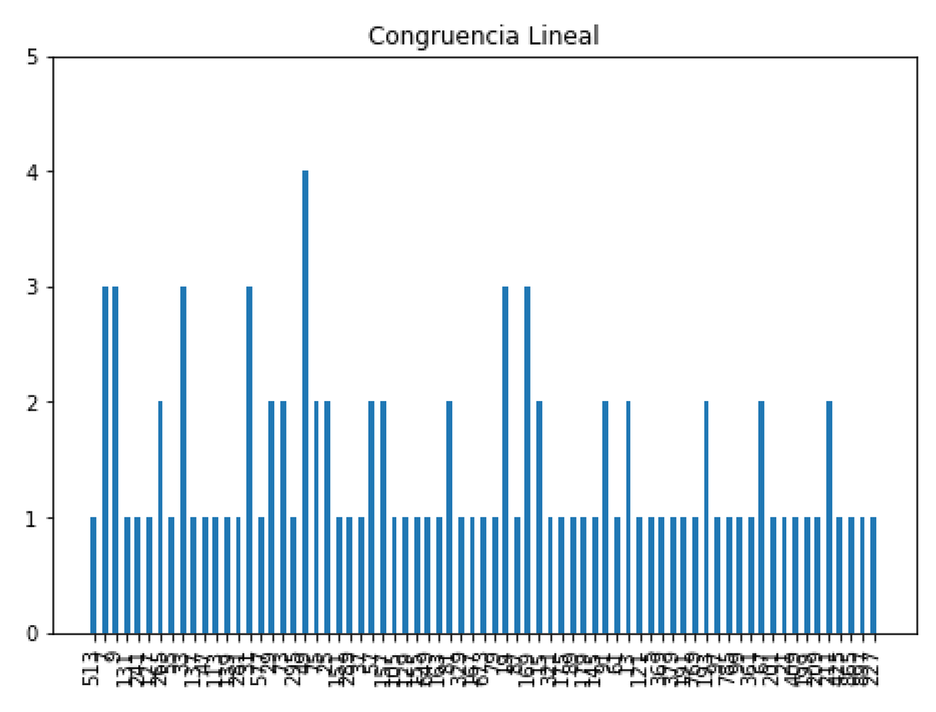
Método de congruencia lineal

In [12]:
```
#321, 5, 1, 512
a=5
c=1
x0=321
m=512
repeticionCL=[]
for k in range(0,100):
    iteraciones=1
    uidCL=[]
    xn=0
    ui=0.0
    axiliar=0
    bandera= True
    xn=x0
    while bandera:
        axiliar=((a*xn)+c)%(m)
        ui=axiliar/m
        xn=axiliar
        if (iteraciones==1):
            uidCL.append(ui)
            iteraciones=iteraciones+1
        else:
            for x in range(0,len(uidCL)):
                if (uidCL[x]==ui):
                    repeticionCL.append(iteraciones)
                    x0=int(xn)
                    a=a+2
                    c=c+1
                    m=m+4
                    bandera=False
            uidCL.append(ui)
            iteraciones=iteraciones+1
print(repeticionCL)
```

```
[513, 7, 9, 131, 241, 127, 265, 55, 33, 137, 265, 47, 113, 139, 281, 31, 577, 29, 73, 295, 49, 75, 25, 151, 289, 37,
57, 31, 49, 157, 105, 157, 129, 155, 649, 163, 81, 31, 329, 167, 673, 79, 9, 19, 49, 87, 169, 15, 321, 175, 89, 179,
145, 91, 169, 61, 33, 13, 121, 5, 369, 379, 73, 191, 769, 193, 97, 7, 785, 99, 361, 67, 33, 67, 201, 29, 49, 21, 409,
199, 193, 91, 169, 15, 209, 211, 425, 7, 865, 211, 9, 19, 81, 13, 25, 75, 897, 19, 57, 227]
```

In [13]:
```
def contarElementosLista(listaCL):

    return {i:listaCL.count(i) for i in listaCL}

listaCL = repeticionCL
resultadoCL=contarElementosLista(listaCL)
print(resultadoCL)

import pylab as pl
import numpy as np

X = np.arange(len(resultadoCL))
fig = pl.figure()
ax = fig.add_axes([0,0,1,1])
pl.bar(X, resultadoCL.values(), align='center', width=0.5)
pl.xticks(X, resultadoCL.keys(), rotation = 90)
pl.title("Congruencia Lineal")
ymax = max(resultadoCL.values()) + 1
pl.ylim(0, ymax)
pl.show()
```

```
{513: 1, 7: 3, 9: 3, 131: 1, 241: 1, 127: 1, 265: 2, 55: 1, 33: 3, 137: 1, 47: 1, 113: 1, 139: 1, 281: 1, 31: 3, 577:
1, 29: 2, 73: 2, 295: 1, 49: 4, 75: 2, 25: 2, 151: 1, 289: 1, 37: 1, 57: 2, 157: 2, 105: 1, 129: 1, 155: 1, 649: 1, 1
63: 1, 81: 2, 329: 1, 167: 1, 673: 1, 79: 1, 19: 3, 87: 1, 169: 3, 15: 2, 321: 1, 175: 1, 89: 1, 179: 1, 145: 1, 91:
2, 61: 1, 13: 2, 121: 1, 5: 1, 369: 1, 379: 1, 191: 1, 769: 1, 193: 2, 97: 1, 785: 1, 99: 1, 361: 1, 67: 2, 201: 1, 2
1: 1, 409: 1, 199: 1, 209: 1, 211: 2, 425: 1, 865: 1, 897: 1, 227: 1}
```



Conclusiones

- Como se observa en cada uno de los histogramas podemos ver que en el primer método la semilla tendrá mucho que ver con los resultados que se esperan, puesto que en el primer ejemplo se trabaja con una secuencia de semilla, pero en el segundo ejemplo se altera un poco la semilla por tanto los resultados como muestra la gráfica son mucho mejores que los primeros.
- Para con el segundo método es un poco mas sencillo el proceso por parte de la semilla, pero se debe tener en cuenta que existen otras variables que intervienen en el proceso, mismas que con una nueva semilla deberían cambiar para un mejor resultado.

Opiniones

- Cada método tiene su forma de encontrar números aleatorios, creo que se puede usar cada método según se necesita, ya que a mi parece el segundo método al tener mas variables que interactuan es mejor para la generación de estos números, sin embargo el primer método será óptimo siempre y cuando la semilla sea **la mejor**.

Recomendaciones

- Seleccionar una semilla adecuada para iniciar el proceso, para de esta manera evitar que los números que el método genere se repitan ciclicamente luego de mínimas iteraciones.
- En caso de usar un método para la generación de una semilla, probarlo con un número considerable de iteraciones para de esta manera saber que tanto se repiten los números a la hora de ser generados y hacer uso de estos.
- Para elegir una semilla se puede usar la hora del ordenador, o tambien la temperatura del CPU entre otros elementos.