Universidad Politecnica Salesiana

Nombre: Marcela Zhagui

In [21]:
```
import matplotlib.pyplot as plt
import math
import numpy as np
import pandas as pd, math
from tabulate import tabulate
from prettytable import PrettyTable
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
```

In [22]:
```
semilla= int(input("Semilla:    "))
iteraciones= int(input("Interaciones:"))
digitos=int(input("Digitos:     "))
sem_1=0
dobleXn=0
pseudo=0
rn=0
```

```
Semilla:    754625412
Interaciones:100
Digitos:     5
```

In [23]:
```
numeroDividir = ''
itera=[]
lista=[]
listar_num=[]
lista.append(['Iteracion','Xn','Xn*Xn','Ui','Rn'])
numeroDividir = numeroDividir.ljust(digitos,'0')
numeroDividir='1'+numeroDividir
print(numeroDividir)

x_0=semilla
for i in range(0,iteraciones):
    itera.append(i+1)
    sem_1=x_0
    dobleXn=pow(sem_1,2)
    longitud=(len(str(dobleXn)))
    medio1=((longitud/2)-(digitos/2))
    medio2=((longitud/2)+(digitos/2))
    pseudo=(int(str(dobleXn)[int(medio1):int(medio2)]))

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

    rn=(pseudo/int(numeroDividir))
    x_0=pseudo
    listar_num.append(rn)

    lista.append([i,sem_1,dobleXn,pseudo,rn])


print(tabulate(lista, stralign='center', tablefmt='fancy_grid',floatfmt='.0f'
))
```

100000

Iteracion	Xn	Xn*Xn	Ui	Rn
0	754625412	569459512436169744	51243	0.51243
1	51243	2625845049	25845	0.25845
2	25845	667964025	79640	0.7964
3	79640	6342529600	42529	0.42529
4	42529	1808715841	87158	0.87158
5	87158	7596516964	96516	0.96516
6	96516	9315338256	15338	0.15338
7	15338	235254244	52542	0.52542
8	52542	2760661764	60661	0.60661
9	60661	3679756921	79756	0.79756
10	79756	6361019536	61019	0.61019
11	61019	3723318361	23318	0.23318
12	23318	543729124	37291	0.37291
13	37291	1390618681	90618	0.90618
14	90618	8211621924	11621	0.11621
15	11621	135047641	50476	0.50476
16	50476	2547826576	47826	0.47826
17	47826	2287326276	87326	0.87326
18	87326	7625830276	25830	0.2583
19	25830	667188900	71889	0.71889
20	71889	5168028321	68028	0.68028
21	68028	4627808784	27808	0.27808
22	27808	773284864	32848	0.32848
23	32848	1078991104	78991	0.78991
24	78991	6239578081	39578	0.39578
25	39578	1566418084	66418	0.66418
26	66418	4411350724	11350	0.1135

27	11350	128822500	88225	0.88225
28	88225	7783650625	83650	0.8365
29	83650	6997322500	97322	0.97322
30	97322	9471571684	71571	0.71571
31	71571	5122408041	22408	0.22408
32	22408	502118464	21184	0.21184
33	21184	448761856	87618	0.87618
34	87618	7676913924	76913	0.76913
35	76913	5915609569	15609	0.15609
36	15609	243640881	36408	0.36408
37	36408	1325542464	25542	0.25542
38	25542	652393764	23937	0.23937
39	23937	572979969	29799	0.29799
40	29799	887980401	79804	0.79804
41	79804	6368678416	68678	0.68678
42	68678	4716667684	16667	0.16667
43	16667	277788889	77888	0.77888
44	77888	6066540544	66540	0.6654
45	66540	4427571600	27571	0.27571
46	27571	760160041	16004	0.16004
47	16004	256128016	61280	0.6128
48	61280	3755238400	55238	0.55238
49	55238	3051236644	51236	0.51236
50	51236	2625127696	25127	0.25127
51	25127	631366129	13661	0.13661
52	13661	186622921	66229	0.66229
53	66229	4386280441	86280	0.8628
54	86280	7444238400	44238	0.44238

55	44238	1957000644	57000	0.57
56	57000	3249000000	49000	0.49
57	49000	2401000000	10000	0.1
58	10000	100000000	0	0.0
59	0	0	0	0.0
60	0	0	0	0.0
61	0	0	0	0.0
62	0	0	0	0.0
63	0	0	0	0.0
64	0	0	0	0.0
65	0	0	0	0.0
66	0	0	0	0.0
67	0	0	0	0.0
68	0	0	0	0.0
69	0	0	0	0.0
70	0	0	0	0.0
71	0	0	0	0.0
72	0	0	0	0.0
73	0	0	0	0.0
74	0	0	0	0.0
75	0	0	0	0.0
76	0	0	0	0.0
77	0	0	0	0.0
78	0	0	0	0.0
79	0	0	0	0.0
80	0	0	0	0.0
81	0	0	0	0.0
82	0	0	0	0.0
83	0	0	0	0.0

84	0	0	0	0.0
85	0	0	0	0.0
86	0	0	0	0.0
87	0	0	0	0.0
88	0	0	0	0.0
89	0	0	0	0.0
90	0	0	0	0.0
91	0	0	0	0.0
92	0	0	0	0.0
93	0	0	0	0.0
94	0	0	0	0.0
95	0	0	0	0.0
96	0	0	0	0.0
97	0	0	0	0.0
98	0	0	0	0.0
99	0	0	0	0.0

In [24]:
```
N=len(listar_num)
n=math.sqrt(N)
c=0
aux=0.0
listaT=[]
aux_EI=[.1,.2,.3,.4,.5,.6,.7,.8,.9,1]
Oi=[]
Ei=[]
f=[]
listaOI=[]
s=[]
while(c<10):
    ceo=0
    f.append(aux)
    for i in listar_num:
        if(i>aux and i<aux_EI[c]):
            ceo+=1
    Oi.append(ceo)
    Ei.append(N/n)
    aux=aux_EI[c]
    c+=1
for ix in range(len(Oi)):
    res=(math.pow((Ei[ix]-Oi[ix]),2)/Ei[ix])
    s.append(res)
rest=0
datos=pd.DataFrame({"INTERVALO ":Ei,"  ":f," ":aux_EI,"Oi":Oi,"(Oi- Ei)²/Ei":s
})
datos
```

Out[24]:

	INTERVALO			Oi	(Oi- Ei)²/Ei
0	10.0	0.0	0.1	0	10.0
1	10.0	0.1	0.2	7	0.9
2	10.0	0.2	0.3	11	0.1
3	10.0	0.3	0.4	4	3.6
4	10.0	0.4	0.5	4	3.6
5	10.0	0.5	0.6	6	1.6
6	10.0	0.6	0.7	8	0.4
7	10.0	0.7	0.8	8	0.4
8	10.0	0.8	0.9	6	1.6
9	10.0	0.9	1.0	3	4.9

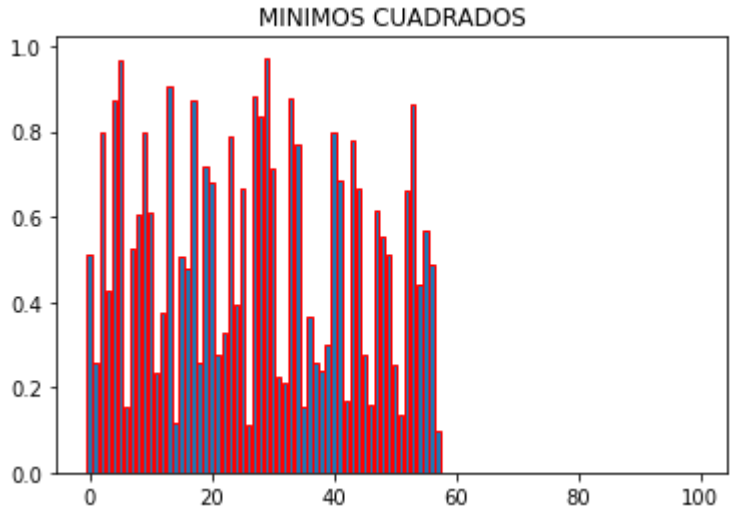
In [25]:
```
for i in s:
    rest+=i;

rest
if (rest>16.9):
    print("la hipotesis es nula NO es válida")
else :
    print ("la hipotesis es valida")
```

```
la hipotesis es nula NO es válida
```

Grafica de los 100 primeros numeros pseudoaleatorios con Minimos Cuadrados

In [26]:
```
X = np.arange(len(listar_num))
y=listar_num
plt.bar(X,y,align="center",edgecolor='red')
plt.title("MINIMOS CUADRADOS")
plt.show()
```



Congruencia Lineal

In [32]:
```
a=int(input("Ingrese el valor de a: "))
c=int(input("Ingrese el valor de c: "))
m=int(input("Ingrese el valor de m: "))
semilla= int(input("Ingrese el valor de la semilla: "))
iteracion= int(input("Ingrese el numero de iteraciones: "))
sem_1=0
un=0
aux_xn=0
aux_un=0
```

```
Ingrese el valor de a: 410
Ingrese el valor de c: 100
Ingrese el valor de m: 200
Ingrese el valor de la semilla: 1200
Ingrese el numero de iteraciones: 100
```

In [33]:
```
listaCo=[]
listCoR=[]
listaCo.append(['Iteracion','Xn','Un'])
sem_1=semilla
for i in range(0,iteracion):
    aux_xn = sem_1
    aux_un = un
    sem_1=(a*sem_1+c)%m
    un=sem_1/m
    listCoR.append(aux_un)
    listaCo.append([i,aux_xn,aux_un])
print(tabulate(listaCo,tablefmt='fancy_grid',stralign='center',floatfmt='.0f'
))
```

Iteracion	Xn	Un
0	1200	0
1	100	0.5
2	100	0.5
3	100	0.5
4	100	0.5
5	100	0.5
6	100	0.5
7	100	0.5
8	100	0.5
9	100	0.5
10	100	0.5
11	100	0.5
12	100	0.5
13	100	0.5
14	100	0.5
15	100	0.5
16	100	0.5
17	100	0.5
18	100	0.5
19	100	0.5
20	100	0.5
21	100	0.5
22	100	0.5
23	100	0.5
24	100	0.5
25	100	0.5
26	100	0.5

27	100	0.5
28	100	0.5
29	100	0.5
30	100	0.5
31	100	0.5
32	100	0.5
33	100	0.5
34	100	0.5
35	100	0.5
36	100	0.5
37	100	0.5
38	100	0.5
39	100	0.5
40	100	0.5
41	100	0.5
42	100	0.5
43	100	0.5
44	100	0.5
45	100	0.5
46	100	0.5
47	100	0.5
48	100	0.5
49	100	0.5
50	100	0.5
51	100	0.5
52	100	0.5
53	100	0.5
54	100	0.5
55	100	0.5

56	100	0.5
57	100	0.5
58	100	0.5
59	100	0.5
60	100	0.5
61	100	0.5
62	100	0.5
63	100	0.5
64	100	0.5
65	100	0.5
66	100	0.5
67	100	0.5
68	100	0.5
69	100	0.5
70	100	0.5
71	100	0.5
72	100	0.5
73	100	0.5
74	100	0.5
75	100	0.5
76	100	0.5
77	100	0.5
78	100	0.5
79	100	0.5
80	100	0.5
81	100	0.5
82	100	0.5
83	100	0.5

84	100	0.5
85	100	0.5
86	100	0.5
87	100	0.5
88	100	0.5
89	100	0.5
90	100	0.5
91	100	0.5
92	100	0.5
93	100	0.5
94	100	0.5
95	100	0.5
96	100	0.5
97	100	0.5
98	100	0.5
99	100	0.5

TEST-CHI

In [34]:
```
N=len(listCoR)
n=math.sqrt(N)
c=0
aux=0.0
listaT=[]
aux_EI=[.1,.2,.3,.4,.5,.6,.7,.8,.9,1]
Oi=[]
Ei=[]
f=[]
listaOI=[]
test=[]
while(c<10):

    ceo=0
    f.append(aux)
    for i in listCoR:
        if(i>aux and i<aux_EI[c]):
            ceo+=1
    Oi.append(ceo)
    Ei.append(N/n)
    aux=aux_EI[c]
    c+=1
for ix in range(len(Oi)):
    res=(math.pow((Ei[ix]-Oi[ix]),2)/Ei[ix])
    test.append(res)
r11=0
datos=pd.DataFrame({" Intervalo ":Ei,"  ":f," ":aux_EI,"Oi":Oi,"(Oi- Ei)²/Ei":
test})
datos


#Calacular los lista_repetidos
#lista_aux=[]
#lista_repetido = []
#unico = []
#lista_result=[]
#print(iteraciones_congruencia)
#def frecuencia(lista,valor):
#    lista_aux=0
 #   for i in valor:
  #      for j in lista:
   #         if i == j:
    #            lista_aux=lista_aux+1
     #   lista_result.append(lista_aux)
      #  lista_aux=0
    #return lista_result


#for x in iteraciones_congruencia:
 #   del lista_aux[:]
  #  if x not in unico:
   #     unico.append(x)
    #else:
     #   if x not in lista_repetido:
      #      lista_repetido.append(x)
```

Out[34]:

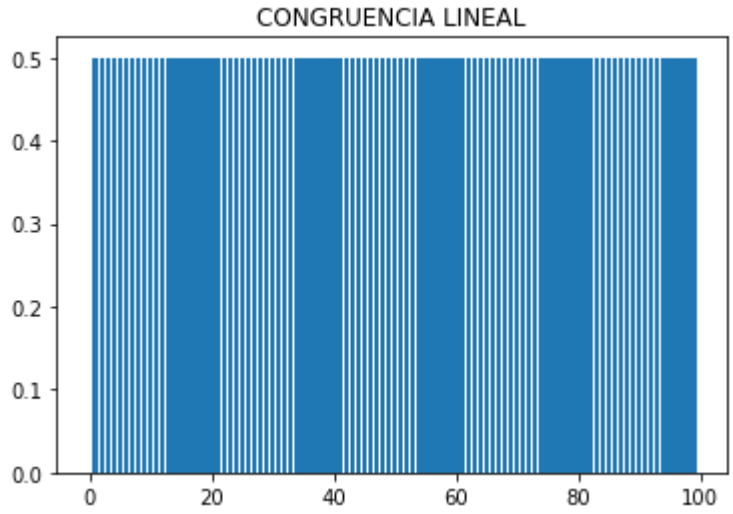
	Intervalo			Oi	(Oi– Ei)²/Ei
0	10.0	0.0	0.1	0	10.0
1	10.0	0.1	0.2	0	10.0
2	10.0	0.2	0.3	0	10.0
3	10.0	0.3	0.4	0	10.0
4	10.0	0.4	0.5	0	10.0
5	10.0	0.5	0.6	0	10.0
6	10.0	0.6	0.7	0	10.0
7	10.0	0.7	0.8	0	10.0
8	10.0	0.8	0.9	0	10.0
9	10.0	0.9	1.0	0	10.0

In [35]:
```
for i in test:
    r11+=i;

if (r11>16.9):
    print("la hipotesis es nula; NO es válida")
else :
    print ("la hipotesis es valida")

X = np.arange(len(listCoR))
y=listCoR
plt.bar(X,y,align="center")
plt.title("CONGRUENCIA LINEAL")
plt.show()
```

```
la hipotesis es nula; NO es válida
```



Conclusiones

- En conclusion se puede apreciar que el metodo de congruencia lineal ofrece una mayor aleatoriedad con respecto al metodo de los cuadrados medios ya que la frecuencia de encontrar un dato repetido es menos probable en dicho metodo. Es por esto que las librerias de numeros randomicos utilizan este metodo para la generacion de dichos numeros.
- Aunque el metodo de cuadrados medios no ofrezca la misma aleatoriedad que el de congruencia lineal, se puede dar una semilla mas grande para que tratar de solucionar este error.

Opiniones

- Estos metodos son muy utiles para entender como trabajan internamente las diferentes librerias de los lenguajes de programacion y asi poder replicarlos en caso de que no se cuente con la documentacion respectiva en un momento determinado.

Recomendaciones

- La semilla debe ser una optima para que los numeros sean lo mas aleatorios posible. Ademas es importante que el metodo considere los diferentes casos que se pueden dar en la modificacion de las semillas.