Random Numbers

- Generate pseudorandom uniformly distributed numbers with the Linear Congruential Method
- Build a frequency table and the histogram of the frequency of the generated numbers.
- Use the Chi-Squared Test to check if the generated numbers are uniformly distributed
- Use the Run Test to check the randomness of the generated numbers

Import python libraries

```
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

Load Excel with random numbers generated with the *Linear Congruential Method*

```
from google.colab import drive

drive.mount("/content/gdrive")
!pwd
```

```
Mounted at /content/gdrive
/content
```

```
%cd "/content/gdrive/My Drive/IntelligentSystems/RandomNumbers"
!ls
```

```
/content/gdrive/My Drive/IntelligentSystems/RandomNumbers
RandomNumbers.ipynb  random_numbers.xlsx  results.txt
```

```
random_df = pd.read_excel("random_numbers.xlsx", usecols=["xi","R"])
random_df.head(12)
```

	xi	R
0	2074941799	0.966220
1	559872160	0.260711
2	1645535613	0.766262
3	1222641625	0.569337
4	1814256879	0.844829
5	95061600	0.044267
6	2119961479	0.987184
7	1291390176	0.601350
8	1924951450	0.896375

```
random_df.shape
```

```
(100, 2)
```

11 1260672530 0.587046

```
constants_df = pd.read_excel("random_numbers.xlsx", usecols=["m","a", "c", "x0"])
constants_df = constants_df.iloc[0]
constants_df.head()
```

```
m     2.147484e+09
a     1.680700e+04
c     0.000000e+00
x0    1.234570e+05
Name: 0, dtype: float64
```

Make Classes Frequency Table and Histogram

```
N = random_df.shape[0]
N
```

```
100
```

```
DECIMALS = 6
DECIMALS
```

```
6
```

```
MAX = round(random_df["R"].max(), DECIMALS)
MAX
```

```
0.987184
```

```
MIN = round(random_df["R"].min(), DECIMALS)
MIN
```

```
0.015981
```

```
C = math.ceil(1 + 3.3 * math.log10(N))
C
```

```
8
```

```
W = round((MAX-MIN)/C, DECIMALS)
W
```

```
0.1214
```

```
SUB_W = 0.000001
SUB_W
```

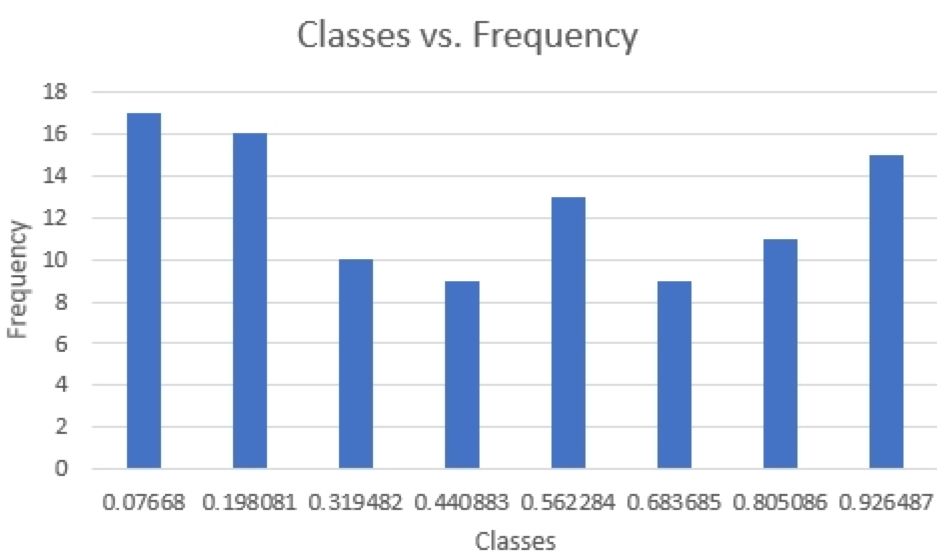
```
1e-06
```

Frequency Table

```
classes_df = pd.DataFrame(columns=["Class Number", "Lower Limit", "Upper Limit", "m", "f"])
classes_df["Class Number"] = np.arange(1, C + 2, 1)
classes_df = classes_df.set_index("Class Number")

classes_df["Lower Limit"] = np.arange(MIN, MAX + W, W + SUB_W)
classes_df["Lower Limit"] = classes_df["Lower Limit"] - SUB_W
classes_df["Lower Limit"].loc[1] = classes_df["Lower Limit"].loc[1] - SUB_W

classes_df["Upper Limit"] = classes_df["Lower Limit"] + W
classes_df["Upper Limit"].loc[1] = classes_df["Upper Limit"].loc[1] - SUB_W

classes_df = classes_df.drop(classes_df.index[C])

classes_df["m"] = (classes_df["Lower Limit"] + classes_df["Upper Limit"]) / 2

for i in range(1, C + 1, 1):
  classes_df["f"][i] = len(
      random_df[
        (random_df["R"] >= classes_df.loc[i]["Lower Limit"]) &
        (random_df["R"] <= classes_df.loc[i]["Upper Limit"])]
  )

classes_df
```

Class Number	Lower Limit	Upper Limit	m	f
1	0.015979	0.137378	0.076678	17
2	0.137381	0.258781	0.198081	16
3	0.258782	0.380182	0.319482	10
4	0.380183	0.501583	0.440883	9
5	0.501584	0.622984	0.562284	13
6	0.622985	0.744385	0.683685	9
7	0.744386	0.865786	0.805086	11
8	0.865787	0.987187	0.926487	15

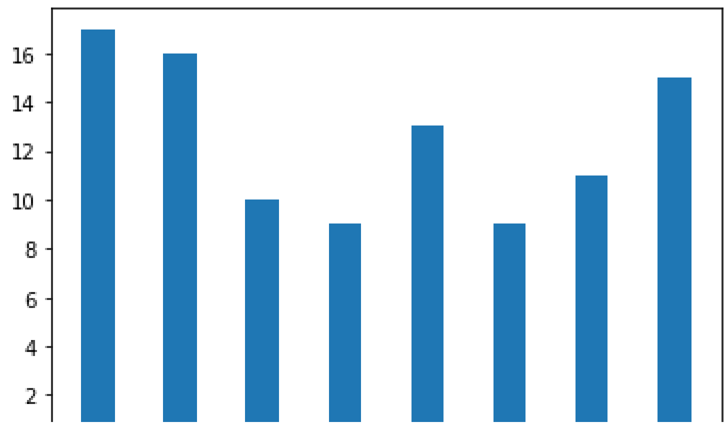
Excel Histogram



Matplotlib Histogram

Only takes two lines of code and no creation of table

```
fig, axs = plt.subplots()
axs.hist(random_df["R"], bins=8, rwidth=0.4)
plt.show()
```



Chi-squared Test

TEST FOR UNIFORMITY
Chi-Squared

$$\chi_0^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

La prueba se aplica a una muestra de datos

Los datos se dividen en *n clases o intervalos*

O_i observed number in the i_{th} ***class***
E_i expected number in the i_{th} ***class***(uniform distribution)
n: number of classes

```
CHI2_CLASSES = 10
ALPHA = 0.05
DEGREES_OF_FREEDOM = CHI2_CLASSES - 1
H0 = "With an alpha of " + str(ALPHA) + " Ri ~ Uniform[0,1], H0 is not rejected"
H1 =  "With an alpha of " + str(ALPHA) + " Ri !~ Uniform[0,1], H0 is rejected"
```

```
# Define our DataFrame
chi2_df = pd.DataFrame(columns=["Class Number", "Lower Limit", "Upper Limit", "O", "E", "O-E"

# There are always 10 classes
chi2_df["Class Number"] = np.arange(1, CHI2_CLASSES + 1, 1)

# Set the dataframe index to the class number
chi2_df = chi2_df.set_index("Class Number")

# Define Lower Limits for each class
chi2_df["Lower Limit"] = np.arange(0, 1, 0.1)

# Difine Upper Limits for each class
chi2_df["Upper Limit"] = chi2_df["Lower Limit"] + 0.1
```

```
# COUNTIF - Count the number of random numbers frequency for each class
for i in range(1, CHI2_CLASSES + 1, 1):
  chi2_df["O"][i] = len(
      random_df[
        (random_df["R"] >= chi2_df.loc[i]["Lower Limit"]) &
        (random_df["R"] <= chi2_df.loc[i]["Upper Limit"])]
)

# Establish E =  N / Classes
chi2_df["E"] = math.ceil(N/CHI2_CLASSES)

# Simple O-E operation
chi2_df["O-E"] = (chi2_df["O"] - chi2_df["E"])

# Simple (O-E)^2
chi2_df["(O-E)^2"] = chi2_df["O-E"] * chi2_df["O-E"]

# (O-E)^2 / E
chi2_df["(O-E)^2/E"] = chi2_df["(O-E)^2"] / chi2_df["E"]

chi2_df
```

Class Number	Lower Limit	Upper Limit	O	E	O-E	(O-E)^2	(O-E)^2/E
1	0.0	0.1	13	10	3	9	0.9
2	0.1	0.2	14	10	4	16	1.6
3	0.2	0.3	11	10	1	1	0.1
4	0.3	0.4	7	10	-3	9	0.9
5	0.4	0.5	7	10	-3	9	0.9
6	0.5	0.6	10	10	0	0	0
7	0.6	0.7	9	10	-1	1	0.1
8	0.7	0.8	7	10	-3	9	0.9
9	0.8	0.9	15	10	5	25	2.5
10	0.9	1.0	7	10	-3	9	0.9

```
## Make sure the frequencies sum up to 100
print(chi2_df["O"].sum())
```

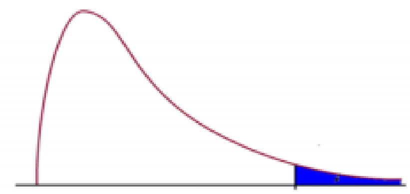
```
100
```

```
## Get x^2
X2 = chi2_df["(O-E)^2/E"].sum()
print(X2)
```

```
8.8
```

Distribución Chi-cuadrada

En las columnas se encuentran las áreas bajo la curva a la derecha.



g.l.	$x^2_{0.995}$	$x^2_{0.990}$	$x^2_{0.975}$	$x^2_{0.95}$	$x^2_{0.9}$	$x^2_{0.1}$	$x^2_{0.05}$	$x^2_{0.025}$	$x^2_{0.01}$	$x^2_{0.005}$
1	3.9E-05	0.0002	0.0010	0.0039	0.0158	2.7055	3.8415	5.0239	6.6349	7.8794
2	0.0100	0.0201	0.0506	0.1026	0.2107	4.6052	5.9915	7.3778	9.2103	10.5966
3	0.0717	0.1148	0.2158	0.3518	0.5844	6.2514	7.8147	9.3484	11.3449	12.8382
4	0.2070	0.2971	0.4844	0.7107	1.0636	7.7794	9.4877	11.1433	13.2767	14.8603
5	0.4117	0.5543	0.8312	1.1455	1.6103	9.2364	11.0705	12.8325	15.0863	16.7496
6	0.6757	0.8721	1.2373	1.6354	2.2041	10.6446	12.5916	14.4494	16.8119	18.5476
7	0.9893	1.2390	1.6899	2.1673	2.8331	12.0170	14.0671	16.0128	18.4753	20.2777
8	1.3444	1.6465	2.1797	2.7326	3.4895	13.3616	15.5073	17.5345	20.0902	21.9550
9	1.7349	2.0879	2.7004	3.3251	4.1682	14.6837	$\boxed{16.9190}$	19.0228	21.6660	23.5894
10	2.1559	2.5582	3.2470	3.9403	4.8652	15.9872	18.3070	20.4832	23.2093	25.1882
11	2.6032	3.0535	3.8157	4.5748	5.5778	17.2750	19.6751	21.9200	24.7250	26.7568

```
X2_ALPHA = 16.9190
print(X2_ALPHA)
```

```
16.919
```

```
CHI2_RESULT = ""
if X2 < X2_ALPHA:
  print(H0)
  CHI2_RESULT = H0
else:
  print(H1)
  CHI2_RESULT = H1
```

```
With an alpha of 0.05 Ri ~ Uniform[0,1], H0 is not rejected
```

Run Test for Randomness

```
H0_R = "With an alpha of " + str(ALPHA) + " Ri ~ Random[0,1], H0 is not rejected"
H1_R = "With an alpha of " + str(ALPHA) + "Ri !~ Random[0,1]"
NUMBER_OF_SIGNS = random_df.shape[0] - 1
print(NUMBER_OF_SIGNS)
```

```
99
```

```
#First we obtain the signs
i = 1
signs = []
while i < random_df.shape[0]:
  if random_df["R"][i - 1] < random_df["R"][i]:
    signs.append('+')
  else:
    signs.append('-')
  i += 1
signs = np.array(signs)
print("*** SIGNS ***")
print(signs)
print(np.array(random_df["R"]))
```

```
*** SIGNS ***
['-' '+' '-' '+' '-' '+' '-' '+' '-' '-' '+' '-' '-' '+' '-' '-' '+' '-'
 '-' '+' '+' '-' '+' '-' '-' '+' '+' '-' '+' '-' '+' '-' '+' '-' '+' '-'
 '-' '+' '+' '+' '+' '-' '-' '+' '+' '+' '-' '+' '-' '+' '+' '+' '-' '+'
 '-' '+' '-' '-' '-' '+' '+' '-' '-' '+' '+' '-' '+' '-' '+' '-' '-' '+'
 '-' '-' '+' '+' '-' '+' '-' '+' '-' '+' '-' '-' '+' '-' '+' '-' '+' '+'
 '+' '-' '-' '-' '+' '-' '+' '-' '+']
[0.96622007 0.26071079 0.76626223 0.56933687 0.84482919 0.04426651
 0.98718399 0.60135041 0.89637537 0.38085417 0.01598067 0.58704639
 0.48873499 0.16904939 0.21307769 0.19676858 0.08951002 0.39497507
 0.34599525 0.14211733 0.56594864 0.89874989 0.28939718 0.89832961
 0.22580152 0.04609446 0.70951826 0.87332853 0.03259875 0.88721835
 0.47876976 0.68338545 0.6592388  0.82646881 0.46122525 0.81285534
 0.65967569 0.1693641  0.50250588 0.61627177 0.67969957 0.71068514
 0.48509855 0.05131961 0.52868335 0.58105119 0.72739263 0.28797253
 0.95436681 0.04290747 0.14579226 0.33051218 0.91825869 0.17378334
 0.77655021 0.47944169 0.97655454 0.95216901 0.10460464 0.09018604
 0.75682055 0.88301642 0.85695542 0.84981705 0.87522339 0.87959793
 0.40242821 0.61100321 0.13100729 0.8395785  0.79580158 0.03716831
 0.68781817 0.15993479 0.02398335 0.08821099 0.56215972 0.21837724
 0.2662666  0.14272865 0.84043751 0.23317539 0.9787961  0.62611924
 0.18614516 0.54174272 0.0698631  0.18909213 0.07136668 0.45978105
 0.54010676 0.57430175 0.2894765  0.2314577  0.10955825 0.34546651
 0.25568983 0.37901484 0.10247951 0.37305509]
```

```
# Calculate R Number of Runs
last_sign = signs[0]
R = 1

if len(signs) == 0:
  R = 0

for sign in signs:
  if sign != last_sign:
    R += 1
    last_sign = sign

print(R)
```

```
70
```

$$\mu_R = E(R) = \frac{2n-1}{3} \qquad \sigma_R^2 = \frac{16n-29}{90}$$

```
MIU_R = ((2*NUMBER_OF_SIGNS) - 1) / 3
print(MIU_R)
```

```
65.66666666666667
```

```
SIGMA_R2 = ((16*NUMBER_OF_SIGNS) - 29) / 90
print(SIGMA_R2)
```

```
17.27777777777778
```

```
SIGMA_R = math.sqrt(SIGMA_R2)
print(SIGMA_R)
```

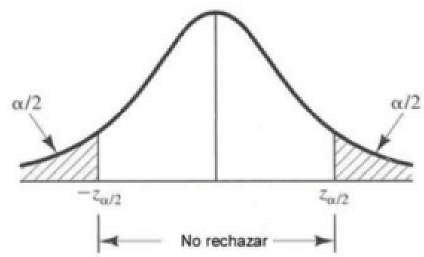
```
4.156654637779975
```

$$Z_R = \frac{R-\mu_R}{\sigma_R} \overset{\approx}{\rightsquigarrow} \mathbf{N}(0,1)$$

```
ZETA_R = (R - MIU_R) / SIGMA_R
print(ZETA_R)
```

```
1.042505021694012
```

With alpha = 0.05



1 - α	α/2	z α/2
0.90	0.05	1.645
0.95	0.025	1.96
0.99	0.005	2.575

Ho is rejected when $|Z_R| > z_{\alpha/2}$

```
ZETA_ALPHA2 = 1.96
RUN_TEST_RESULT = ""
if abs(ZETA_R) > ZETA_ALPHA2:
  print(H1_R)
  RUN_TEST_RESULT = H1_R
else:
  print(H0_R)
  RUN_TEST_RESULT = H0_R
```

```
With an alpha of 0.05 Ri ~ Random[0,1], H0 is not rejected
```

Save Results

```
results_df = pd.DataFrame(columns=["MIU_R", "SIGMA_R", "R", "ZETA_R", "CHI2_RESULT", "RUN_TES
results_df["MIU_R"] = [MIU_R]
results_df["SIGMA_R"] = [SIGMA_R]
results_df["R"] =  [R]
results_df["ZETA_R"] = [ZETA_R]
results_df["CHI2_RESULT"] = [CHI2_RESULT]
results_df["RUN_TEST_RESULT"] = [RUN_TEST_RESULT]
results_df["SIGNS"] = [''.join([sign for sign in signs ])]
results_df.index.name = "Row"
results_df.head()
```

	MIU_R	SIGMA_R	R	ZETA_R	CHI2_RESULT	RUN_TEST_RESULT	SIGNS
Row							
0	65.666667	4.156655	70	1.042505	With an alpha of 0.05 Ri ~ Uniform[0.1]. H0	With an alpha of 0.05 Ri ~ Random[0.1]. H0	-+-+-+-+--+--+--+- -++-+--++-+-+-+-

```
tfile = open('results.txt', 'w')
tfile.write(results_df.to_string())
tfile.close()
```