



TP1 - Simulación

PICCO, MARTÍN ALEJANDRO - 99289

RIPARI, SEBASTIAN DANIEL - 96453

NOCETTI, TOMAS AGUSTIN - 100853

DANERI, ALEJANDRO NICOLAS -97839



Punto 1 - Paper Idea Aplicada

- Generar números pseudoaleatorios con operaciones de Shift - Xor.
- Gran velocidad de operatoria.
- Posibilidad de generalizar a palabras de 98, 128, ...

```
unsigned long long xor64(){  
    static unsigned long long x=88172645463325252LL;  
    x^=(x<<13); x^=(x>>7); return (x^=(x<<17));  
}
```

Implementación XOR Shift

```
class XORShiftGenerator:

    PERIOD = 2**64

    def __init__(self, external_seed):
        self.seed = external_seed % self.PERIOD

    #funcion para generar un numero pseudo-aleatorio
    def random(self):
        seed_tuple = (23,13,58)
        self.seed = (self.seed ^ (self.seed << seed_tuple[0])) % self.PERIOD
        self.seed = (self.seed ^ (self.seed >> seed_tuple[1])) % self.PERIOD
        self.seed = (self.seed ^ (self.seed << seed_tuple[2])) % self.PERIOD
        return self.seed

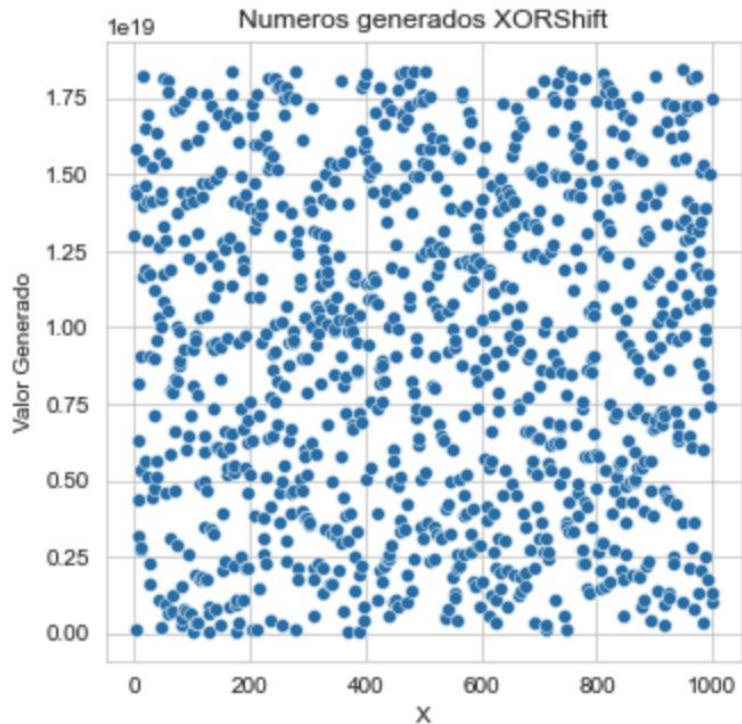
    #funcion para generar un numero pseudo-aleatorio entre 0 y 1
    def uniform_random(self):
        return self.random() / self.PERIOD

    def generate_sample(self, n):
        sample = []
        for i in range(n):
            sample.append(self.random())
        return sample

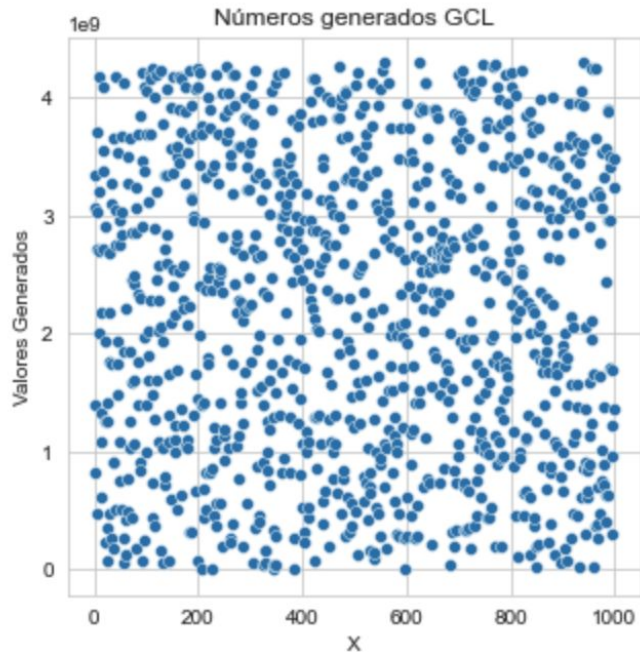
    def generate_uniform_sample(self, n):
        sample = []
        for i in range(n):
            sample.append(self.uniform_random())
        return sample
```

Resultados en Gráficos | Scatter

XORShift

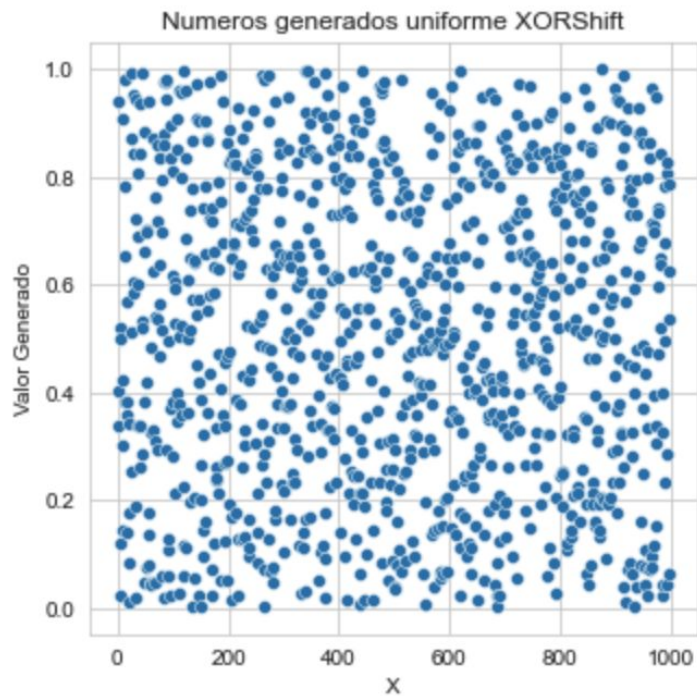


GCL

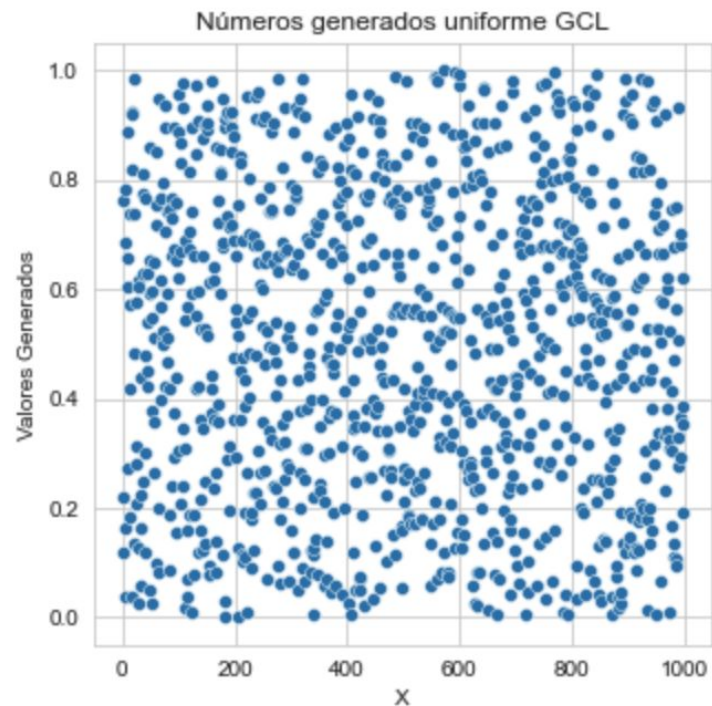


Scatter | Caso [0,1]

XORShift

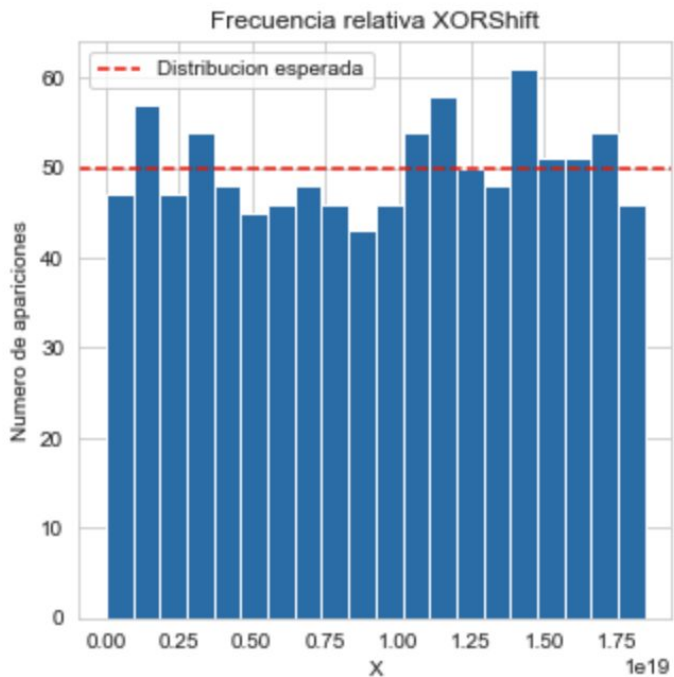


GCL

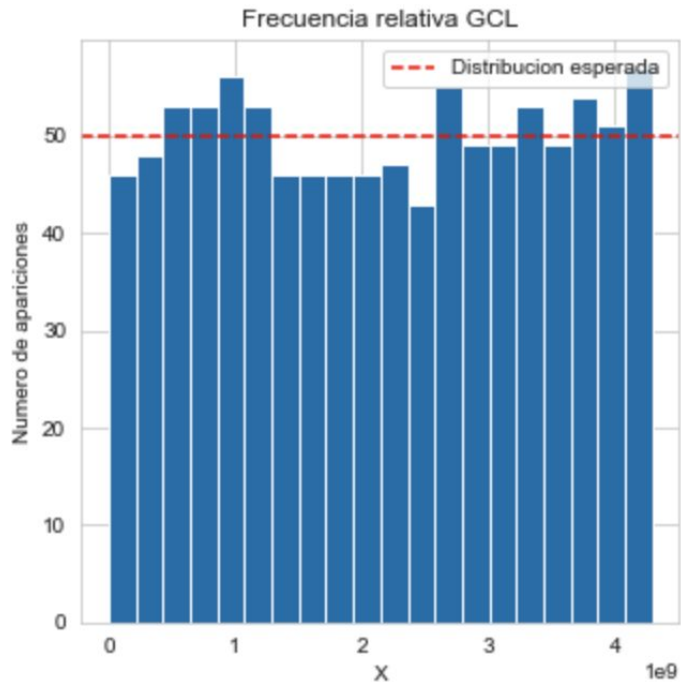


Histograma

XORShift

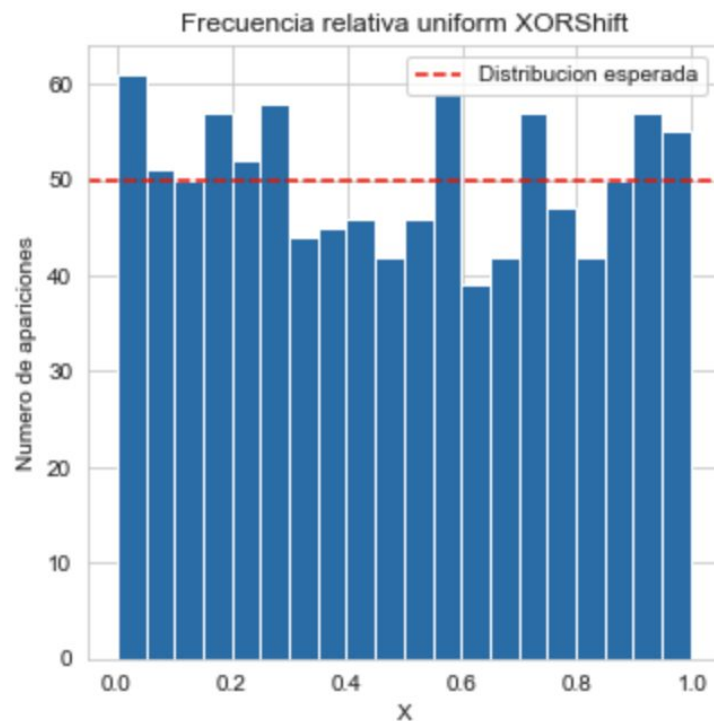


GCL

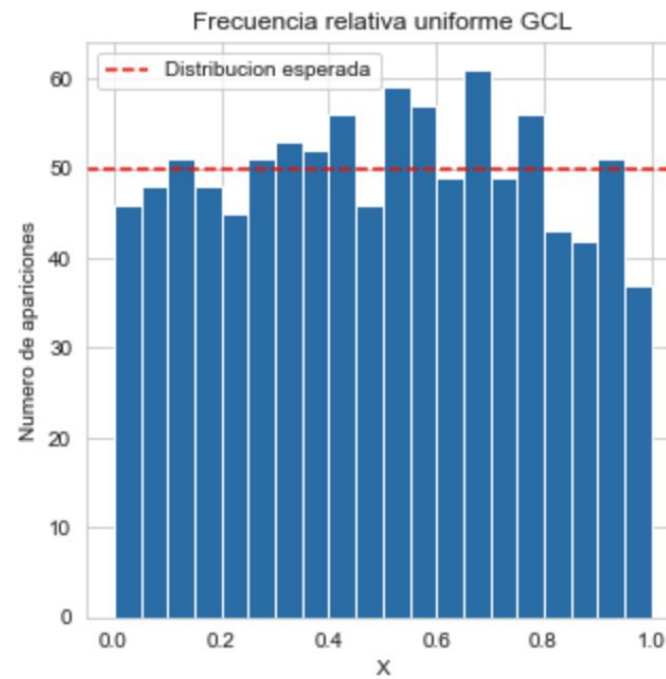


Histograma | Caso [0,1]

XORShift



GCL



Punto 2 - Test Kolmogorov contra uniforme (0.05)



Xor Shift

Muestra 10000 - *Acepta Ho*

- Estadístico: 0.0084
- P Valor: 0.4751

Muestra 1000 - *Acepta Ho*

- Estadístico: 0.0279
- P Valor: 0.4087

GCL

Muestra 10000 - *Acepta Ho*

- Estadístico: 0.0083
- P Valor: 0.4993

Muestra 1000 - *Acepta Ho*

- Estadístico: 0.0296
- P Valor: 0.3367

Punto 2 - Test Independencia - Bernoulli 0.05

Objetivo: probar independencia entre 6 tiradas sucesivas

Xor Shift

Acepta H_0

Limite superior: 9.49

Estadistico: 3.21

Tabla de frecuencias:

	Sale 0	Sale 1
0	3	3
1	3	3
2	3	3
3	5	1
4	2	4

El test acepta la hipotesis nula

GCL

Acepta H_0

Limite superior: 9.49

Estadistico: 1.87

Tabla de frecuencias:

	Sale 0	Sale 1
0	2	4
1	3	3
2	4	2
3	4	2
4	3	3

El test acepta la hipotesis nula

Test Chi2

```
In [86]: generator = XORShiftGenerator(int(time.time()))  
  
         n = 10000  
         k = 10  
         p = 1/k  
  
         sample = generator.generate_uniform_sample(n)  
  
         str(sample[0:5]) + '...'
```

```
Out[86]: '[0.06325734080860392, 0.9421685011010822, 0.7224391500943322, 0.5837709911457596, 0.7846551093616525]...'
```

Se requiere discretizar !

Test Chi2

Discretizacion de la muestra

```
In [87]: def in_range(value, start, end):
          return value > start and value <= end

          # asigna un valor (a.k.a value) a un bind
          # k es la cantidad de binds
          def mapper(value, k):


              width = 1.0 / k

              for i in range(k):
                  start = i * width
                  end = (i + 1) * width
                  if in_range(value, start, end): return i


              raise Exception("number generated is invalid")

          def frequency_sum(values):
              elements_count = collections.Counter(values)
              count = []
              for key, value in elements_count.items():
                  count.insert(key, value)
              return count
```

```
In [88]: mapped_values = list(map(lambda value: mapper(value, k), sample))
          "{...}".format(str(mapped_values[0:5]))
```

```
Out[88]: '[0, 9, 7, 5, 7]...' 
```

```
In [89]: frecuencias = frequency_sum(mapped_values)
          frecuencias
```

```
Out[89]: [1040, 985, 1002, 990, 1007, 1000, 986, 939, 1017, 1034] 
```

Frecuencias Actuales

```
In [91]: frecuencias
```

```
Out[91]: [1040, 985, 1002, 990, 1007, 1000, 986, 939, 1017, 1034]
```

Frecuencias Esperadas

```
In [92]: f_e = int(n * p)
          [f_e, f_e, f_e, f_e, f_e, f_e, f_e, f_e, f_e, f_e]
```

```
Out[92]: [1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000]
```

Calculemos la dispersion:

$$D^2 = \sum_{i=1}^K \frac{(N_i - np_i)^2}{np_i}$$

```
def dispersion(n, p, frecuencias):  
    k = len(frecuencias)  
  
    sum_ = 0  
  
    dispersiones = []  
  
    for i in range(k):  
        dispersion_i = (frecuencias[i] - n * p) ** 2 / (n * p)  
        dispersiones.append(dispersion_i)  
        sum_ = sum_ + dispersion_i  
  
    print("dispersiones: {}".format(dispersiones))  
  
    return sum_
```

```
In [95]: test_chi2(frecuencias, k)
```

```
dispersiones: [1.6, 0.225, 0.004, 0.1, 0.049, 0.0, 0.196, 3.721, 0.289, 1.156]  
7.34 16.918977604620448  
El test acepta la hipotesis nula
```



Chi2 | Resultados

Cota superior=16.9189

```
def test_chi2(frecuencias, k):  
    upper_limit = chi2.ppf(0.95, df=k-1)  
    d2 = dispersion(n, 1/k, frecuencias)  
    return d2 < upper_limit
```

Xor Shift

Muestra 10000 - *Acepta Ho*

- Estadístico: 5.388

Muestra 1000 - *Acepta Ho*

- Estadístico: 10.36

GCL

Muestra 10000 - *Acepta Ho*

- Estadístico: 6.016

Muestra 1000 - *Acepta Ho*

- Estadístico: 8.719

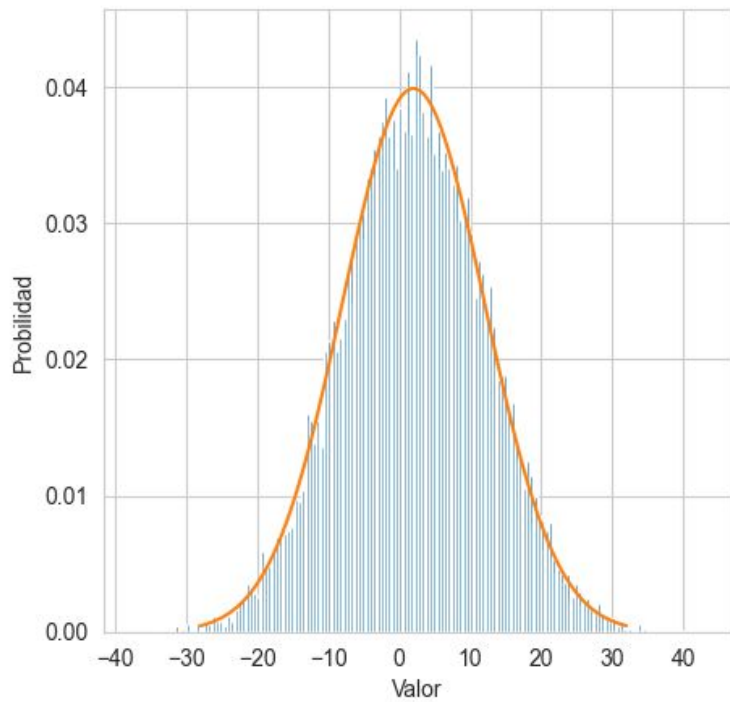
Punto 3 - Generar Normal



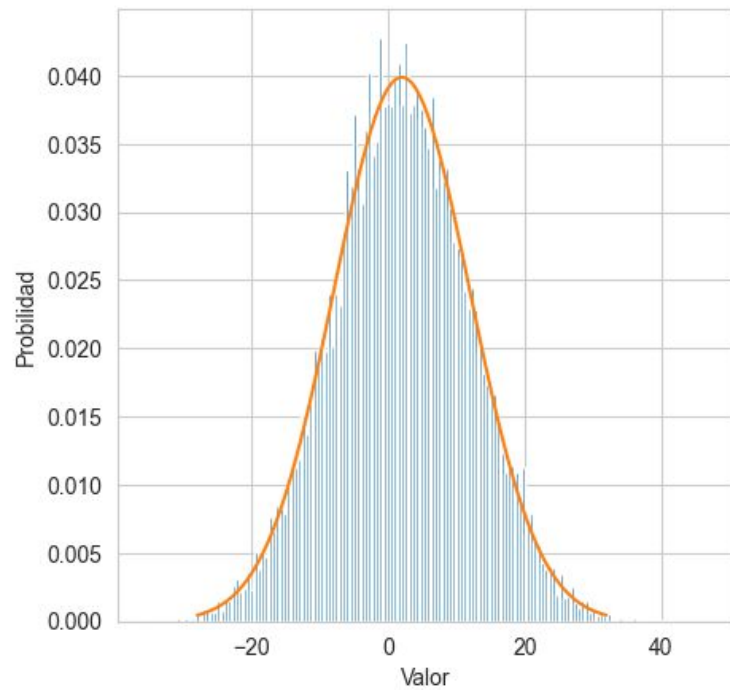
- Método Utilizado: **Aceptación/Rechazo** con Exponencial
- Pasos:
 - 1) Generar una muestra de N exponenciales a partir de un valor aleatorio mediante el método de transformada inversa.
 - 2) Calcular para cada valor “ t ” la probabilidad de aceptar realizando el cociente $f_x(t) / c * f_y(t)$; con f_x una p.d.f normal standard, y f_y una p.d.f un $\exp(1)$
 - 3) Generar un valor aleatorio “ r_1 ” y compararlo contra la probabilidad calculada. Si $r_1 < P(\text{aceptar})$, aceptamos el valor; si no, lo rechazamos
 - 4) Generar un nuevo valor random “ r_2 ”. Si $r_2 < 0.5$, tomamos t como negativo, lo que nos permite ubicarlo del lado izquierdo de la campana
 - 5) Finalmente, calcular el valor para la $N(2, 10)$ desplazando por la media y escalando por el desvío ($v = \sigma * x + \mu$)
- Eficiencia con XorShift: ~38%
- Eficiencia con GCL: ~37.8%



Xor Shift



GCL



Punto 3 - Tests Estadísticos



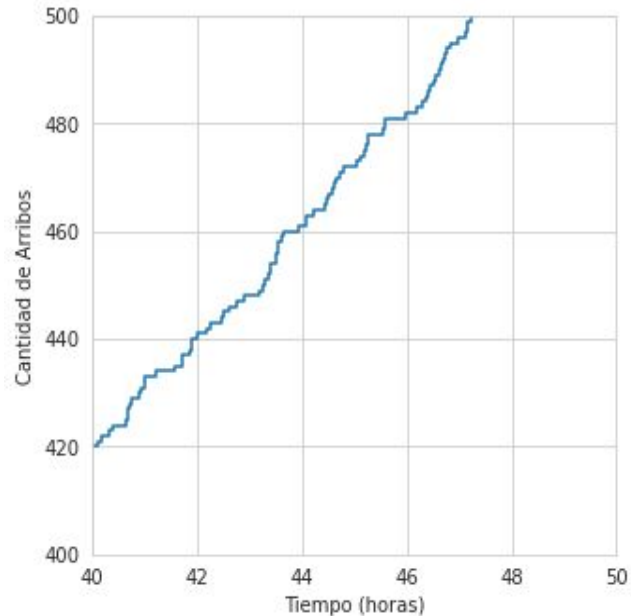
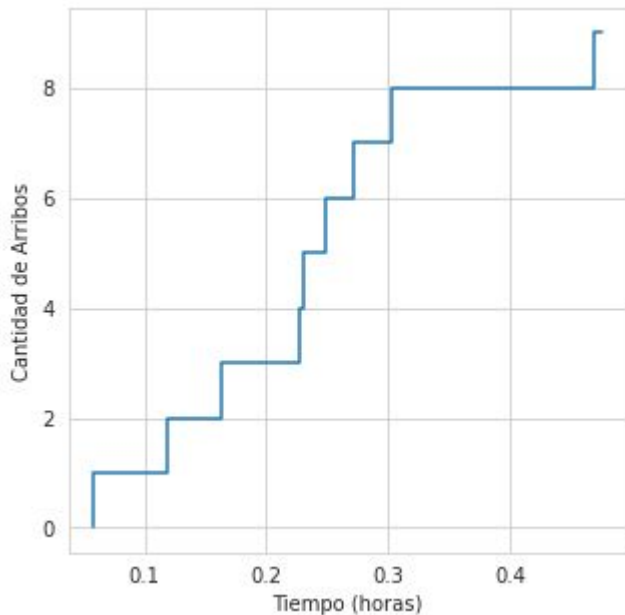
Para ambos tests nuestra H_0 es que los valores generados siguen una distribución $N(2, 10)$. Mientras que H_1 será que no sigue tal distribución. Los tests nos permiten determinar si la evidencia es suficiente para rechazar H_0 y decir que no se distribuye de tal manera.

Para ambos tests asumimos un nivel de significación $\alpha=0.05$, que compararemos frente al p-valor del test.

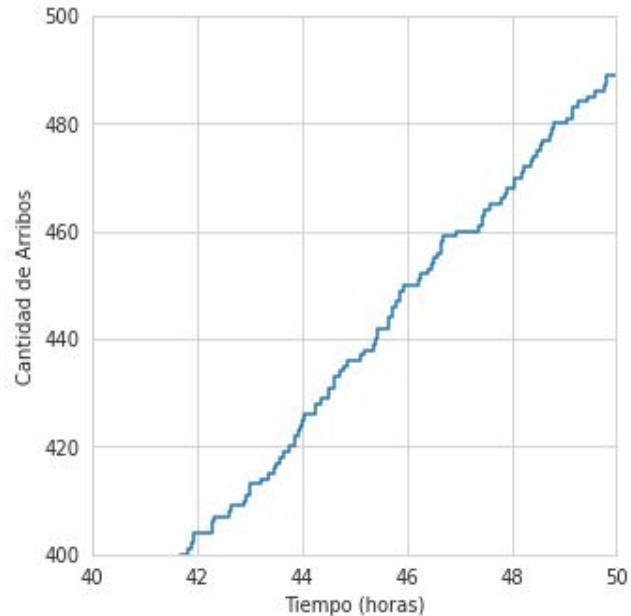
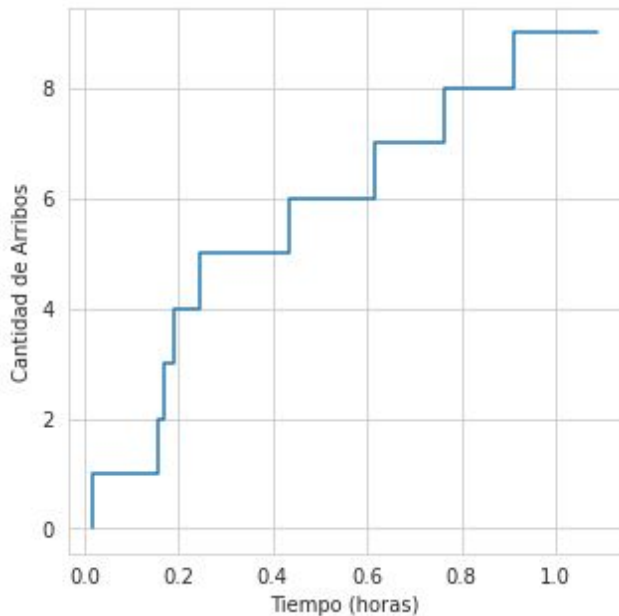
- Kolmogorov-Smirnov: $p\text{-}V=0.623 \Rightarrow$ No se puede rechazar H_0
- Shapiro-Wilk: $p\text{-}V=0.137 \Rightarrow$ No se puede rechazar H_0

Encontramos que ninguno de ambos rechaza H_0 , por lo que se puede seguir sosteniendo que los valores generados respetan una distribución $N(2, 10)$

Simulación de llegada de vehículos | XORShift



Simulación de llegada de vehículos | GCL



Probabilidad de que el primer vehículo arribe antes de los 10'



$$P(X_1 < 0.16) = 1 - P(X_1 > 0.16)$$

$$P(N_{(0,0.16]} = 0) = \frac{e^{-\lambda 0.16} (\lambda 0.16)^0}{0!}$$

Método	Resultado
XORShift	0.814
GCL	0.787
Teórico	0.7944938584160515

Probabilidad de que el undécimo vehículo arribe después de los 60'



$$P(X_{11} > 1)$$

$$P(N_{(0,1]} \leq 10) = \sum_{i=0}^{10} P(N_{(0,1]} = i)$$

Método	Resultado
XORShift	0.598
GCL	0.611
Teórico	0.4718605839507912

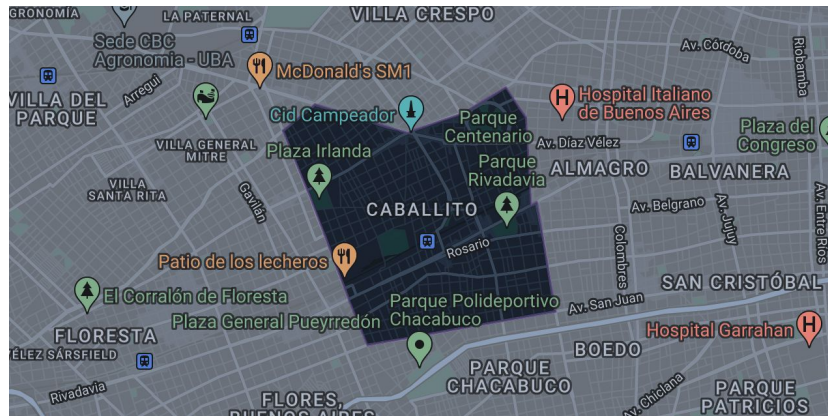
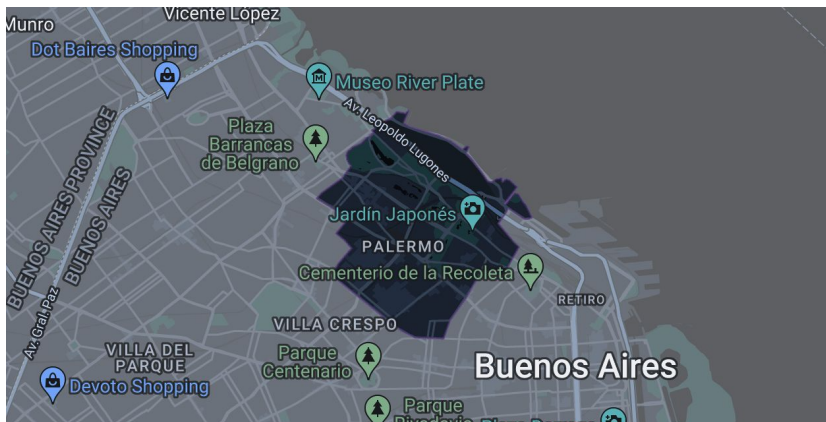
Probabilidad de que arriben al menos 750 vehículos antes de las 72 hs



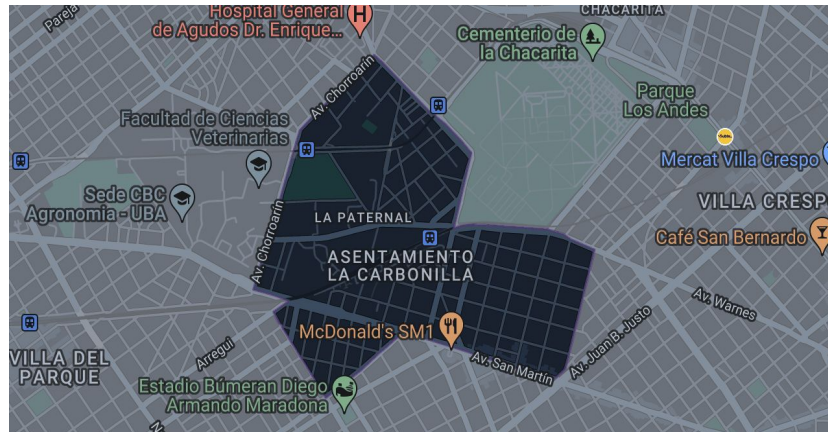
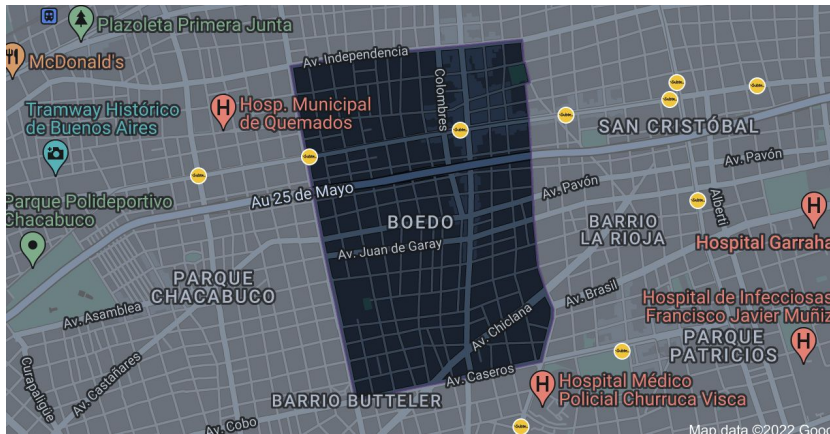
$$P(N_{(0,72]} \leq 750) = \sum_{i=0}^{750} P(N_{(0,72]} = i)$$

Método	Resultado
XORShift	0.082
GCL	0.079
Teórico	~ 0

Barrios CABA | GeoJSON



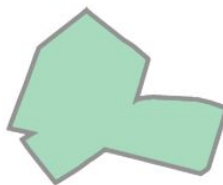
Generar puntos dentro de un barrio determinado



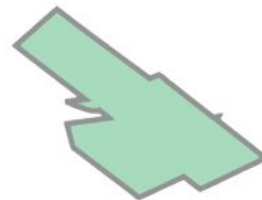
```
{
  "type": "FeatureCollection",
  "name": "barrios",
  "crs": {
    "type": "name",
    "properties": {
      "name": "urn:ogc:def:crs:OGC:1.3:CRS84"
    }
  },
  "features": [
    {
      "type": "Feature",
      "properties": {
        "BARRIO": "CHACARITA",
        "COMUNA": 15.0,
        "PERIMETRO": 7724.8529545700003,
        "AREA": 3115707.1062699999,
        "OBJETO": "BARRIO"
      },
      "geometry": {
        "type": "MultiPolygon",
        "coordinates": [
          [
            [
              [
                -58.452820049279104,
                -34.59598865706387
              ],
              [
                -58.45365519313701,
                -34.596555716304088
              ],
              ...
            ]
          ]
        ]
      }
    },
    ...
  ]
}
```

```
def get_polygon_barrio_by_name(name):
    barrios = data["features"]
    for barrio in barrios:
        if barrio["properties"]["BARRIO"] == name:
            coordinates = barrio["geometry"]["coordinates"][0][0]
            print("coordinates: {} ...".format(coordinates[0:5]))
            return Polygon(coordinates)
        raise Exception("barrio not found")
```

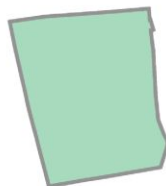
```
get_polygon_barrio_by_name("PATERNAL")
```



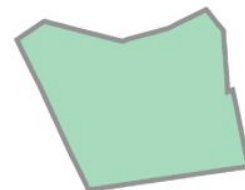
```
get_polygon_barrio_by_name("VILLA LURO")
```



```
get_polygon_barrio_by_name("BOEDO")
```



```
get_polygon_barrio_by_name("CABALLITO")
```



Generación de Puntos

```
def random_points_in_bounds(polygon, number, generator):
    points = []
    attempts = 0
    minx, miny, maxx, maxy = polygon.bounds
    while len(points) < number:
        x = minx + ((maxx - minx) * generator.uniform_random())
        y = miny + ((maxy - miny) * generator.uniform_random())
        point = Point(x, y)
        attempts = attempts + 1
        if polygon.contains(point):
            points.append(point)


    # rendimiento
    polygon_area = polygon.area
    polygon_bounday_area = (maxx - minx) * (maxy - miny)
    rendimiento_teorico = polygon_area / polygon_boundary_area
    rendimiento_real = number / attempts

    prints(...)

    return points
```

```
points = random_points_in_bounds(polygon, 20, generator)
points_gcl = random_points_in_bounds(polygon, 20, generator_gcl)
```

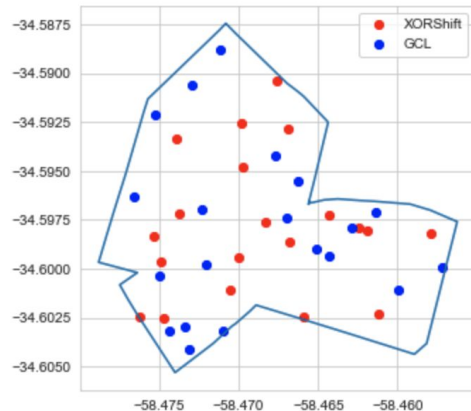
Resultados

	Rendimiento teórico		Attempts
Boedo	75.8%		28
Caballito	66.98%		35
Palermo	59.03%		37
Paternal	54.28%		37
Villa Luro	41.0%		50

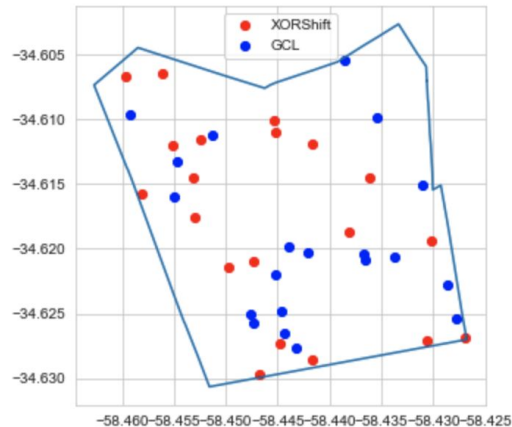


Visualizaciones

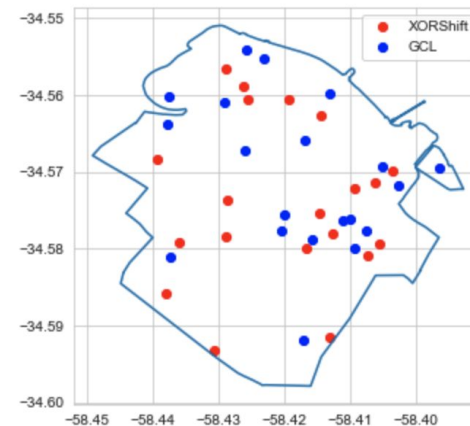
Paternal



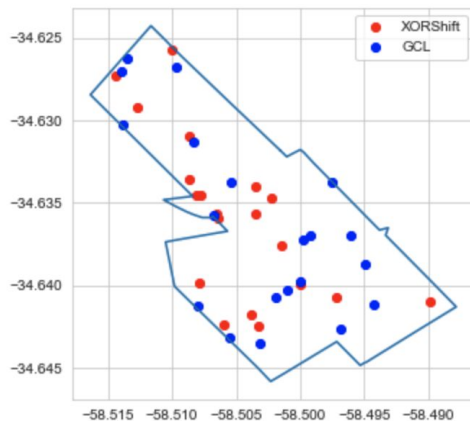
Caballito



Palermo



Villa Luro



Boedo

