

Números aleatorios

Generación de números aleatorios

Ordenador → generadores de números pseudo-aleatorios (PRNG)

C++: `rand()`
(entero entre 0 y `RAND_MAX`)

Python: `random()`
(real entre 0 y 1)

Requisito: los números generados deben estar uniformemente distribuidos.

Generador congruencial lineal:

$$x_{n+1} = (ax_n + c) \bmod(m)$$

Se necesita un valor inicial o semilla.

En Python es un valor relacionado con la hora del sistema.

En C++ está predeterminado ($x_0=1$), pero se puede cambiar con

`srand(time(0))`

Números aleatorios

Generación de números aleatorios

Source	m	a	c	output bits of seed in <i>rand()</i> / <i>Random(L)</i>
<i>Numerical Recipes</i>	2^{32}	1664525	1013904223	
Borland C/C++	2^{32}	22695477	1	bits 30..16 in <i>rand()</i> , 30..0 in <i>lrand()</i>
<i>glibc</i> (used by GCC) ^[4]	2^{32}	1103515245	12345	bits 30..0
ANSI C: Watcom, Digital Mars, CodeWarrior, IBM VisualAge C/C++	2^{32}	1103515245	12345	bits 30..16
Borland Delphi, Virtual Pascal	2^{32}	134775813	1	bits 63..32 of (<i>seed</i> * <i>L</i>)
Microsoft Visual/Quick C/C++	2^{32}	214013	2531011	bits 30..16
Microsoft Visual Basic (6 and earlier) ^[5]	2^{24}	1140671485	12820163	
RtlUniform from Native API ^[6]	$2^{31} - 1$	2147483629	2147483587	
Apple CarbonLib	$2^{31} - 1$	16807	0	see MINSTD
MMIX by Donald Knuth	2^{64}	6364136223846793005	1442695040888963407	
VAX's MTH\$RANDOM, ^[7] old versions of <i>glibc</i>	2^{32}	69069	1	
Java's java.util.Random	2^{48}	25214903917	11	bits 47...16
LC53 ^[8] in Forth	$2^{32} - 5$	$2^{32} - 333333333$	0	

Números aleatorios

Generación de números aleatorios

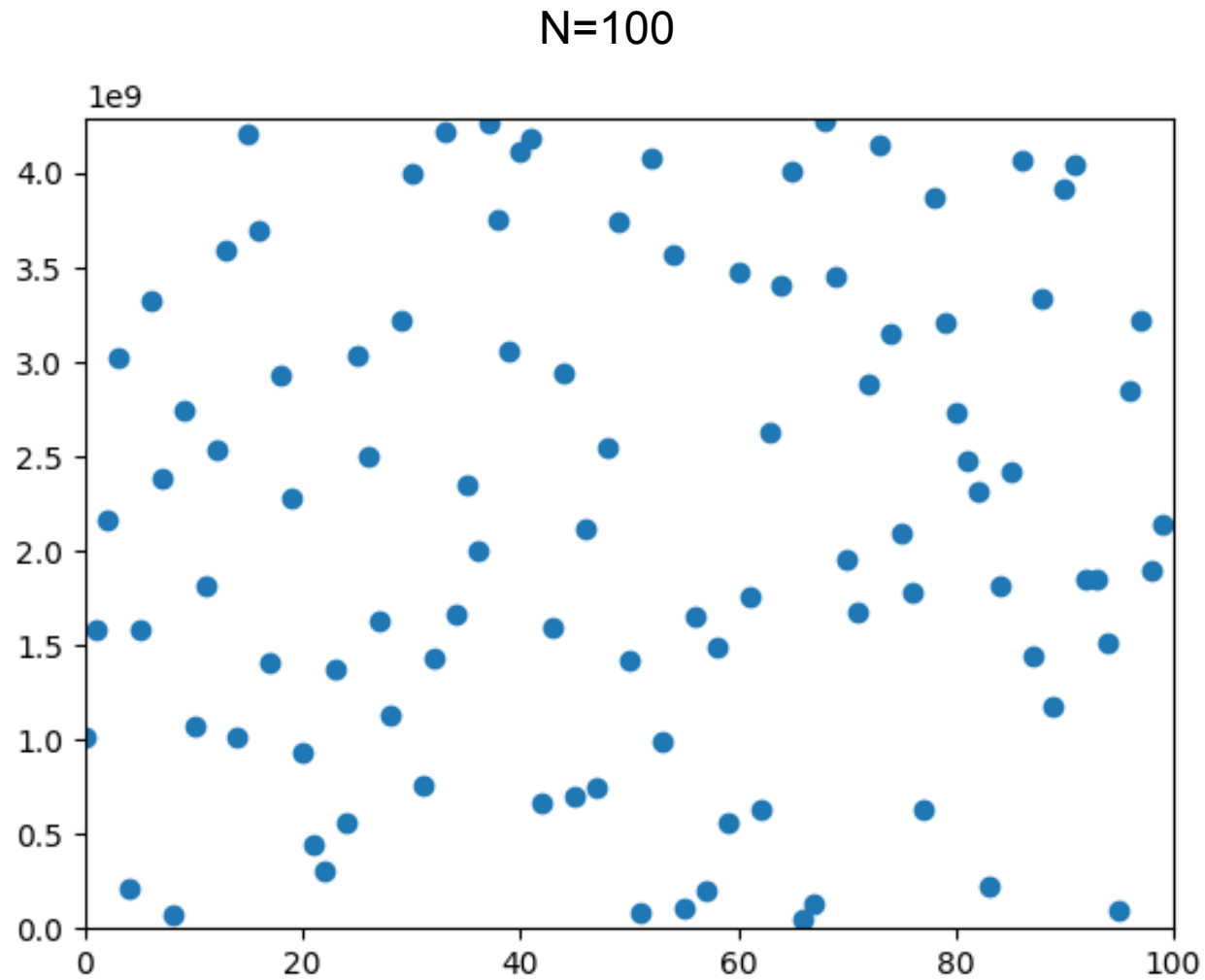
Código en Python para implementar el generador congruencial lineal

```
# Programa para generar números aleatorios por el método congruencial lineal
from matplotlib.pyplot import plot, show
# Establecemos las constantes del método
a=1664525
c=1013904223
m=4294967296

N=100          # Número de puntos
x=1           # Semilla
results=[]     # Lista para guardar los valores
for i in range(N):
    x=(a*x+c)%m
    results.append(x)
# Hacemos una gráfica con los resultados:
plot(results,"o")
show()
```

Números aleatorios

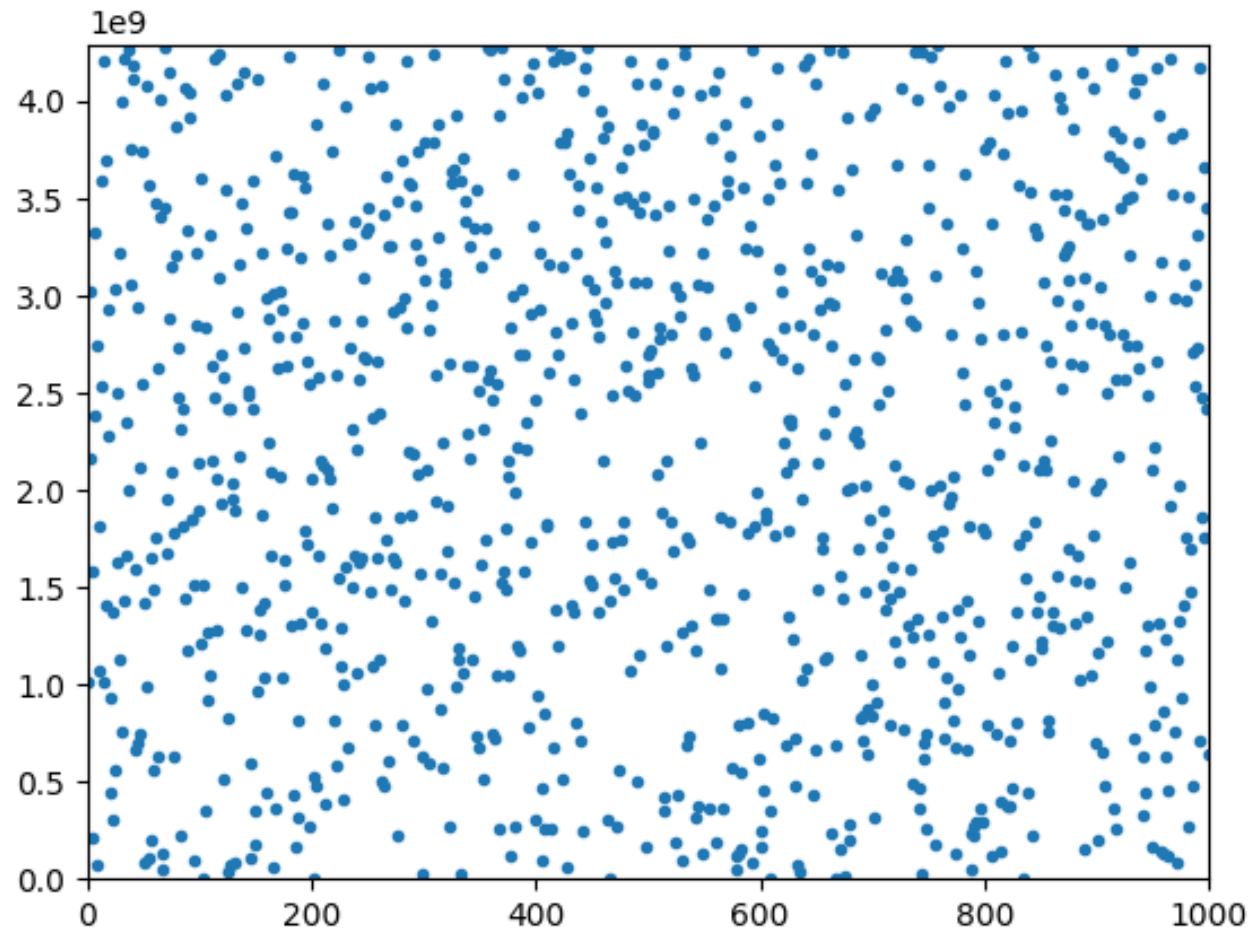
Generación de números aleatorios



Números aleatorios

Generación de números aleatorios

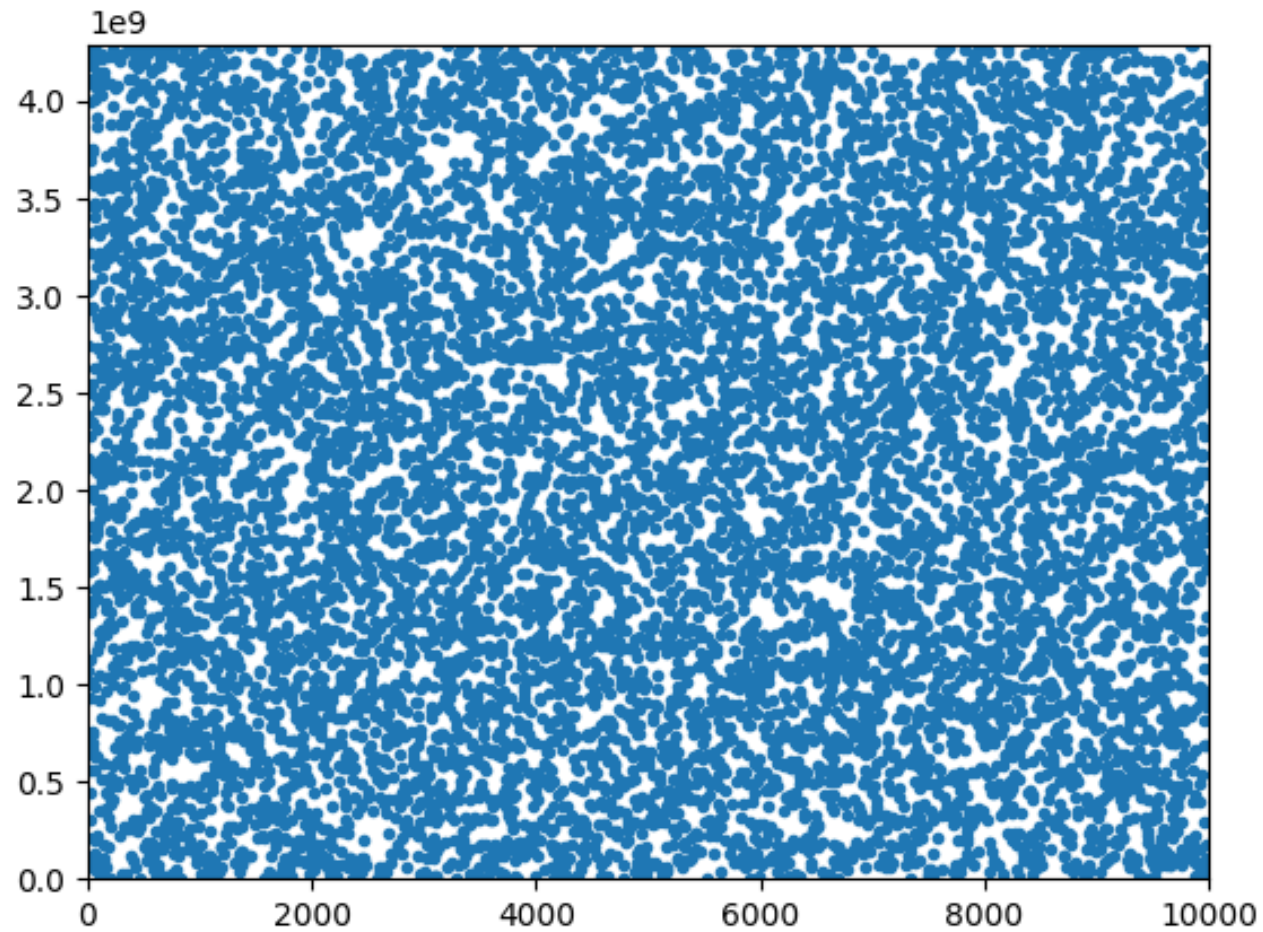
N=1000



Números aleatorios

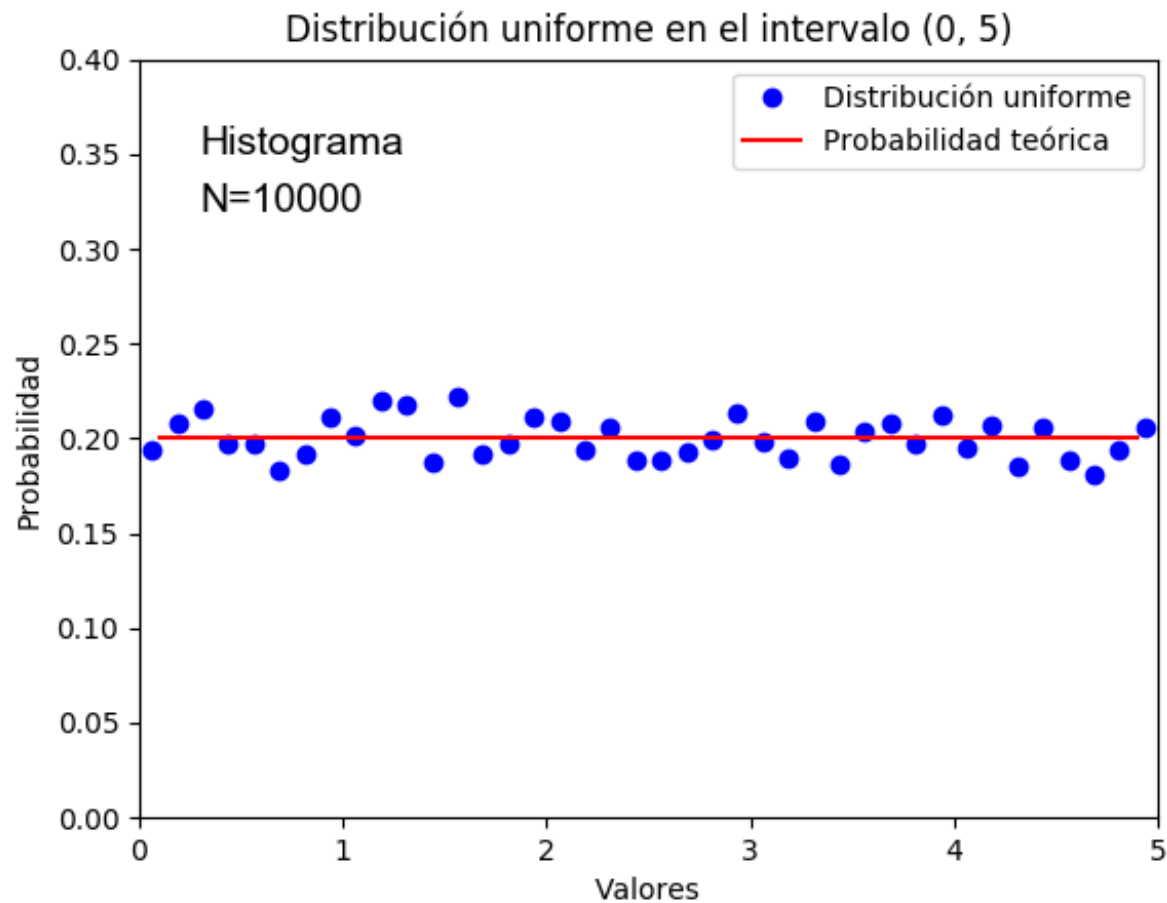
Generación de números aleatorios

N=10000



Números aleatorios

Generación de números aleatorios



$$P = P(x) dx$$

Números aleatorios

Generación de números aleatorios

En realidad, Python no usa el generador congruencial lineal, sino otro generador denominado "Mersenne Twister", o Tornado de Mersenne.

Está basado en los números primos de Mersenne: $2^n - 1$

El más utilizado usa $2^{19937} - 1$. En C++ se puede utilizar con la función "mt19937".

Para usar este generador de números aleatorios en Python tenemos que importar el paquete "random":

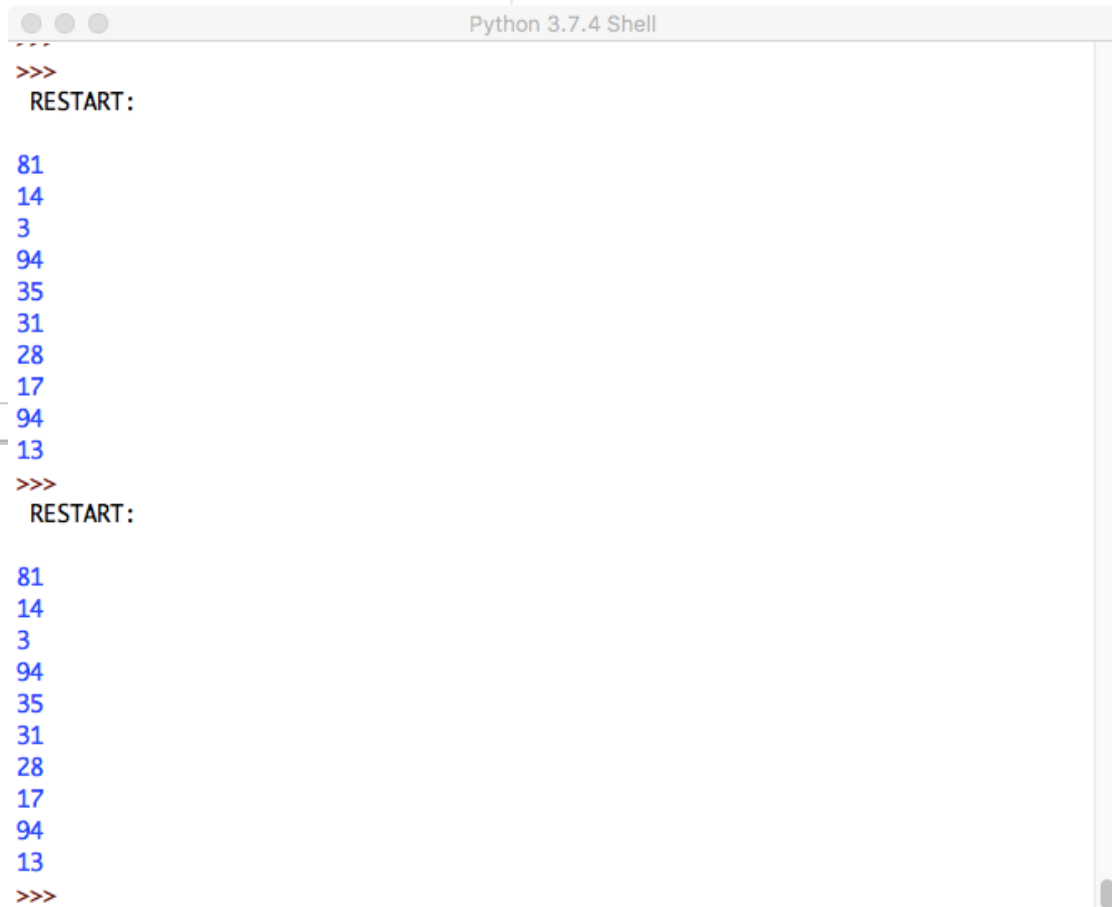
```
from random import *
a=1
lista=["a","e","i","o","u"]
while a!=0:
    print("Número real entre 0 y 1, el 1 no cuenta:", random() )
    print("Número entero entre 0 y 99:", randrange(100) )
    print("Número entero entre 50 y 99:", randrange(50,100) )
    print("Número entero entre 50 y 99, múltiplo de 5:", randrange(50,100,5) )
    print("Número real entre 50 y 100, el 100 no cuenta:", uniform(50,100) )
    print("Elemento al azar de la lista",lista,":", choice(lista) )
    print("Número entero entre 50 y 100, el 100 sí que entra:", randint(50,100) )
    a=int(input("\nPara parar introduce 0, para seguir, cualquier otro número:"))
```

Para fijar la semilla en Python a un valor fijo: `seed()`. Útil para depurar.

Números aleatorios

Generación de números aleatorios

```
from random import *  
  
seed(42)  
for i in range(10):  
    print(randrange(100))
```



```
Python 3.7.4 Shell  
>>>  
RESTART:  
  
81  
14  
3  
94  
35  
31  
28  
17  
94  
13  
>>>  
RESTART:  
  
81  
14  
3  
94  
35  
31  
28  
17  
94  
13  
>>>
```

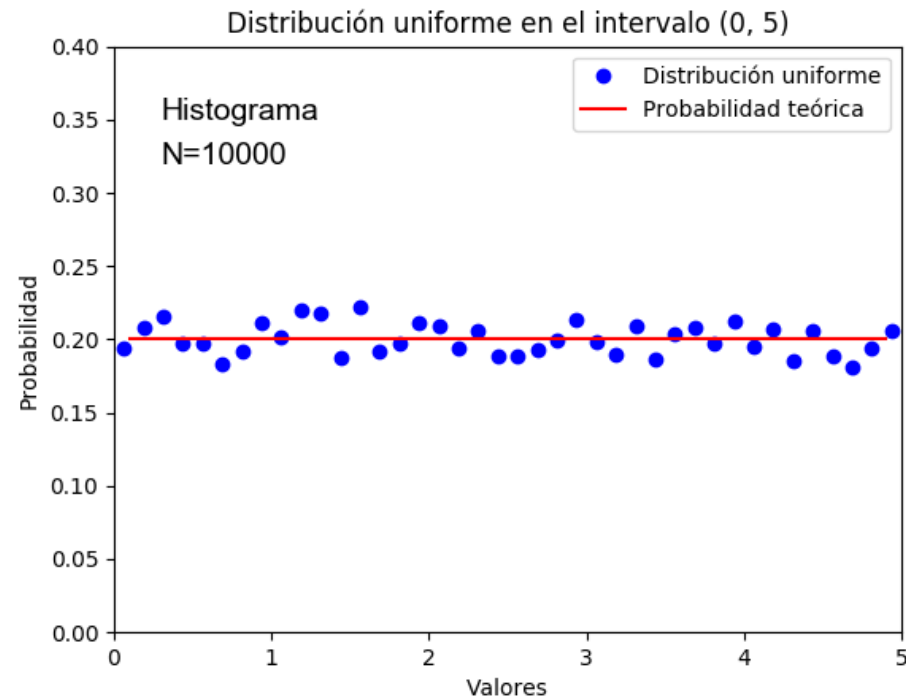
Ln: 76180 Col: 0

Números aleatorios

Generación de números aleatorios

Distribuciones de números aleatorios no uniformes

No siempre se busca generar una secuencia de números uniforme.

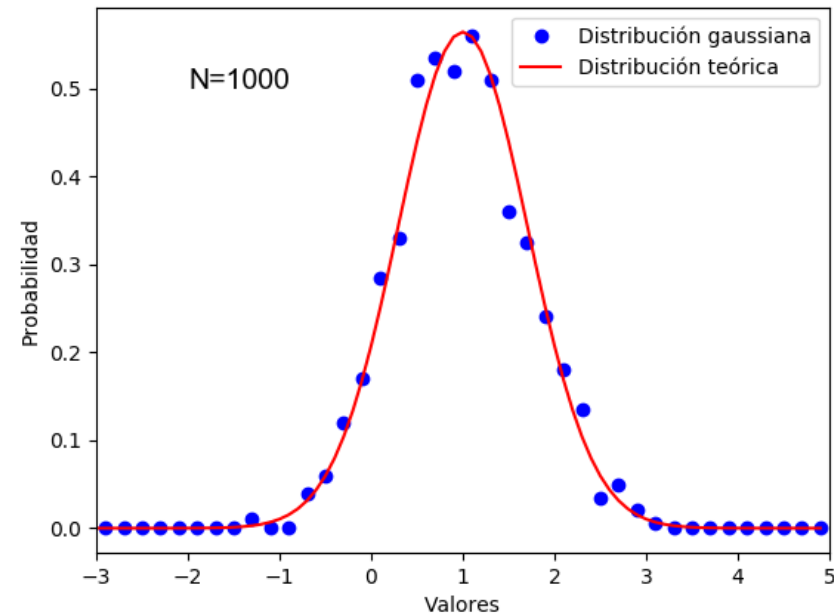
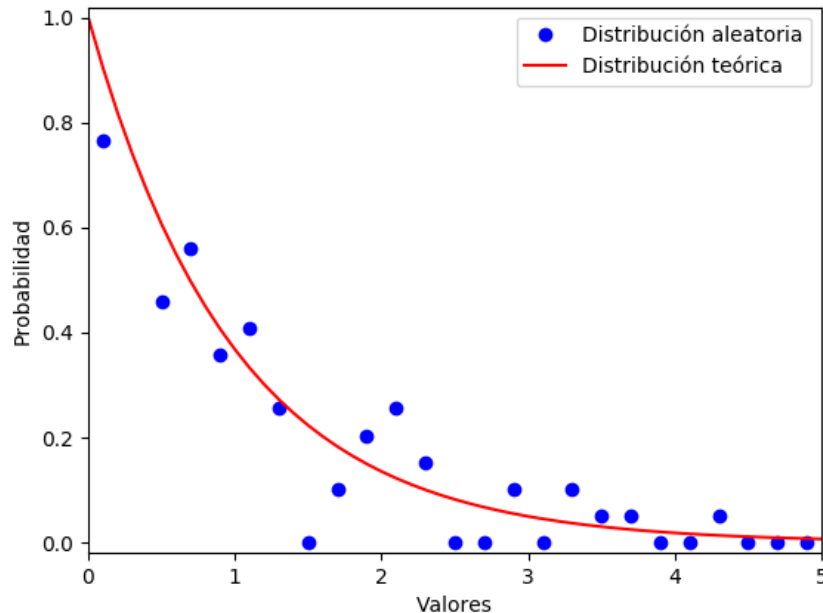


Números aleatorios

Generación de números aleatorios

Distribuciones de números aleatorios no uniformes

No siempre se busca generar una secuencia de números uniforme. Por ejemplo:



El Método de la Transformación consiste en relacionar la distribución que queremos usar con otra distribución conocida, como puede ser la uniforme.

El Método del Rechazo es más general y consiste en generar una secuencia uniforme y rechazar valores de acuerdo a la probabilidad que nos da la distribución que queremos producir.

Números aleatorios

Generación de números aleatorios

Método de la transformación

Se parte de una colección de valores, $\{x_1, x_2, \dots\}$, distribuidos según $P_x(x)$:

La probabilidad de encontrar un valor entre x y $x+dx$ es $P_x(x)dx$.

$$y=f(x)$$

Los valores de y , $\{y_1, y_2, \dots\}$, están distribuidos según $P_y(y)$.

La probabilidad de encontrar un valor entre y y $y+dy$ es $P_y(y)dy$.

Se cumple que:

$$P_x(x)dx = P_y(y)dy$$

Si los valores de x son uniformes:

$$P_x(x) = \text{constante} = c$$

$$P_y(y) = P_x(x) \left| \frac{dx}{dy} \right| = c \left| \frac{dx}{dy} \right|$$

Números aleatorios

Generación de números aleatorios

$$P_y(y) = P_x(x) \left| \frac{dx}{dy} \right| = c \left| \frac{dx}{dy} \right|$$

$$y=f(x): \quad \left| \frac{dx}{dy} \right| = \left| \frac{dy}{dx} \right|^{-1} = \left| \frac{df(x)}{dx} \right|^{-1} = \frac{1}{c} P_y$$

Ejemplo:

$$P_y(y) = \exp(-y)$$

$$\left| \frac{dx}{dy} \right| = e^{-y} \Rightarrow |dx| = |e^{-y} dy|$$

$$\left| \int dx \right| = |x| = \left| \int e^{-y} dy \right| = |-e^{-y}| \Rightarrow x = e^{-y}$$

$$f(x) = y = -\ln(x)$$

$$\{x_1, x_2, \dots\} \Rightarrow \{y_1, y_2, \dots\}, y_i = -\ln(x_i)$$

Números aleatorios

Generación de números aleatorios

Método de la transformación

Ejemplo: para reproducir una distribución exponencial a partir de una distribución uniforme se usa la transformación $y = -\ln(x)$

```
# Programa para generar una distribución exponencial a partir de una uniforme por el  
# método de la transformación
```

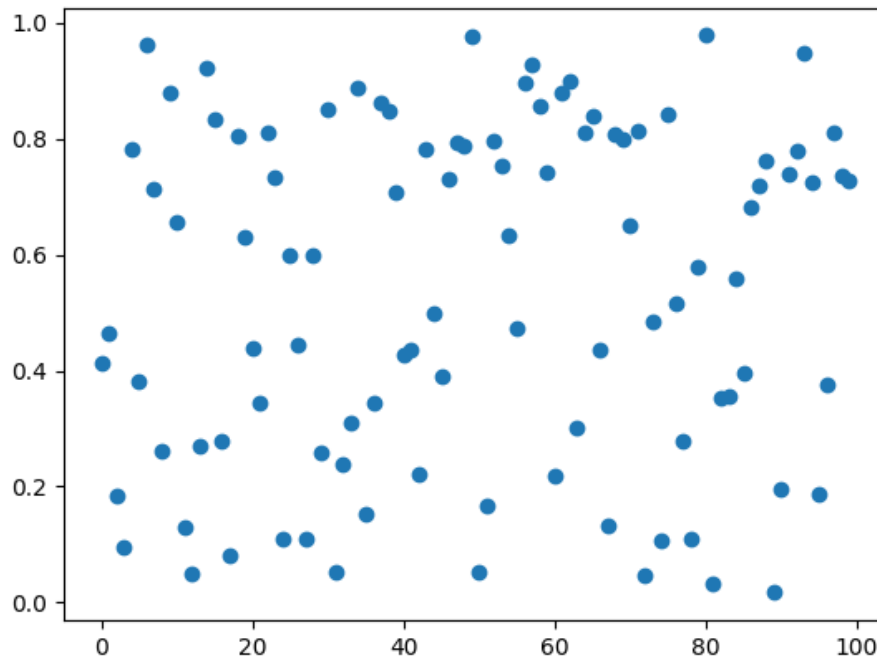
```
from random import *  
from matplotlib.pyplot import *  
from math import log  
import numpy as np  
  
n=100                                # Número de valores  
valores=[]                          # Lista para guardarlos  
valores_x=[]                        # Guardamos también los valores uniformes  
  
for i in range(n):  
    x=random()  
    y=-log(x)  
    valores_x.append(x)  
    valores.append(y)  
  
plot(valores_x,"o")  
show()  
plot(valores,"o")  
show()
```

Números aleatorios

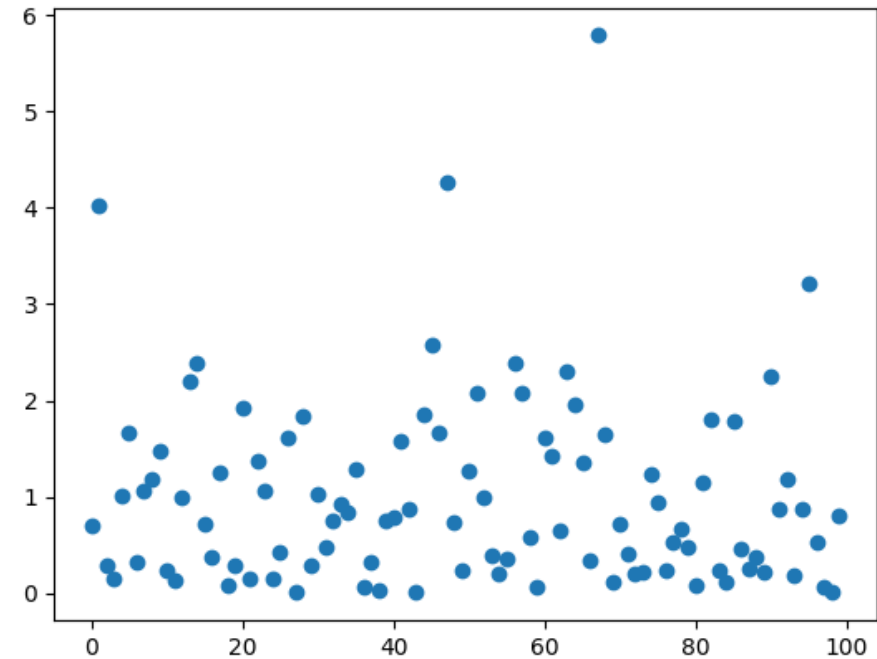
Generación de números aleatorios

Método de la transformación

Ejemplo: para reproducir una distribución exponencial a partir de una distribución uniforme se usa la transformación $y = -\ln(x)$



Uniforme



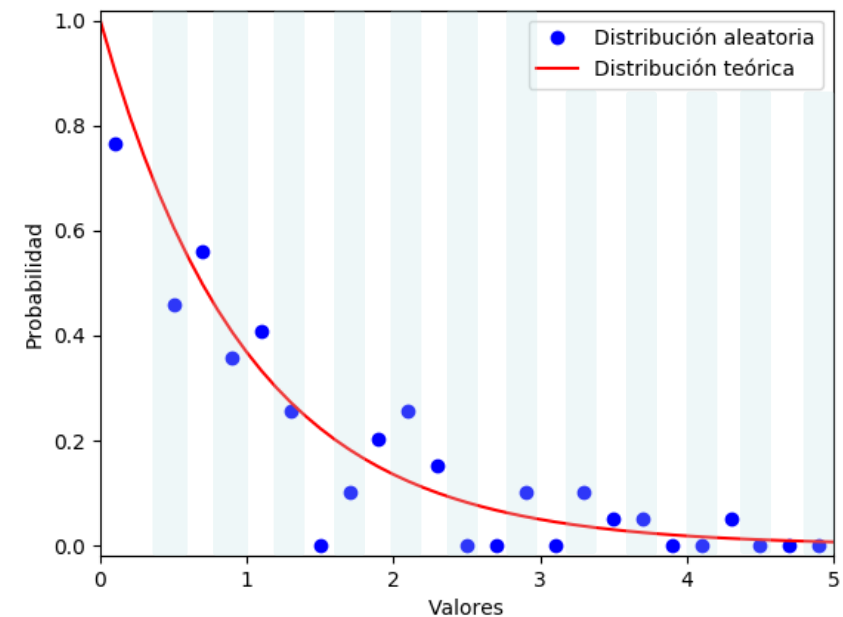
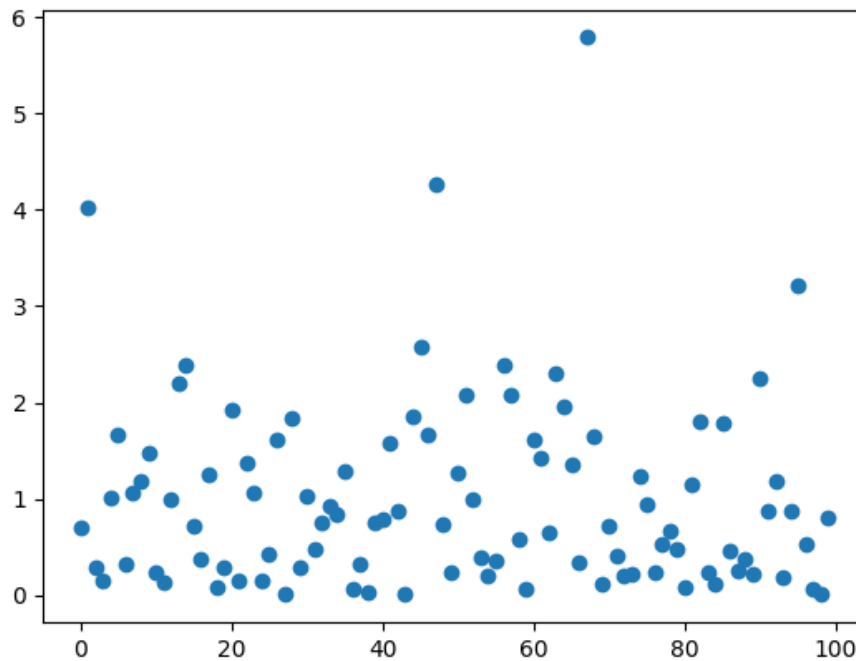
Exponencial

Números aleatorios

Generación de números aleatorios

Método de la transformación

Ejemplo: para reproducir una distribución exponencial a partir de una distribución uniforme se usa la transformación $y = -\ln(x)$



Números aleatorios

Generación de números aleatorios

Representación con histogramas en Python

```
num_clases=25                # Número de recipientes para el histograma
val_max=5.0                  # Máximo valor que vamos a considerar

# Usamos la función histogram de numpy. Devuelve un array con las cuentas de cada clase y
# otro con los límites entre clases. La dimensión de éste último es una unidad mayor que
# número de clases

cuentas,lims_clases=np.histogram(valores,bins=num_clases,range=(0,val_max),density=True)

# Creamos un array (clases) con valores centrados en cada clase

clases = lims_clases[:-1].copy()    # Copia de lims_clases con el último elemento borrado
clases = clases+(1/2)*val_max/num_clases    # Sumamos la mitad del tamaño de cada clase

# Creamos el gráfico:

plot(clases,cuentas,"ob", label="Distribución aleatoria")

# Creamos una curva con la distribución teórica y la añadimos al plot

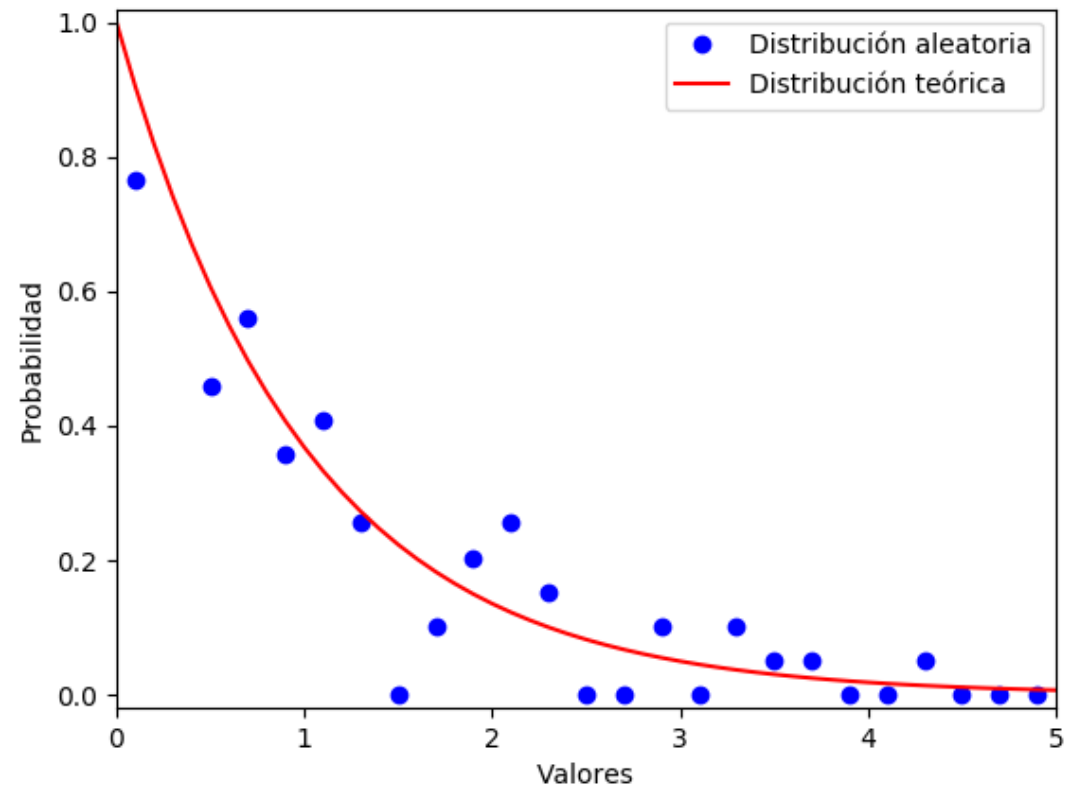
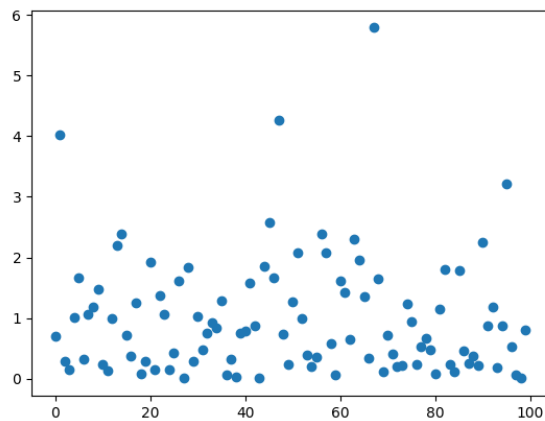
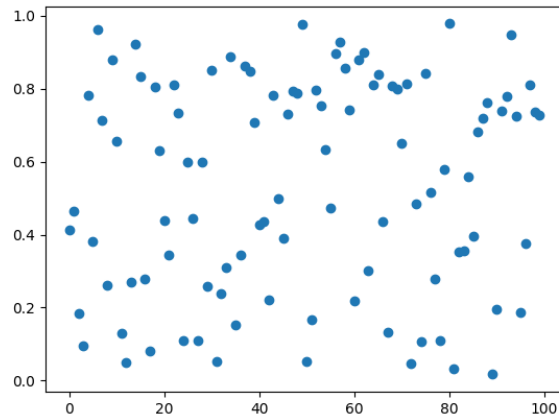
x_teo=np.arange(0,5.1,0.1)
y_teo=np.exp(-x_teo)
plot(x_teo,y_teo,"-r", label="Distribución teórica")

legend()    # Para incluir las leyendas que figuran en los "labels"
xlabel("Valores")
ylabel("Probabilidad")
xlim(0,5)
ylim(-0.02,1.02)
show()
```

Números aleatorios

Generación de números aleatorios

Representación con histogramas en Python



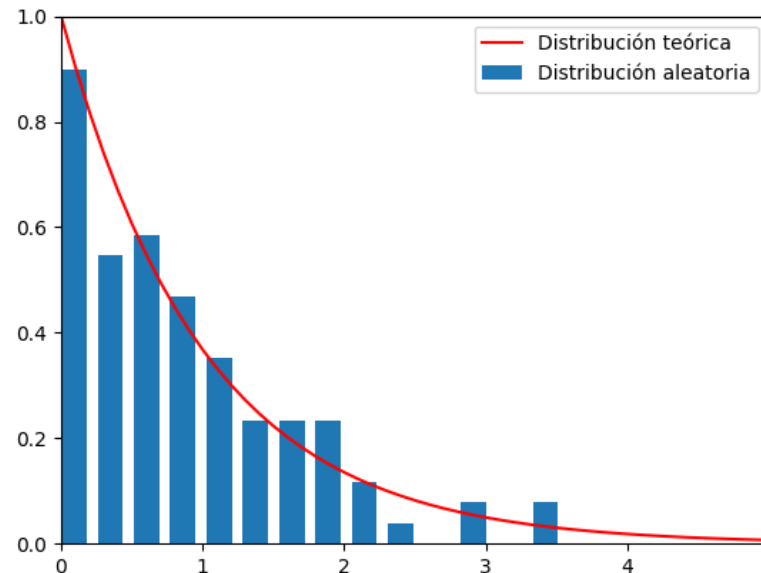
Números aleatorios

Generación de números aleatorios

Representación con histogramas en Python

También podemos usar la función "hist" incluida en el paquete matplotlib.pyplot, aunque de esta manera se obtiene un gráfico de barras.

```
num_clases=25                                # Número de recipientes para el histograma
val_max=5.0                                   # Máximo valor que vamos a considerar
hist(valores,num_clases,density=True,width=0.9*val_max/num_clases, label="Distribución
aleatoria")
plot(x_teo,y_teo,"-r", label="Distribución teórica")
legend()
xlim(0,5)
ylim(0,1)
show()
```

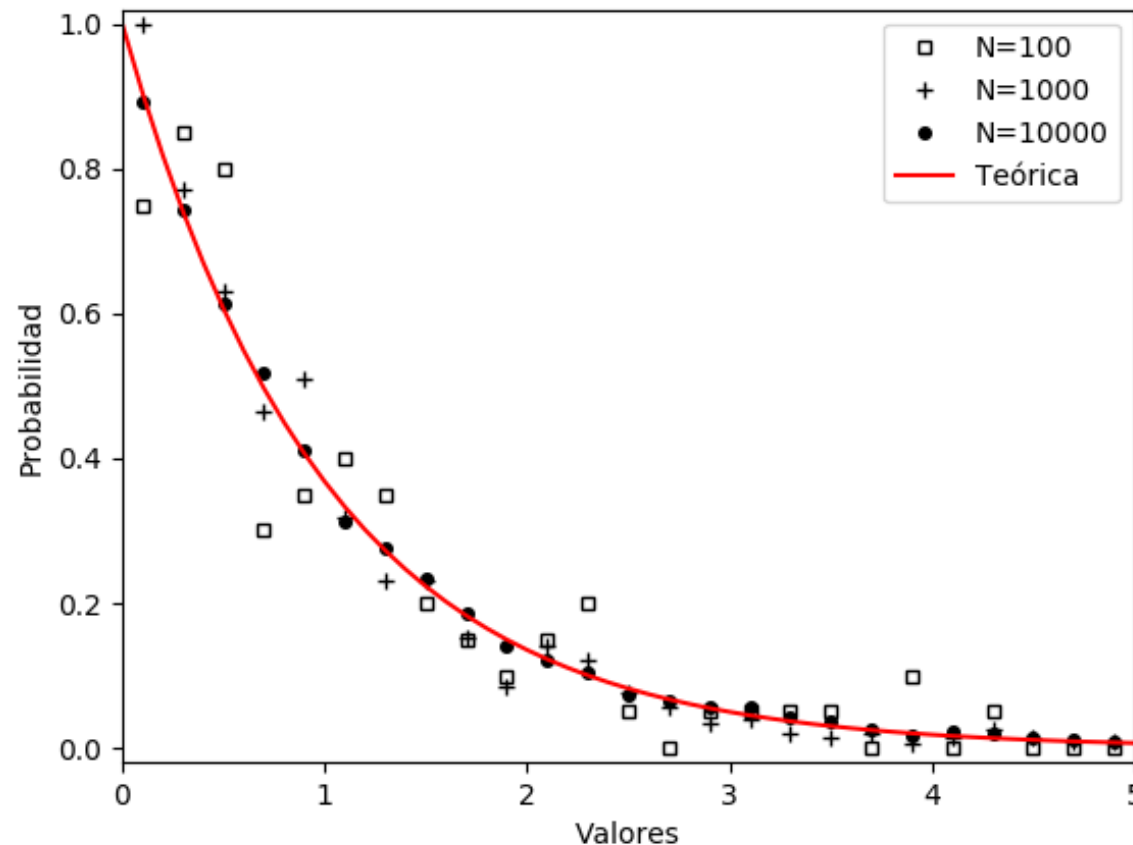


Números aleatorios

Generación de números aleatorios

Método de la transformación

Ejemplo: para reproducir una distribución exponencial a partir de una distribución uniforme se usa la transformación $y = -\ln(x)$

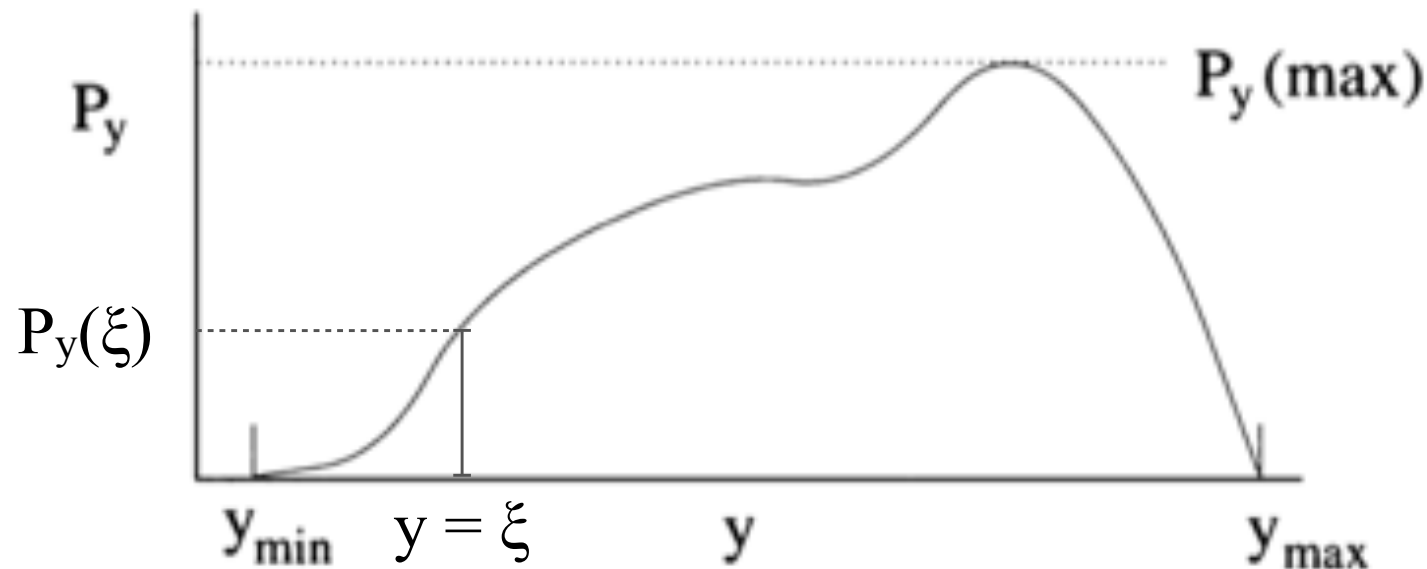


Números aleatorios

Generación de números aleatorios

Método del rechazo

Se genera una secuencia de números uniforme: $\{y_1, y_2, \dots\}$. Para cada y_i se obtiene un valor aleatorio, "test", entre 0 y $P_y(\max)$. Si $(\text{test} < P(y_i))$ el valor se acepta, y si $(\text{test} > P(y_i))$, el valor se rechaza. Se va creando una lista o array sólo con los valores aceptados.



Números aleatorios

Generación de números aleatorios

Para algunas funciones comunes, Python ofrece la posibilidad de generar número aleatorios según las correspondientes distribuciones. En los ejercicios anteriores serían `expovariate(1)` y `gauss(1,1)`.