



```
def rng(m=2**32, a=1103515245, c=12345):
    rng.current = (a * rng.current + c) % m
    return rng.current / m

# setting the seed
rng.current = 1
```

Выведем несколько первых элементов последовательности:

```
[rng() for i in range(10)]
```




```
[0.25693503906950355,
 0.5878706516232342,
 0.15432575810700655,
 0.767266943352297,
 0.9738139626570046,
 0.5858681506942958,
 0.8511155843734741,
 0.6132153405342251,
 0.7473867232911289,
 0.06236015981994569]
```

Выбор параметров m , a и c существенно влияет на качество последовательности. Если привести к неожиданным последствиям:

```
def rng(m=97, a=5, c=0):
    rng.current = (a * rng.current + c) % m
    return rng.current / m

rng.current = 7

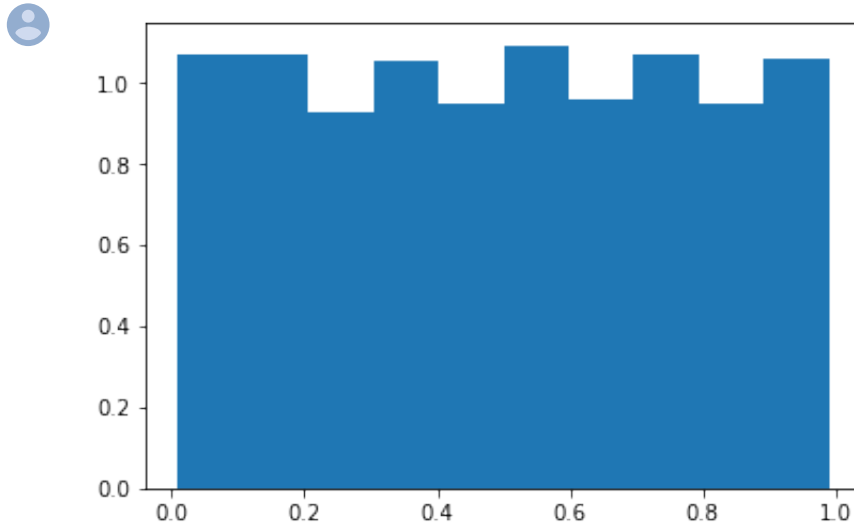
random = [rng() for i in range(1000)]
print("This sequence looks as random:")
random[:10]
```



```
This sequence looks as random:
[0.36082474226804123,
 0.8041237113402062,
 0.020618556701030927,
 0.10309278350515463,
 0.5154639175257731,
 0.5773195876288659,
 0.8865979381443299,
 0.4329896907216495,
 0.16494845360824742,
 0.8247422680412371]
```

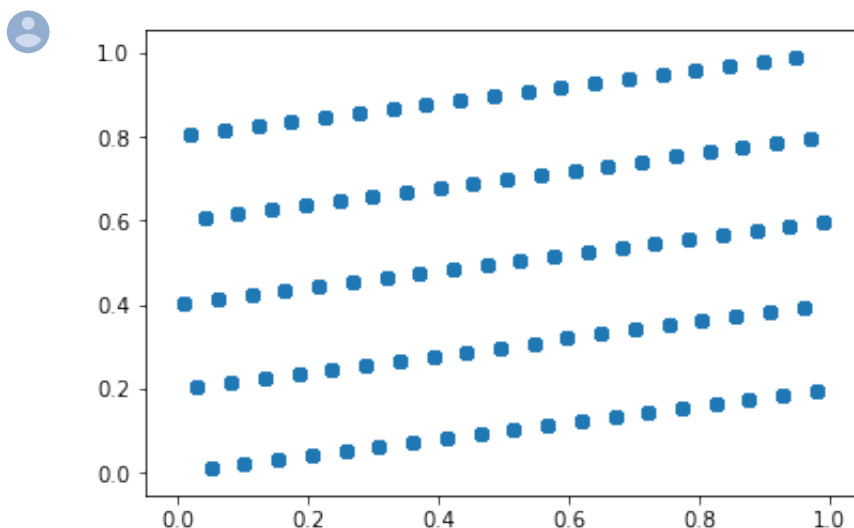
Гистограмма распределения похожа на равномерное:

```
import matplotlib.pyplot as plt
plt.hist(random, normed=True)
plt.show()
```



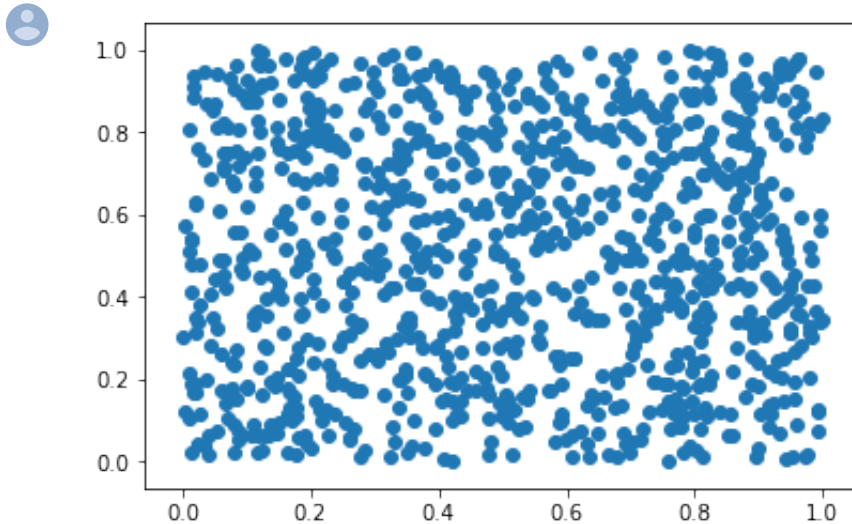
Однако, спектральный тест показывает, что точки располагаются на гиперплоскостях, что свидетельствует о недостаточной случайности:

```
plt.scatter(random[1:], random[:-1])
plt.show()
```



Более аккуратный выбор параметров приводит к более "случайному" распределению:

```
def rng(m=2**32, a=1103515245, c=12345):  
    rng.current = (a * rng.current + c) % m  
    return rng.current / m  
rng.current = 1  
  
random = [rng() for i in range(1000)]  
plt.scatter(random[1:], random[:-1])  
plt.show()
```



Существует набор тестов для проверки "случайности". Например, тесты [Diehard tests](#).

Больше методов генерации собрано [здесь](#).

▼ Генерация выборки из заданного распределения

Допустим, у нас есть генератор случайных чисел из отрезка [0, 1]. Как получить выборку

▼ Задача

Смоделировать выборку объема 1000 из дискретного распределения на множестве цифр 0.31, 0.54, 0.111, 0.02, 0.001, 0.2. По выборке построить гистограмму. Оптимизируйте алгоритм генерации выборки с неупорядоченными и упорядоченными весами.

```
# Solution here
```

▼ Inverse transform method

В следующем предложении заключается идея метода *inverse transform*:

Если ξ имеет равномерное распределение в $[0, 1]$, тогда $F^{-1}(\xi)$ распределена по закон

▼ Задача

Смоделируйте выборку размера 1000 из распределения $Exp(\lambda)$. Постройте выборочную распределения.

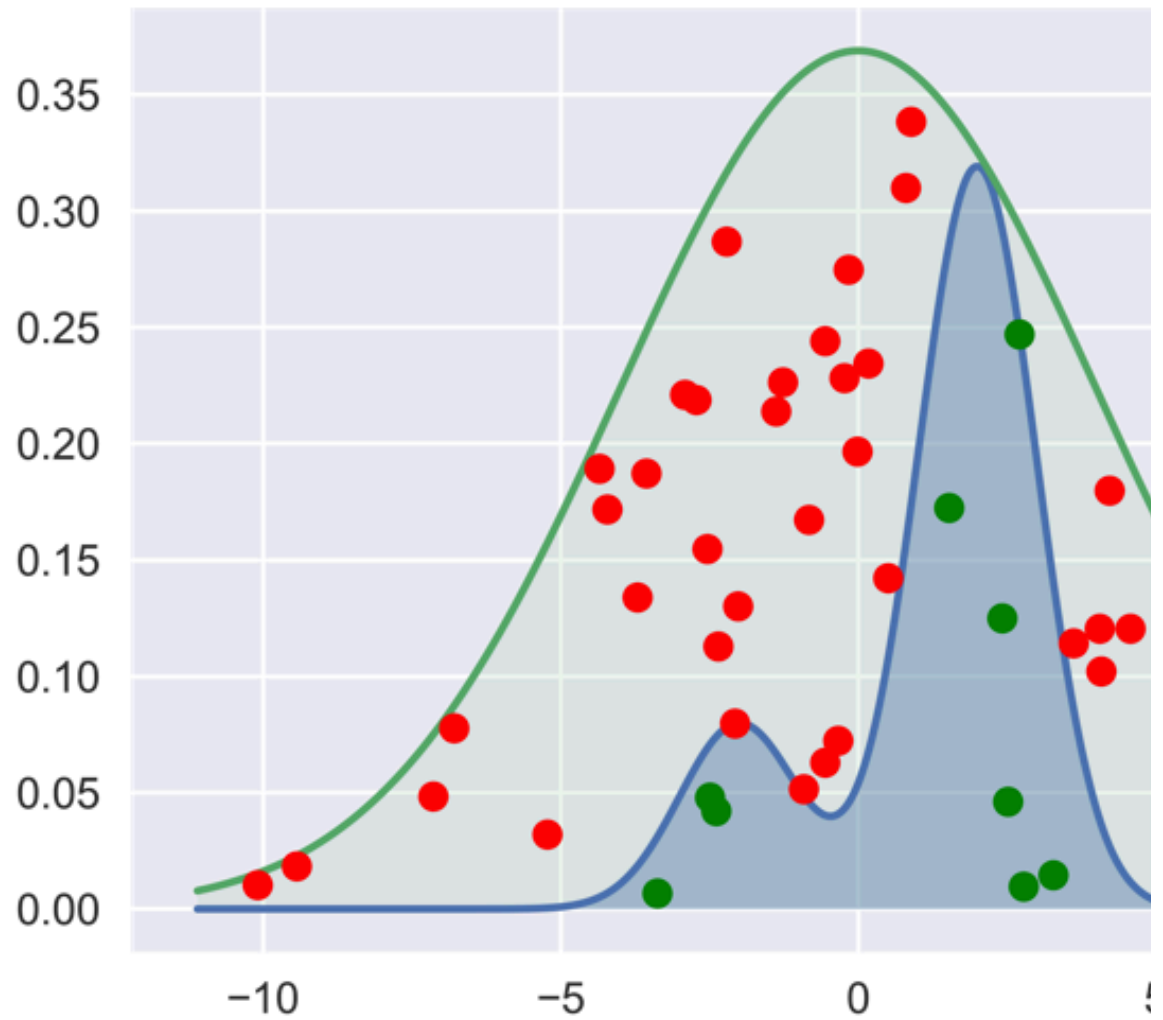
```
# Solution here
```

Rejection sampling (Accept-reject method)

Идея метода: сэмпить из распределения, из которого умеем, а затем отбирать точки, кс
Картинка иллюстрирует идею метода:

REJECTION SAMPLING

.....



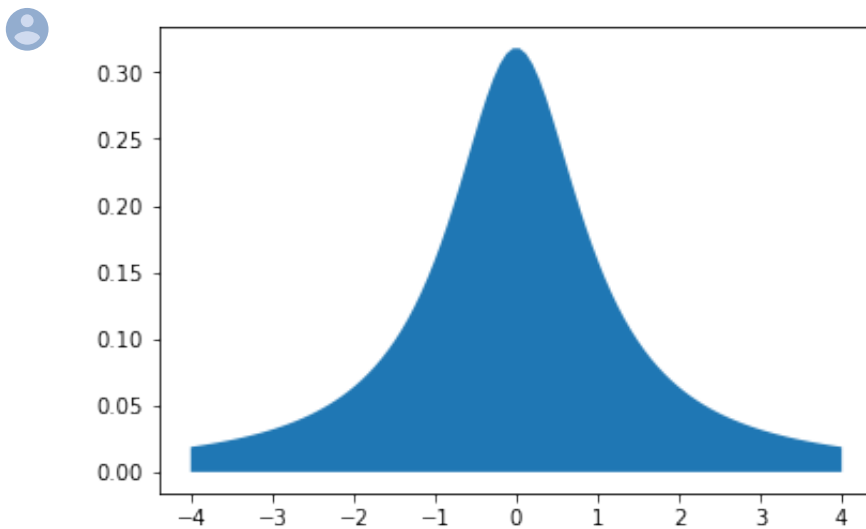
p, known

T

▼ Задача

Обоснуйте, что метод ассерт-режект действительно производит выборку из нужного распределенного распределения Коши, приведенного ниже, используя генератор равномерного распределения. Сравните полученную выборочную гистограмму с графиком точной функции плотности.

```
from scipy import stats
import numpy as np
dist = stats.cauchy()
x = np.linspace(-4, 4, 100)
plt.fill_between(x, 0, dist.pdf(x)) #needs to be normalized!
plt.show()
```



Solution here

▼ Coordinate transformation method

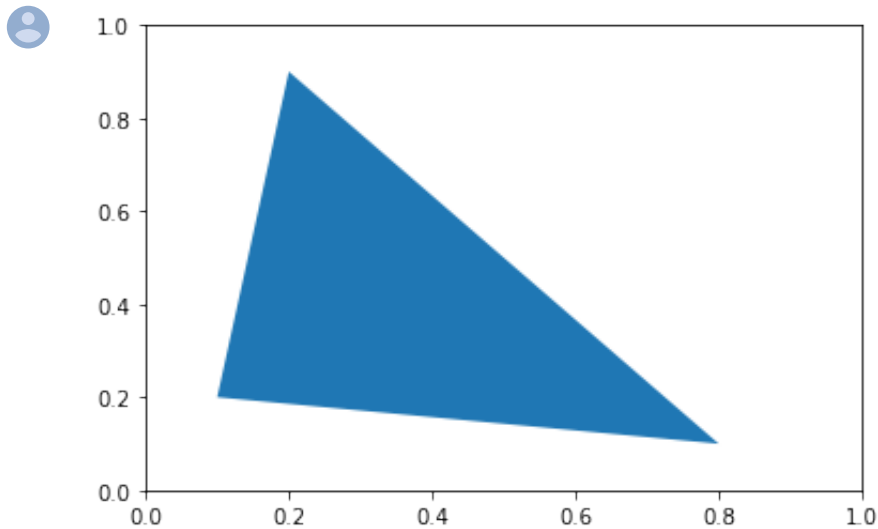
Метод ассерт-режект в ряде случаев может оказываться неэффективным и требовать попробовать найти преобразование координат, которое переводит простую область (из единичный квадрат) в требуемую, но при этом сохраняет соотношение площадей.

▼ Задача

Смоделировать выборку из 500 точек равномерно распределенных внутри данного треугольника.

```
import matplotlib
from matplotlib.patches import Polygon
from matplotlib.collections import PatchCollection

polygon = Polygon(0.1 * np.array([[1, 2], [2, 9], [8, 1]]), True)
plt.gca().add_collection(PatchCollection([polygon]))
plt.show()
```

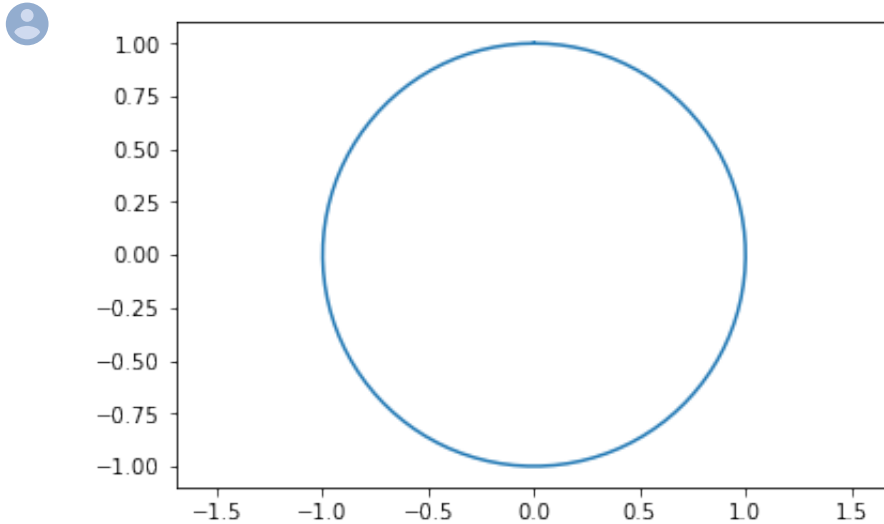


Solution here

▼ Задача

Смоделировать выборку из 500 точек внутри данного круга без использования метода о


```
from matplotlib.patches import Circle
t = np.linspace(0, 2 * np.pi, 100)
plt.plot(np.sin(t), np.cos(t))
plt.axis('equal')
plt.show()
```



```
# Solution here
```

▼ Задача

Напишите функцию, которая моделирует случайное симметричное блуждание на двумерной точке (0, 0). Приведите графики выборочных траекторий для $n=100$.

```
# Solution here
```

▼ Random normal generator

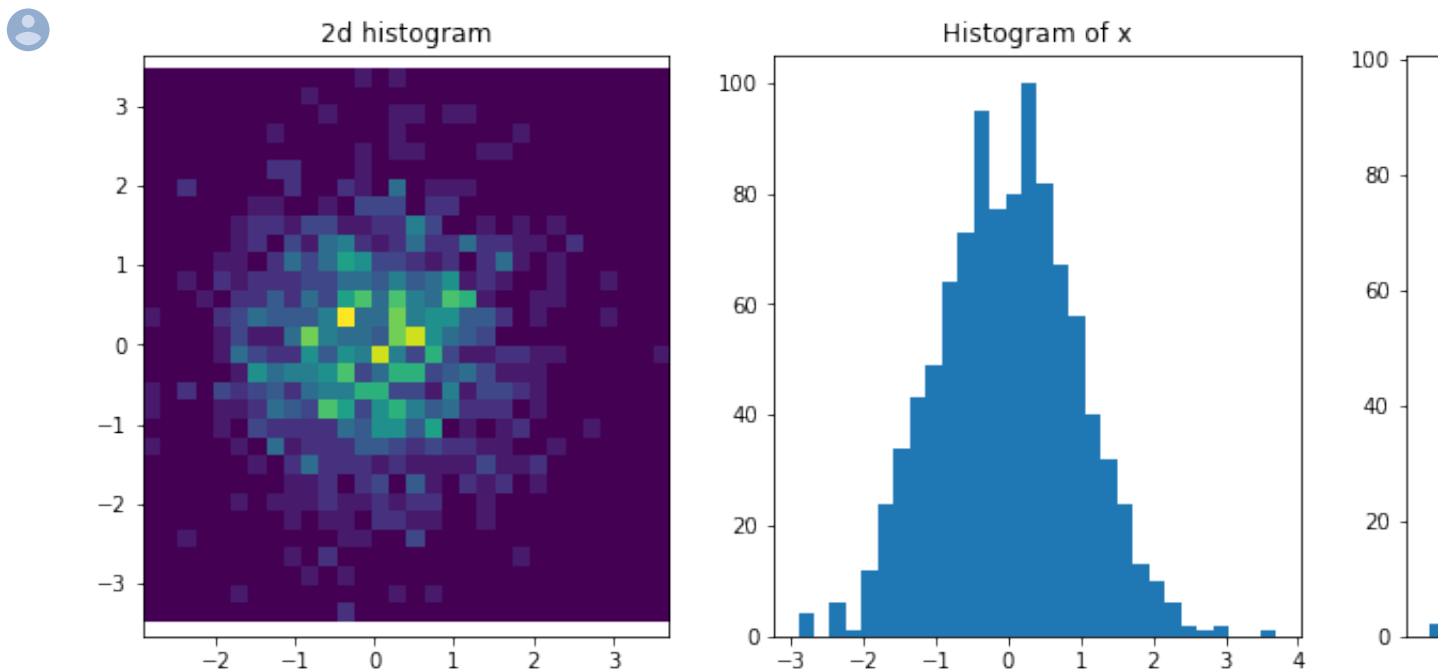
▼ Задача

Докажите, что приведенный ниже алгоритм (Box-Muller algorithm) формирует выборку из Модифицируйте метод, чтобы исключить вызовы тригонометрических функций `np.sin` метода смоделируйте выборку объема 1000 из двумерного гауссовского распределения матрицей $\begin{pmatrix} 2 & 1 \\ 1 & 4 \end{pmatrix}$. Постройте 2D гистограмму полученного распределения.

```

n = 1000
u1, u2 = np.random.rand(2, n)
r = np.sqrt(-2 * np.log(u1))
theta = 2 * np.pi * u2
x = r * np.cos(theta)
y = r * np.sin(theta)
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].hist2d(x, y, bins=30)
ax[0].axis('equal')
ax[1].hist(x, bins=30)
ax[2].hist(y, bins=30)
ax[0].set_title("2d histogram")
ax[1].set_title("Histogram of x")
ax[2].set_title("Histogram of y")
plt.show()

```



Solution here

▼ Практическое задание

Реализовать метод генерации случайного разбиения n -элементного множества на подмножества ожидаемое число подмножеств в случайном разбиении множества из 100 элементов.

Подсказка 1: Ширяев, Вероятность, т1, задача 2 к параграфу 1.

Подсказка 2: <http://djalil.chafai.net/blog/2012/05/03/generating-uniform-random-partitions/>

```
# Solution here
```