

## random числа

Вот случайное число: 4242. Если хочешь другое, напиши снова!

По какому алгоритму формируются квантили после работы генератора случайных чисел ?

### Ответ

Алгоритм формирования квантилей после работы генератора случайных чисел зависит от конкретной реализации и метода оценки квантилей. Рассмотрим общий подход на примере равномерного распределения.

### Шаги алгоритма

1. Генерация случайных чиселГенератор случайных чисел создает последовательность значений, равномерно распределенных на интервале  $[0,1)$ .
2. Сортировка выборкиПолученную выборку сортируют по возрастанию. Это позволяет упорядочить значения таким образом, чтобы легко вычислять квантили.
3. Вычисление позиций квантилейДля нахождения квантиля порядка  $\alpha$ , используют формулу расчета позиции элемента в отсортированной последовательности:

$$p = (\alpha \times n) \quad p = (\alpha \times n)$$

где:

- $n$  — размер выборки,
- $\alpha$  — уровень квантили (например, для медианы  $\alpha=0.5$ , для верхнего квартиля  $\alpha=0.75$ ).

4. Интерполяция между элементамиЕсли позиция  $p$  является целым числом, то соответствующий элемент выборки и есть искомым квантилем. Однако чаще всего  $p$  оказывается дробным значением. Тогда используется линейная интерполяция между двумя ближайшими соседними элементами:

$$Q(\alpha) = x_{[p]} + (p - [p])(x_{[p]+1} - x_{[p]}) \quad Q(\alpha) = x_{[p]} + (p - [p])(x_{[p]+1} - x_{[p]})$$

где:

- $x_i$  — элементы отсортированного массива,
- $[p]$  — целая часть  $p$ ,
- $[p]+1$  — наименьшее целое больше  $p$ .

### Пример

Допустим, имеем следующую выборку размером  $n=8$ :

$X = \{0.12, 0.35, 0.48, 0.61, 0.73, 0.84, 0.92, 0.98\}$

Хотим найти верхний квартиль ( $\alpha=0.75$ ):

1. Размер выборки  $n=8$ .
2. Вычисляем позицию квантили:

$$p = 0.75 \times 8 = 6$$

3. Так как  $p=6$  — целое число, значение верхней квартили совпадает с шестым элементом выборки:

$$Q_{0.75}(X) = X_6 = 0.84$$

Таким образом, квантиль формируется путем сортировки выборки и последующей интерполяции или выбора подходящего элемента в зависимости от расчетной позиции.

Как можно сгенерировать массив случайных чисел? Какие методы существуют для генерации случайных чисел? Можно ли проверить случайность сгенерированных чисел?

Ответ

## Генерация массива случайных чисел

Для генерации массива случайных чисел существует несколько подходов и методов, зависящих от требований к качеству случайности, производительности и криптографической стойкости.

## Методы генерации случайных чисел

Существует два основных типа генераторов случайных чисел:

### 1. Датчики псевдослучайных чисел (PRNG):

Эти датчики генерируют последовательность чисел, кажущихся случайными, но фактически определяемых начальным состоянием ("зерном"). К наиболее популярным алгоритмам относятся:

- **Linear Congruential Generator (LCG):** Простой метод, использующий рекурсивную формулу вида:

$$X_{n+1} = (aX_n + c) \bmod m$$

Этот метод прост в реализации, но имеет ограниченный период повторяемости и невысокое качество случайности.

- **Mersenne Twister:** Один из самых популярных PRNG, известный своей скоростью и качеством. Алгоритм основан на большом периоде  $(2^{19937} - 1)$  и обеспечивает хорошие статистические свойства.
- **XORShift:** Очень быстрый генератор, основанный на операциях XOR и сдвигах битовых строк. Обладает неплохими свойствами случайности и подходит для высокопроизводительных приложений.
- **CryptoRandom:** Специальные классы генераторов, используемые в криптографии, такие как RC4, AES в режиме CTR и др., обеспечивают высокую степень непредсказуемости.

### 2. Источники истинно случайных чисел (RNG):

Используют физические процессы для генерации действительно случайных последовательностей. Примеры включают:

- **Использование шума электроники (термодинамический шум, радиационный фон);**
- **Время реакции пользователей (для веб-приложений);**
- **Специализированные аппаратные модули (HWRNG).**

## **Проверка случайности сгенерированных чисел**

**Проверка качества случайности осуществляется различными методами и тестовыми пакетами. Вот некоторые из них:**

### **Статистические тесты:**

- **Chi-Square Test:** проверяет распределение частот появления каждого числа в выборке.
- **Kolmogorov-Smirnov Test:** сравнивает эмпирическое распределение с теоретическим.
- **Serial Correlation Test:** определяет наличие корреляций между числами в последовательности.
- **Runs Test:** анализирует количество серий подряд идущих одинаковых знаков.

### **Тестовые пакеты:**

- **Diehard Tests:** коллекция тестов, разработанная Джорджем Марсалья, включающая около 15 различных проверок.
- **NIST Statistical Test Suite:** комплекс тестов, рекомендованный Национальным институтом стандартов и технологий США.
- **TestU01:** библиотека для тестирования генераторов случайных чисел, содержащая большое количество тестов разной сложности.

## **Заключение**

**Выбор метода генерации случайных чисел зависит от требуемого уровня случайности, скорости и области применения. Для большинства повседневных задач достаточно качественных PRNG, таких как Mersenne Twister, однако для критически важных областей (криптография, научные исследования) рекомендуется использование специализированных датчиков или даже аппаратных устройств. Когда мы говорим набор случайных чисел мы подразумеваем Равномерное распределение ?**

### **Ответ**

**Нет, понятие набора случайных чисел не обязательно подразумевает именно равномерное распределение. Случайные числа могут подчиняться различным вероятностным законам, среди которых равномерное распределение лишь одно из возможных вариантов.**

## Типичные виды распределений случайных величин:

1. **Равномерное распределение (Uniform Distribution):** Каждое число из заданного диапазона имеет равную вероятность появления. Например, бросок игральной кости даёт равные шансы выпадения каждой грани  $(1/6)$ . Формула плотности вероятности выглядит следующим образом:

$$f(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & \text{иначе} \end{cases}$$

Здесь  $a$  и  $b$  — границы интервала.

2. **Нормальное распределение (Gaussian or Normal Distribution):** Характеризуется колоколообразной формой кривой плотности вероятности. Оно широко применяется в естественных науках, экономике и статистике благодаря центральной предельной теореме. Нормальное распределение задаётся параметрами среднего ( $\mu$ ) и стандартного отклонения ( $\sigma$ ), плотность вероятности определяется формулой:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

3. **Экспоненциальное распределение (Exponential Distribution):** Часто используется для моделирования временных интервалов между событиями, такими как моменты прихода клиентов или отказов оборудования. Его плотность вероятности задаётся параметром интенсивности ( $\lambda$ ):

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

4. **Биномиальное распределение (Binomial Distribution):** Применяется, когда рассматривается фиксированное число испытаний Бернулли (два исхода). Вероятность успеха обозначается  $p$ , общее число испытаний —  $n$ . Плотность вероятности задаётся формулой:

$$P(X=k) = \binom{n}{k} p^k (1-p)^{n-k}$$

5. **Распределение Пуассона (Poisson Distribution):** Используется для описания количества редких событий за единицу времени или пространства. Параметр  $\lambda$  характеризует среднее число событий за этот промежуток. Распределение выражается формулой:

$$P(k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

## Когда применяют равномерное распределение?

Равномерное распределение часто используется в качестве базового инструмента при создании случайных последовательностей, особенно в вычислительной технике. Многие библиотеки программирования содержат встроенные функции для генерации равномерно распределённых случайных чисел, поскольку это простой и удобный способ задания начальной точки для преобразования в другие типы распределений.

Однако важно понимать, что само по себе "случайное число" не связано исключительно с равномерностью. Выбор конкретного закона распределения диктуется прикладной задачей и характером изучаемого явления.

В C# такой код сгенерирует нам равномерное распределение?

```
Random random = new Random();
double[] data = new double[numOfValues];
for (int i = 0; i < numOfValues; i++) {
    data[i] = random.NextDouble() * 100; // числа от 0 до 100
}
```

### Ответ

Да, приведённый вами код на C# сгенерирует массив случайных чисел с равномерным распределением.

Давайте разберём пошагово:

```
Random random = new Random(); // Создаем экземпляр класса Random
double[] data = new double[numOfValues]; // Массив для хранения результатов
for (int i = 0; i < numOfValues; i++) {
    data[i] = random.NextDouble() * 100; // Заполняем массив случайными числами
}
```

### Объяснение:

- `random.NextDouble()` возвращает случайное вещественное число в диапазоне  $[0, 1)$  (включительно ноль, исключая единицу).
- Умножение результата на 100 масштабирует диапазон случайных чисел до  $[0, 100)$ .

Таким образом, каждый элемент массива `data` будет содержать случайное число, равномерно распределённое в пределах от 0 включительно до 100 не включительно. Это соответствует равномерному распределению, так как каждое число в указанном диапазоне имеет равную вероятность появления.

А как нам сгенерить в C# нормальное распределение ?

### Ответ

Чтобы сгенерировать случайные числа с нормальным распределением в C#, удобно воспользоваться методом Бокса-Мюллера или преобразованием обратных функций (метод обратной функции). Стандартная библиотека .NET не включает прямой поддержки нормального распределения, поэтому вам потребуется реализовать собственную реализацию либо использовать стороннюю библиотеку вроде Math.NET Numerics.

Рассмотрим оба подхода подробнее:

### Метод Бокса-Мюллера

Метод Бокса-Мюллера преобразует две независимые случайные величины с равномерным распределением в две случайные величины с нормальным распределением.

Формулы метода выглядят так:

$$Z1 = -2 \ln U1 \cos(2\pi U2) \quad Z1 = -2 \ln U1 \cos(2\pi U2)$$

$$Z2 = -2 \ln U1 \sin(2\pi U2) \quad Z2 = -2 \ln U1 \sin(2\pi U2)$$

Где:

- $U1$  и  $U2$  — случайные числа с равномерным распределением в промежутке  $[0, 1]$ ;
- $Z1$  и  $Z2$  — полученные нормально распределённые случайные числа с нулевым средним и дисперсией равной 1.

Реализация на C#:

```
using System;
```

```

class Program
{
    static void Main()
    {
        int numValues = 1000;
        double mean = 50;    // Среднее значение
        double stdDev = 10;  // Стандартное отклонение
        Random rand = new Random();

        double[] normalData = new double[numValues];

        for (int i = 0; i < numValues / 2; ++i)
        {
            double u1 = rand.NextDouble();
            double u2 = rand.NextDouble();

            double z1 = Math.Sqrt(-2 * Math.Log(u1)) * Math.Cos(2 * Math.PI * u2); // Z1
            double z2 = Math.Sqrt(-2 * Math.Log(u1)) * Math.Sin(2 * Math.PI * u2); // Z2

            normalData[2*i] = mean + stdDev * z1;
            if ((2*i + 1) < numValues)
                normalData[2*i + 1] = mean + stdDev * z2;
        }

        foreach (double value in normalData)
        {
            Console.WriteLine(value);
        }
    }
}

```

#### Описание:

- Генерируем пары равномерно распределённых случайных чисел (rand.NextDouble()).
- Используем формулы Бокса-Мюллера для получения двух новых случайных чисел с нормальным распределением.
- Масштабируем получившиеся значения путём добавления среднего значения и умножения на стандартное отклонение, чтобы задать нужное среднее и дисперсию.

#### Библиотека Math.NET Numerics

Альтернативный вариант — использование готовой библиотеки, такой как Math.NET Numerics, которая предоставляет готовые инструменты для генерации нормально распределённых случайных чисел.

Пример с использованием Math.NET Numerics:

```
using System;
using MathNet.Numerics.Distributions;

class Program
{
    static void Main()
    {
        int numValues = 1000;
        double mean = 50;    // Среднее значение
        double stdDev = 10;  // Стандартное отклонение
        var rng = new System.Random();

        var distribution = new Normal(mean, stdDev);
        double[] normalData = new double[numValues];

        for(int i = 0; i < numValues; i++)
        {
            normalData[i] = distribution.Sample(rng);
        }

        foreach(double value in normalData)
        {
            Console.WriteLine(value);
        }
    }
}
```

Описание:

- Устанавливаете NuGet-пакет MathNet.Numerics, который добавляет готовую поддержку различных типов распределений.
- Используя класс Normal, создаёте объект, передавая среднее значение и стандартное отклонение.
- Затем с помощью метода .Sample() извлекаете случайные числа с указанным нормальным распределением.

Итог:

Методом Бокса-Мюллера вы можете самостоятельно создать нормальный генератор случайных чисел в C#, а используя библиотеку Math.NET Numerics, вы получите готовый инструмент, позволяющий быстро решить вашу задачу.