НЕКОТОРЫЕ МЕТОДЫ УСКОРЕНИЯ ГЕНЕРАЦИИ ЭКСПОНЕНЦИАЛЬНО РАСПРЕДЕЛЕННЫХ НЕПРЕРЫВНЫХ СЛУЧАЙНЫХ ВЕЛИЧИН

SOME METHODS OF ACCELERATING THE GENERATION OF EXPONENTIALLY DISTRIBUTED CONTINUOUS RANDOM VALUES

УДК 004

Саватюгина Юлия Борисовна, Студент кафедры ИУ6 МГТУ им Н.Э.Баумана

Гаипов Мурад Магомедшапиевич, Студент кафедры ИУ5 МГТУ им Н.Э.Баумана

Savatyugina Yulia Borisovna, Student of the IU6 department of the Bauman Moscow State Technical University, yaayulya@yandex.ru

Gaipov Murad Magomedshapievich, Student of the IU5 Department of MSTU named after N.E. Bauman, m.gaipov@mail.ru

Аннотация: Задача генерации случайных величин стала очень важной с развитием вычислительной техники. Такие генераторы случайных чисел главным образом применяются при проведении статистических испытаний, в криптографии, имитационном моделировании, эконометрике, теория информации и т.д. Большинство существующих генераторов имеют псевдослучайную природу и плохо годятся в решении серьезных задач. В данной работе мы разрабатываем библиотеку для генерации случайных величин распределенных по разным законам.

Summary: The problem of generating random variables has become very important with the development of computer technology. Such random number

generators are mainly used in statistical testing, cryptography, simulation, econometrics, information theory, etc. Most of the existing generators are of a pseudo-random nature and are poorly suited for solving serious problems. In this work, we are developing a library for generating random variables distributed according to different laws.

Ключевые слова: генерация случайных величин, нормальное распределение, экспоненциальное распределение, RDRAND, Intel, Bull Mountain, Ziggurat, Box-Muller

Keywords: random variable generation, normal distribution, exponential distribution, RDRAND, Intel, Bull Mountain, Ziggurat, Box-Muller

Экспоненциальное распределение

Экспоненциальное распределение – абсолютно непрерывное распределение модулирующее время между двумя последовательными совершениями одного и того же события. Плотность экспоненциального распределения имеет вид:

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0, \\ 0, & x < 0. \end{cases}$$

А функцию распределения можно получить интегрированием плотности:

$$F_X(x) = \begin{cases} 1 - e^{-\lambda x} & , x \geq 0, \\ 0 & , x < 0. \end{cases}$$

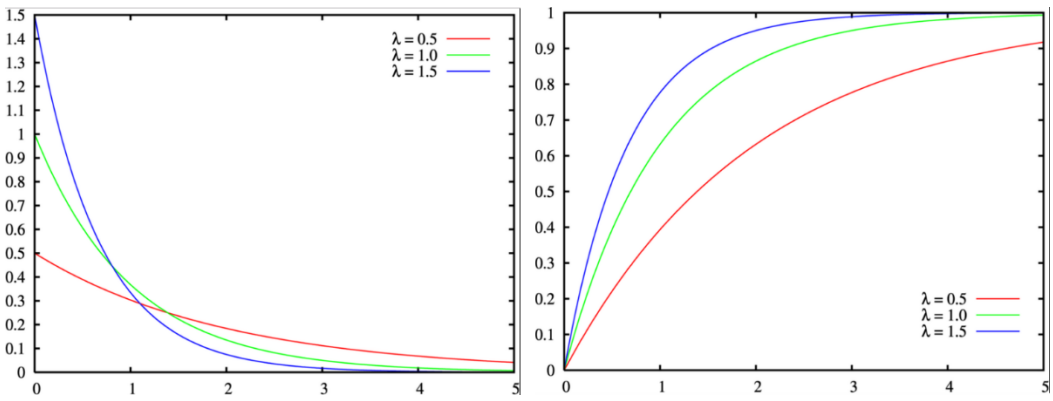


Рис.5 Графики плотности вероятности и функции распределения

Для ее реализации сначала рассмотрим метод обратного преобразования (преобразование Смирнова). Суть в том что мы берем генератор равномерно распределенных случайных величин на [0;1] который дает выборку U_1 ,…, U_n ~ $U_{[0;1]}$. Вычисляем функцию обратную той, которую нам необходимо получить. Вычисляем $X_i = F^{-1}(U_i)$ и получаем выборку X_1 ,…, X_n нужного нам распределения. Доказательство занимает одну строчку:

$$\mathbb{P}(F^{-1}(U) \leq x) = \mathbb{P}(U \leq F(x)) = F(x)$$

Обратная функция к F(X) выглядит следующим образом:

$$F^{-1}(x) = -\frac{1}{\lambda} \ln(1 - x), \text{ где } x \sim U_{[0;1]}$$

ln(1-x) можно заменить на ln(x), от этого ничего не изменится так как равномерное распределение нам нужно для генерации значения квантили нужного нам случайного распределения.

Функция для генерации случайной величины распределенной по экспоненциальному закону методом обратного преобразования:

```
double ExponentialDistSmirnov(double lambda)
return (-1/lambda)*log(UniformDistribution(0,1));
```

Так же очень быстрым алгоритмом для генерации случайной величины, распределенной по экспоненциальному закону является метод именуемый Ziggurat (Зиккурат). Функция плотности распределения делится на прямоугольники с одинаковой площадью S. Эти прямоугольники расположены горизонтально, в виде слоев. Самый первый слой, строится таким образом:

Выбираем точки x_1 и y_0 у основания функции f(x). То есть мы получаем прямоугольник от $(0; 0)$ до $(x_1; y_0)$ и еще хвостовая часть функции $f(x_1 > x)$ площадь которой равна S. Поверх него ставится следующий слой, прямоугольник шириной x_1 и высотой $y_1 = \frac{S}{y_0}$. И так строим все оставшиеся прямоугольники до вершины нашей функции.

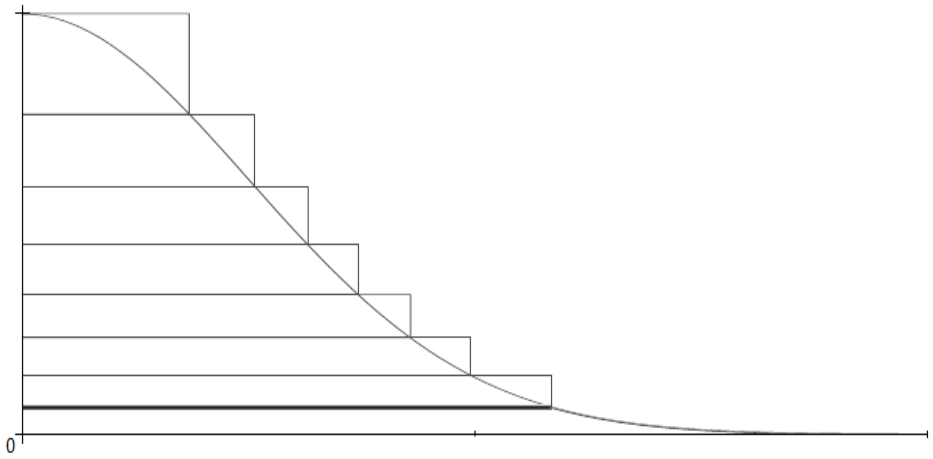


Рис.6 Пример построения слоев в методе Ziggurat

Функция для покрытия экспоненциальной функции плотности экспоненциального распределения слоями выглядит следующим образом:

```
static double x[257], y[256];
const double x1e = 7.69711747013104972;
const double Se = 3.9496598225815571993e-3;
void ReactExponential() {
    y[0] = exp(-x1e);
    x[256] = 0;
    x[0] = Se / y[0];
    for (unsigned i = 1; i <= 255; ++i)
    {
        x[i] = -log(y[i - 1]);
        y[i] = y[i - 1] + Se/x[i];
    }
}
```

Алгоритм попадания точки во все слои, кроме первого слоя выглядит следующим образом:

Случайно выбираем i-й прямоугольник и генерируем равномерно распределенную случайную величину x на $[0; x_i]$.

Если $x < x_{i+1}$ – возвращаем x.

Генерируем равномерно распределенную случайную величину y на $[y_i; y_{i+1}]$. Если точка $[x; y]$ лежит под графиком функции, то есть выполняется условие $f(x) > y$ возвращаем x. Если это условие не выполняется, возвращаемся к пункту 1.

Алгоритм попадания точки в первый слой:

Если попали в первый слой возвращаем x_1+ ExpZ();

Сама реализация генерирует экспоненциальное распределение с $\lambda = 1$. Для этого мы создали дополнительная функцию ExponentialZiggurat(double lambda). Реализация алгоритма выглядит следующим образом:

```
double ExpZ() {
    int i = 0, idReact;
    unsigned int r;
    double e;
    for (;;) {
    _rdrand32_step(&r);
    idReact = r & 255;
    e = UniformDistribution(0, x[idReact]);
    if (e < x[idReact + 1]) // Если попали в любой прямоугольник кроме 1го
    return e;
    if (idReact == 0) // Если попали в 1й прямоугольник
    return x1e + ExpZ(); // рекурсия, в случае попадания в 1й прямоугольник
    if (UniformDistribution(y[idReact - 1], y[idReact]) < exp(-e))
    return e;
    return 0;
```

Для получения из стандартного экспоненциального распределения, распределение с нужным нам λ, необходимо поделить его на λ.

```
double ExponentialZiggurat(double lambda) { // Дополнительная функция
    return ExpZ() / lambda;
```

Практические испытания показали, что метод обратных преобразований ExponentialDistSmirnov дает более высокую скорость чем метод Ziggurat, прирост производительности составляет около 35%. Это объясняется тем что обратная функция от экспоненциального распределения не является вычислительно тяжелой. Ниже приведена гистограмма, для доказательства того что алгоритм основанный на методе обратных преобразований работает ($\lambda = 1.5$).

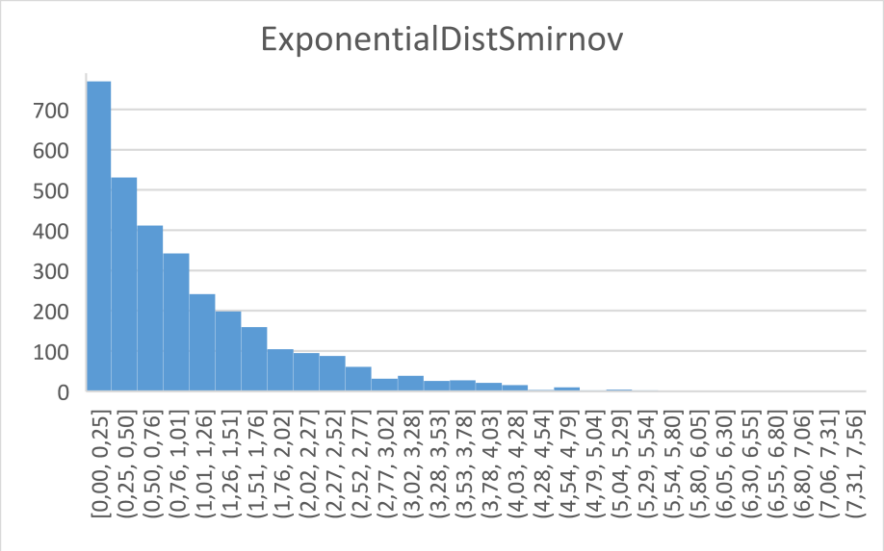


Рис.7 Гистограмма на основе выборки, полученной с помощью ExponentialDistSmirnov

Для того что бы определить, получили ли мы выборку случайных величин с нужным нам рапределением, методом точечной оценки определим величину математического ожидания:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} X_i = 0{,}6666577$$

Вычислим величину $\lambda = \frac{1}{\mu} = \frac{1}{0{,}6666577} = 1{,}50002018 \approx 1{,}5$. Видим, что мы получили выборку с экспоненциальным распределением нужного параметра.

Тестирование

Для тестирования библиотеки использовались 2 процессора Intel разных микроархитектур:

Настольный процессор Core i5-3340 Ivy Bridge

Мобильный процессор Core i5-5200U Broadwell

Все используемые компьютеры работали на штатных частотах и ОС Windows 8. Тестирование проводилось только для самых быстрых из всех реализованных алгоритмов. Тест заключается в генерации 5 млн. случайных величин экспоненциального распределения, Время, затраченное на генерацию приведено в миллисекундах. Для сравнения те же алгоритмы были реализованы на генераторе псевдослучайных последовательностей rand() встроенном в Си. И результаты на том же мобильном процессоре Core i5-5200U намного хуже, время генерации в 3-6 раз больше чем у нашей реализации. Результаты тестирования приведены в таблице 1.

Таблица 1. Результаты испытаний разработанной библиотеки

Метод	Core i5-3340	Core i5-5200U
Результаты испытаний разработанной библиотеки	607	1094
Результаты испытаний генераторов на основе rand()	-	3569

Заключение

В ходе научной работы, была спроектирована библиотека для генерации истинно случайных величин, распределенных по закону экспоненциального распределения. В отличии от существующих библиотек, разработанная библиотека показывает скорость от 3 до 6 раз выше чем остальные. Так же необходимо заметить, что данная библиотека генерирует по-настоящему случайные числа, в то время как большинство остальных генерируют псевдослучайные, что может плохо повлиять на результаты исследований, в ходе которых используются эти выборки.

Литература

1. Колмогоров А.Н. Теория вероятностей и математическая статистика. // М.: Наука, 2005, 584 с.
2. Кнут Д. Искусство программирования. Том 2. Получисленные алгоритмы. // М.: Вильямс, 2001, 832 с.
3. Теория вероятностей. Под редакцией Зарубина В.С., Крищенко А.П. // М.: Издательство МГТУ имени Н.Э. Баумана, 2006, 456 с.
4. Математическая статистика. Под редакцией Зарубина В.С., Крищенко А.П. // М.: Издательство МГТУ имени Н.Э. Баумана, 2002, 424 с.

Literature

1. Kolmogorov A.N. Theory of Probability and Mathematical Statistics. // M .: Nauka, 2005, 584 p.
2. Knut D. The art of programming. Volume 2. The resulting algorithms. // M .: Williams, 2001, 832 p.
3. Probability theory. Edited by V.S. Zarubin, A.P. Krishchenko // M .: Publishing house of the Moscow State Technical University named after N.E. Bauman, 2006, 456 p.
4. Mathematical statistics. Edited by V.S. Zarubin, A.P. Krishchenko // M .: Publishing house of the Moscow State Technical University named after N.E. Bauman, 2002, 424 p.