УДК 681.3.06 DOI: 10.18698/0236-3933-2020-1-101-123

ВИХРЕВОЙ ГЕНЕРАТОР СЛУЧАЙНЫХ ВЕЛИЧИН ПУАССОНА ПО ТЕХНОЛОГИИ КУМУЛЯТИВНЫХ ЧАСТОТ

А.Ф. Деон[1] deonalex@mail.ru
Д.Д. Дмитриев[1] ddd.1955@gmail.com
Ю.А. Меняев[2] yamenyaev@uams.edu

[1] **МГТУ им. Н.Э. Баумана, Москва, Российская Федерация**
[2] **Институт исследования рака им. Уинтропа Рокфеллера, Литл-Рок, Арканзас, США**

Аннотация

Широко известные генераторы случайных величин Пуассона связаны с различными модификациями алгоритма на основе сходимости по вероятности последовательности равномерных случайных величин к пуассоновской случайной величине. Однако такой подход в некоторых ситуациях дает различные дискретные распределения вероятностей Пуассона и пропуски генерируемых величин. Предложен новый подход для создания случайных величин Пуассона на основе полного вихревого генератора равномерных случайных величин, используя технологию кумулятивных частот. Результаты моделирования подтверждают, что вероятностное и частотное распределение получаемых величин полностью совпадает с теоретическим распределением Пуассона. Кроме того, комбинирование нового подхода с алгоритмом настройки базовой вихревой генерации позволяет существенно увеличить длину создаваемых последовательностей без использования дополнительной оперативной памяти компьютера

Ключевые слова

Генератор псевдослучайных величин, случайные последовательности, распределение Пуассона, вихревой генератор

Поступила 15.11.2019
Принята 25.11.2019

Введение. Направление генераторов пуассоновских случайных величин реализует процесс создания целых случайных величин $\eta \in \mathrm{H}$, имеющих следующее распределение вероятностей по вещественному параметру α [1, 2]:

$$P(\eta, \alpha) = \frac{\alpha^{\eta}}{\eta!} e^{-\alpha}, \quad (1)$$

где η принимает любые целые значения $0, 1, 2, \ldots, \infty$.

Распределение Пуассона обладает замечательными свойствами начальных вероятностных моментов по математическому ожиданию $m(\eta,\alpha)=E_1(\eta,\alpha)=\alpha$ и дисперсии $D(\eta,\alpha)=E_2(\eta,\alpha)=E_1(\eta^2,\alpha)=\alpha.$ Эти и другие свойства позволяют использовать распределение Пуассона в теоретической и статистической математике [3, 4], в исследовании физических явлений [5, 6], в радиотехнике и ядерной физике [7, 8], в информатике и информационных системах [9], в моделировании сетей передачи данных [10, 11], в экономике и финансовом анализе [12], а также в других областях вплоть до биологических исследований [13–15] и разработок для медицинской физики и техники [16–18].

Существуют различные варианты реализации генераторов псевдослучайных величин на основе распределения Пуассона. Широко известен генератор, предложенный Д. Кнутом [19, 20] и активно используемый его последователями. Генератор представлен на произвольном языке псевдокода, где параметр α имеет обозначение λ, а случайная величина η равна k:

algorithm Poisson random number (Knuth):
init:
Let $L \leftarrow e-\lambda$, $k \leftarrow 0$ *and* $p \leftarrow 1$.
do:
    $k \leftarrow k + 1$.
    Generate uniform random number u in [0, 1] and let $p \leftarrow p \times u$.
while $p > L$.
return $k - 1$.

Далее представлен программный код на языке C# для *Microsoft Visual Studio*. Согласно (1), вместо величины λ используется параметр α, которому можно присвоить произвольное значение, например $\alpha = 2,0$. Функция *KnuthPoisson()* создает целые числа k, являющиеся аналогом случайных величин η в (1). Их частотное распределение запоминается в массиве `nuK`. В качестве генератора равномерных случайных величин применяется функция *Random.Next()*. Функция *PoissonP()* размещает в массиве `pEta` вероятности событий $P(\eta,\alpha)$. Для формирования распределения используются $N = 2^w = 2^{16} = 65536$ равномерных случайных величин. Программные имена *P060102* и *cP060102* выбраны произвольным образом.

Программный код на языке C# для *Microsoft Visual Studio*

```
namespace P060102
{  class cP060102
   {  static uint gc = 0;          // число равномерных генераций
```

```
static void Main(string[] args)
{  int w = 16;  // битовая длина равномерных целых величин
   long N = 1L << w;       // количество случайных величин
   Console.WriteLine("w = {0}  N = {1}", w, N);
   double Alpha = 2.0;
   Console.WriteLine("Alpha = {0:F2}", Alpha);
   Random G = new Random();       // равномерный генератор
   int wX = 200;
   int[] nuK = new int[wX];            // частоты по Кнуту
   for (int i = 0; i < wX; i++) nuK[i] = 0;
   int maxK = 0;                     // длина распределения
   for (int i = 0; i < N; i++)
   {  int k = KnuthPoisson(Alpha, N, G);
      nuK[k]++;                        // частота по Кнуту
      if (k > maxK) maxK = k;       // длина распределения
   }
   Console.WriteLine("maxK = {0}", maxK);
   double[] pEta = new double[wX];          // вероятности
   long[] nuEta = new long[wX];        // частоты Пуассона
   int cEta = PoissonP(Alpha, N, pEta, nuEta);
   VerifyProbability(N, cEta, pEta, nuEta);
   Console.WriteLine("cEta = {0} ", cEta);
   long snuEta = 0;                // сумма частот Пуассона
   double spEta = 0.0;      // сумма вероятностей Пуассона
   int snuK = 0;                   // сумма частот по Кнуту
   double spK = 0.0;        // сумма вероятностей по Кнуту
   snuEta = 0;                  // сумма частот по Пуассону
   spEta = 0.0;          // сумма вероятностей по Пуассону
   Console.Write("Eta     pK              nuK");
   Console.Write("       pEta         nuEta");
   Console.WriteLine("     nuK - nuEta");
   int nEta = cEta > maxK ? cEta : maxK;
   for (int i = 0; i <= nEta; i++) // табулирование частот
   {  double pK = (double)nuK[i] / (double)N;
      int dnu = (int)(nuK[i] - nuEta[i]);
      Console.Write("{0,2}  {2,12:F10}  {1,10}",
                    i, nuK[i], pK);
      Console.Write("  {0,12:F10} {1,10}",
                    pEta[i], nuEta[i]);
      Console.WriteLine("  {0,8}", dnu);
      snuK += nuK[i];             // сумма частот по Кнуту
      spK += pK;            // сумма вероятностей по Кнуту
      snuEta += nuEta[i];      // сумма частот по Пуассону
      spEta += pEta[i];  // сумма вероятностей по Пуассону
   }
   Console.Write("sum    spK             snuK");
   Console.WriteLine("      spEta        snuEta");
   Console.Write("    {0,12:F10}  {1,10}",
                 spK, snuK);
   Console.WriteLine("  {0,12:F10} {1,10}",
                     spEta, snuEta);
   Console.WriteLine("gc = {0}", gc);
```

```
         Console.ReadKey();               // просмотр результата
      }
//-----------------------------------------------------------------
      static int KnuthPoisson(double Lam, long N, Random G)
      {  double L = Math.Exp(-Lam);
         double p = 1.0;
         double dN = (double)N;
         int k = 0;
         do
         {  k++;
            long z = (long)G.Next();     // равномерная величина
            z = z & (N - 1);
            double u = (double)z / dN;
            p = p * u;
            gc++;        // всеобщее число равномерных генераций
         } while (p > L);
         return k - 1;
      }
//-----------------------------------------------------------------
        static int PoissonP(double alpha, long N,
                            double[] pEta, long[] nuEta)
        {
            double emAlpha = Math.Exp(-alpha);
            double spEta = 0.0;            // сумма вероятностей
            long snuEta = 0L;                    // сумма частот
            pEta[0] = 1.0 * emAlpha;// вероятность p(0) Пуассона
            spEta += pEta[0];              // сумма вероятностей
            nuEta[0] = (long)Math.Round(pEta[0] * (double)N);
            snuEta += nuEta[0];                  // сумма частот
            double r = alpha;        // первое слагаемое Тейлора
            pEta[1] = r * emAlpha;  // вероятность p(1) Пуассона
            spEta += pEta[1];              // сумма вероятностей
            nuEta[1] = (long)Math.Round(pEta[1] * (double)N);
            snuEta += nuEta[1];                  // сумма частот
            int Eta = 2;                   // случайная величина
            do
            {
                r *= alpha / (double)Eta;   // слагаемое для exp
                double p = r * emAlpha;    // вероятность p(Eta)
                long nu = (long)Math.Round(p * (double)N);
                long sd = snuEta + nu;
                if (nu == 0L  || sd > N) break;         // хвост
                pEta[Eta] = p;             // вероятность p(Eta)
                spEta += p;             // суммарная вероятность
                nuEta[Eta] = nu;               // частота nu(Eta)
                snuEta += nu;                  // сумма частот
                Eta++;       // следующая случайная величина Eta
            } while (snuEta < N);

            long d = N - snuEta;           // недостающие частоты
            if (d == 0L) return Eta - 1;
            double d1N = (1.0 - spEta) / (double)d;
```

```
            do
            {
                pEta[Eta] = d1N;//вероятность хвостового события
                nuEta[Eta] = 1;         // одночастотное событие
                snuEta++;                       // сумма частот
                Eta++;
            } while (snuEta < N);
            return Eta - 1;
      }
//-------------------------------------------------------------
      static void VerifyProbability(long N, int cEta,
                                    double[] pEta, long[] nuEta)
      {  double dN = (double)N;
         for (int i = 0; i <= cEta; i++)
            pEta[i] = (double)nuEta[i] / dN;
      }
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   }
}
```

После запуска программы *P060102* на мониторе появится следующий результат:

```
w = 16  N = 65536
Alpha = 2.00
maxK = 12
cEta = 10
Eta     pK           nuK        pEta         nuEta   nuK - nuEta
 0   0.1352539063    8864    0.1353302002     8869     -5
 1   0.2691345215   17638    0.2706756592    17739   -101
 2   0.2699890137   17694    0.2706756592    17739    -45
 3   0.1814422607   11891    0.1804504395    11826     65
 4   0.0910491943    5967    0.0902252197     5913     54
 5   0.0361175537    2367    0.0360870361     2365      2
 6   0.0125427246     822    0.0120239258      788     34
 7   0.0032806396     215    0.0034332275      225    -10
 8   0.0008850098      58    0.0008544922       56      2
 9   0.0002441406      16    0.0001983643       13      3
10   0.0000457764       3    0.0000457764        3      0
11   0.0000000000       0    0.0000000000        0      0
12   0.0000152588       1    0.0000000000        0      1
sum    spK          snuk       spEta       snuEta
     1.0000000000   65536    1.0000000000    65536
gc = 197025
```

В этом листинге столбцы `pK` и `nuK` показывают вероятностные и частотные значения, полученные по алгоритму Кнута. Эти величины хорошо соответствуют аналогичным столбцам `pEta` и `nuEta`, которые рассчитываются по модели Пуассона (1). Однако есть некоторые особенности, которые следует рассмотреть.

Первый недостаток связан с тем, что строка 11 показывает 0 по алгоритму Кнута. Это означает, что генератор не создал случайную величину 11, хотя при этом он создал одну случайную величину 12. В теоретическом распределении Пуассона это не допускается. Для некоторых прикладных задач, которые не ограничены строгими условиями, этим можно было бы пренебречь. Однако если генератор используется для всестороннего моделирования реальных ситуаций, то лучше этого не допускать.

Второй недостаток проявляется тогда, когда генератор Кнута запускается многократно. Контроль значений в столбцах `pK` и `nuK` фиксирует их непостоянство. В теории вероятностей по аксиоматике Колмогорова категорически не допускается изменение вероятностной меры в заданном пространстве ни при каких обстоятельствах. Нарушение аксиом пространства может затруднить интерпретацию проводимых испытаний, когда требуются повторные опыты, например, как в случае аварийных ситуаций, т. е. когда необходимо повторять особые наблюдения.

Третий недостаток алгоритма Кнута связан с тем, что счетчик `gc`, который подсчитывает число генерируемых равномерных случайных величин, оказался равным 197 025. Цикл генератора Кнута произвольный, следовательно, значение `gc` также может быть произвольным. В рассматриваемом случае значение `gc` = 197 025, что почти в 3 раза превосходит число $N = 65\,536$ генерируемых равномерных случайных величин, из которых алгоритм Кнута создает случайные величины Пуассона. Это влечет неконтролируемые повторения базовых случайных величин с нарушением их равномерности и, как следствие, приводит к недостаточно высокому качеству получаемых результатов.

Необходимо ликвидировать эти недостатки. *Цель работы* — создание генератора случайных величин в строгом соответствии с теорией распределения Пуассона без избыточной и промежуточной генерации равномерных величин.

Теория. Процесс Пуассона оперирует со случайными величинами, которые линейно зависят от непрерывного параметра. Обычно таким параметром является время наблюдения случайного события, но возможны и другие интерпретации. Интерес представляет как сам момент времени t, так и следующий за ним интервал времени τ. Случайность событий в двух последовательных непрерывных интервалах времени $[0, t]$ и $(t, t+\tau]$ предполагает [21], что за общее время $[0, t+\tau]$ могут произойти случайные события числом η. Если события числом k наблюдаются в интервале $[0, t]$, то $\eta - k$ событий должны произойти в полуоткрытом интервале

$(t, t+\tau]$. Первое аксиоматическое ограничение связано с тем, что оба интервала независимы и что события в них по раздельности происходят также независимо. Следствием этого ограничения является то, что вероятность наблюдения событий η на общем интервале $[0, t+\tau]$ является совместной вероятностью независимых испытаний:

$$P_{\eta}\left(0, t+\tau\right]\right) = P_k([0,t])P_{\eta-k}((t,t+\tau]). \quad (2)$$

В модели Пуассона вероятность $P_{\eta-k}((t,t+\tau])$ также имеет серьезные ограничения, которые можно сформулировать следующим образом.

1. Вероятность событий в интервале времени $(t, t+\tau]$ не зависит от его начала t:

$$P_{\eta-k}\left((t,t+\tau]\right) = P\left((0,\tau]\right). \quad (3)$$

2. Вероятность одного события в интервале времени $(0,\tau]$ линейно зависит от длины интервала τ с заданной интенсивностью λ, причем вероятность $o(\tau)$ наблюдения других событий ничтожно мала:

$$P_1\left((0,\tau]\right) = P_1\left(\tau\right) + \alpha + o\left(\tau\right). \quad (4)$$

В выражении (4) используется общепринятое в теории вероятностей [2] и теории случайных процессов обозначение $\alpha = \lambda\tau$ (1). Условие (4) исключает наблюдение в интервале времени τ одновременно двух и более событий. Это позволяет упростить его, не рассматривая события с бесконечно малой вероятностью более высокого порядка $o(\tau)$:

$$P_1\left(\tau\right) = \lambda\tau. \quad (5)$$

Выражение (5) позволяет однозначно определить вероятность отсутствия события в промежутке времени τ:

$$P_0\left(\tau\right) = 1 - P_1\left(\tau\right) = 1 - \lambda\tau. \quad (6)$$

Применяя совместно (2) и (6), получаем вероятность отсутствия событий в момент времени t в общем интервале $\left[0, t+\tau\right]$:

$$P_0\left(t+\tau\right) = P_0\left(t\right)P_0\left(\tau\right) = P_0\left(t\right)\left(1-\lambda\tau\right). \quad (7)$$

Выражение (7) ведет к определению производной по вероятности:

$$\frac{dP_0\left(t\right)}{d\tau} = \lim_{\tau\to 0}\frac{P_0\left(t+\tau\right) - P_0\left(t\right)}{\tau} = -\lambda P_0\left(t\right). \quad (8)$$

Решение дифференциального уравнения (8) определяет вероятность отсутствия событий в момент времени t, в котором константа $c=1$ вычисляется из начального условия $P_0(0)=1$. В результате получаем

$$P_0(t)=ce^{-\lambda t}=e^{-\lambda t}. \tag{9}$$

Выражение (9) с учетом (2) и ограничений (5) и (6) позволяет вычислить вероятность одиночного события $P_1([0,t+\tau])$ следующим образом:

$$P_1([0,t+\tau])=P_0(t)P_1(\tau)+P_1(t)P_0(\tau)=$$

$$=e^{-\lambda t}\lambda\tau+P_1(t)(1-\lambda\tau). \tag{10}$$

По аналогии с преобразованиями (7) и (8) выражение (10) ведет к дифференциальному уравнению

$$\frac{dP_1(t)}{d\tau}=\lambda e^{-\lambda t}-\lambda P_1(t). \tag{11}$$

Решение уравнения (11) определяет $P_1(t)$ при начальном условии $P_1(0)=0$:

$$P_1(t)=\lambda te^{-\lambda t}. \tag{12}$$

Выполняя последовательно преобразования (10) и (11) для всех величин $\eta\in[0,\infty]$, получаем распределение вероятностей Пуассона $P_\eta(\lambda t)$ числа случайных событий η с интенсивностью λ в момент времени t:

$$P_\eta(\alpha=\lambda t)=\frac{(\lambda t)^\eta}{\eta!}e^{-\lambda t}=\frac{\alpha^\eta}{\eta!}e^{-\alpha}. \tag{13}$$

Поскольку множество $\eta\in \mathrm{H}=\left[\overline{0:\infty}\right]$ случайных событий по аксиоматике Колмогорова содержит σ-алгебру, то вероятностная мера (13) однозначно определяет кумулятивную функцию распределения вероятностей:

$$F_{\mathrm{H}}(\eta,\alpha=\lambda t)=\sum_{k=0}^{\eta}\frac{(\lambda t)^k}{k!}e^{-\lambda t}=\sum_{k=0}^{\eta}\frac{\alpha^k}{k!}e^{-\alpha}. \tag{14}$$

Используя определение числа $e^x=\sum\limits_{k=0}^{\infty}x^k/k!$ в выражении (14), получаем, что $F_{\mathrm{H}}(\eta,\lambda t)\in[0,1]$. Следует отметить, что вероятностное пространство гарантирует однозначность определения обратной функции распределения вероятностей. Если задано какое-либо значение h кумулятивной функции распределения $F_{\mathrm{H}}(\eta_h,\alpha=\lambda t)=h$, то значение случай-

ной величины η_h можно получить как обратное преобразование $\eta_h = F_{\mathrm{H}}^{-1}(h)$. Таким образом, задавая полные равномерные случайные значения $F_{\mathrm{H}}^{-1}(\eta, \alpha = \lambda t) \in [0,1]$, можно однозначно получать случайные величины η этого распределения. Эта главная математическая модель содержит основания для конструирования генераторов случайных величин по заданным функциям их распределения. Используем это утверждение для разработки генератора пуассоновских случайных величин. Для этого необходимо иметь абсолютно полный и равномерный генератор без повторений и пропусков случайных величин [22–25].

В дискретном вероятностном пространстве число N событий фиксировано. Каждой количественной величине η с вероятностью $p(\eta, \alpha = \lambda t)$ соответствует частота $\nu(\eta, \alpha = \lambda t)$ наблюдений случайных событий:

$$\nu(\eta, \alpha = \lambda t) = p(\eta, \alpha = \lambda t) N = P_{\eta}(\lambda t) N. \qquad (15)$$

Согласно аксиоматике Колмогорова, определение вероятности $p(\eta, \alpha = \lambda t)$ имеет приоритет над определением частоты $\nu(\eta, \alpha = \lambda t)$. Поэтому предложено для значений частоты $\nu(\eta, \alpha = \lambda t)$ в (15) использовать математическое округление в дробной части для произведения $p(\eta, \alpha = \lambda t) N$.

Программный код, в котором вихревой генератор *nsDeonYuliTwist32D* [24] создает полное множество $\left[\overline{0:2^w - 1}\right]$ равномерных целых случайных величин z длиной w бит, представлен ниже. Функция *PoissonD()* формирует массив `pEta` вероятностей Пуассона (13) и массив соответствующих частот `nuEta` (15). Последние хвостовые единичные частоты дополняют массив распределений частот `nuEta` до полноты 2^w базовых равномерных случайных величин (15). Соответственно дополняются хвостовые вероятности `pEta` по хвостовым единичным частотам в `nuEta`. Такое незначительное отклонение в хвостовой части распределения (13) позволяет сохранить полноту генерации пуассоновских случайных величин по исходной генерации равномерных случайных величин $z \in \left[\overline{0:2^w - 1}\right]$. Значения кумулятивной частотной функции размещены в массиве суммируемых частот `snuEta`. Обратная функция $F_{\mathrm{H}}^{-1}(\eta_h, \alpha = \lambda t)$ реализована с помощью функции *SearchEta()* по алгоритму поиска индекса элемента в массиве кумулятивных частот `cnuEta`. Программные имена *P060202* и *cP060202* выбраны произвольно.

Программный код

```
using nsDeonYuliTwist32D;           // вихревой полный генератор
                                    // целых равномерных величин
namespace P060202
{  class cP060202
   {  static void Main(string[] args)
      {  int w = 32;  // битовая длина равномерных целых величин
         long N = 1L << w;       // количество случайных величин
         Console.WriteLine("w = {0}  N = {1}", w, N);
         double Alpha = 2.0;
         Console.WriteLine("Alpha = {0:F2}", Alpha);
         int wX = 200;
         double[] pEta = new double[wX]; // вероятности Пуассона
         long[] nuEta = new long[wX];        // частоты Пуассона
         long[] cnuEta = new long[wX];   // кумулятивные частоты
         int cEta = PoissonDY(Alpha, pEta, nuEta, cnuEta, N);
         VerifyProbability(N, cEta, pEta, nuEta, cnuEta);
         Console.WriteLine("cEta = {0}", cEta);
         cDeonYuliTwist32D DYG = new cDeonYuliTwist32D();
         DYG.SetW(w);       // битовая длина равномерных величин
         DYG.Start();            // старт равномерного генератора
         long[] nuDYG = new long[wX];      // частоты генератора
         for (int i = 0; i < wX; i++) nuDYG[i] = 0;
         for (long j = 0; j < N; j++)
         {  long z = DYG.Next();         // равномерная величина
            int Eta = SearchEta(z, cnuEta, cEta);
            nuDYG[Eta]++;           // счетчик случайных величин
         }
         double spEta = 0.0;   // сумма вероятностей по Пуассону
         long snuEta = 0;            // сумма частот по Пуассону
         long snuDYG = 0;         // сумма величин по генератору
         Console.Write("Eta      pEta          nuEta");
         Console.WriteLine("      cnuEta       nuDYG");
         for (int Eta = 0; Eta <= cEta; Eta++)
         {  Console.WriteLine(
                 "{0,2}   {1,12:F10}  {2,10}  {3,10}  {4,10}",
                  Eta, pEta[Eta], nuEta[Eta],
                  cnuEta[Eta], nuDYG[Eta]);
            spEta += pEta[Eta];
            snuEta += nuEta[Eta];
            snuDYG += nuDYG[Eta];
         }
         Console.Write("Sum     spEta         snuEta");
         Console.WriteLine("                  snuDYG");
         Console.Write("     {0,12:F10}  {1,10}",
                        spEta, snuEta);
         Console.WriteLine("              {0,10}", snuDYG);
         Console.ReadKey();              // просмотр результата
      }
//-------------------------------------------------------------
      static int PoissonDY (double alpha, double[] pEta,
                            long[] nuEta, long[] cnuEta, long N)
```

```
      {  double emAlpha = Math.Exp(-alpha);
         double spEta = 0.0;               // сумма вероятностей
         long snuEta = 0L;                        // сумма частот
         pEta[0] = 1.0 * emAlpha;   // вероятность p(0) Пуассона
         spEta += pEta[0];                 // сумма вероятностей
         nuEta[0] = (long)Math.Round(pEta[0] * (double)N);
         snuEta += nuEta[0];                      // сумма частот
         cnuEta[0] = snuEta;      // кумулятивная частота cnu(0)
         double r = alpha;           // первое слагаемое Тейлора
         pEta[1] = r * emAlpha;     // вероятность p(1) Пуассона
         spEta += pEta[1];                 // сумма вероятностей
         nuEta[1] = (long)Math.Round(pEta[1] * (double)N);
         snuEta += nuEta[1];                      // сумма частот
         cnuEta[1] = snuEta;      // кумулятивная частота cnu(1)
         int Eta = 2;                      // случайная величина
         do
         {  r *= alpha / (double)Eta;       // слагаемое для exp
            double p = r * emAlpha;        // вероятность p(Eta)
            long nu = (long)Math.Round(p * (double)N);
            long sd = snuEta + nu;
            if (nu == 0L || sd > N) break;              // хвост
            pEta[Eta] = p;                 // вероятность p(Eta)
            spEta += p;                // суммарная вероятность
            nuEta[Eta] = nu;                  // частота nu(Eta)
            snuEta += nu;                        // сумма частот
            cnuEta[Eta] = snuEta;        // кумулятивная частота
            Eta++;           // следующая случайная величина Eta
         } while (snuEta < N);
         Eta--;
         long d = N - snuEta;             // недостающие частоты
         if (d == 0L) return Eta;
         double d1N = (1.0 - spEta) / (double)d;
         do
         {  Eta++;
            pEta[Eta] = d1N;   // вероятность хвостового события
            nuEta[Eta] = 1;             // одночастотное событие
            snuEta++;                            // сумма частот
            cnuEta[Eta] = snuEta;        // кумулятивная частота
         } while (snuEta < N);
         return Eta;
      }
//-------------------------------------------------------------
      static void VerifyProbability (long N, int cEta,
            double[] pEta, long[] nuEta, long[] cnuEta)
      {  double dN = (double)N;
         for (int i = 0; i <= cEta; i++)
                pEta[i] = (double)nuEta[i] / dN;
      }


//-------------------------------------------------------------
      static int SearchEta (long z, long[] cnuEta, int cEta)
```

```
      {  int Eta = 0;
         for (; Eta <= cEta; Eta++)
            if (z < cnuEta[Eta]) break;
         return Eta;
      }
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   }
}
```

После запуска программы *P060202* на мониторе появляется следующий результат:

```
w = 32  N = 4294967296
Alpha = 2.00
cEta = 16
Eta       pEta           nuEta       cnuEta        nuDYG
 0    0.1353352831   581260615   581260615   581260615
 1    0.2706705665  1162521231  1743781846  1162521231
 2    0.2706705665  1162521231  2906303077  1162521231
 3    0.1804470443   775014154  3681317231   775014154
 4    0.0902235222   387507077  4068824308   387507077
 5    0.0360894089   155002831  4223827139   155002831
 6    0.0120298029    51667610  4275494749    51667610
 7    0.0034370865    14762174  4290256923    14762174
 8    0.0008592717     3690544  4293947467     3690544
 9    0.0001909493      820121  4294767588      820121
10    0.0000381898      164024  4294931612      164024
11    0.0000069437       29823  4294961435       29823
12    0.0000011572        4970  4294966405        4970
13    0.0000001781         765  4294967170         765
14    0.0000000254         109  4294967279         109
15    0.0000000035          15  4294967294          15
16    0.0000000005           2  4294967296           2
Sum       spEta         snuEta                  snuDYG
      1.000000000   4294967296              4294967296
```

В этом листинге столбец `pEta` содержит распределение вероятностей Пуассона (13). В следующем столбце `nuEta` находится соответствующее распределение частот. Сумма всех частот `snuEta` = 4 294 967 296 должна совпадать с генерацией базовых равномерных случайных величин длиной $w = 32$ бит полным числом $N = 2^w = 2^{32} = 4\,294\,967\,296$. Счетчики последнего столбца `nuDYG` подтверждают полное совпадение распределения генерируемых случайных величин с теоретическим распределением частот `nuEta`.

Теория завершена. Технология кумулятивного поиска обеспечивает безупречную генерацию случайных величин с распределением частот по вероятностям Пуассона. Тестирование программы *P060202* с произволь-

ным числом случайных величин длиной $w \in \left[\overline{3,32}\right]$ бит и параметром $\alpha \in [0,1,10,0]$ подтверждает безупречность получаемых результатов.

Конструкция и результаты. Класс *nsDeonYuliCPoissonTwist32D*, в котором создаются случайные величины по распределению Пуассона (13), приведен ниже. Этот класс является производным над базовым классом вихревого генератора равномерных случайных величин *nsDeonYuliTwist32D*. Пример генерации пуассоновских случайных величин показан далее в программе *P060302*.

Класс *nsDeonYuliCPoissonTwist32D*

```
using nsDeonYuliTwist32D;           // вихревой полный генератор
                                    // целых равномерных величин
namespace nsDeonYuliCPoissonTwist32D
{  class cDeonYuliCPoissonTwist32D : cDeonYuliTwist32D
   {  public long N;           // количество равномерных событий
      public double dN;        // количество равномерных событий
      public double Alpha = 2.0;              // параметр Alpha
      double emAlpha;                            // exp(-Alpha)
      public int cEta;                       // максимальное Eta
      public double[] pC;          // распределение вероятностей
      public long[] nuC;                // распределение частот
      public long[] cnuC;               // кумулятивные частоты
//-----------------------------------------------------------------
      public cDeonYuliCPoissonTwist32D () {}
//-----------------------------------------------------------------
      public void CStart(double alpha)
      {  Alpha = alpha;                         // параметр Alpha
         base.Start();         // равномерный вихревой генератор
         CStartInside();
      }
//-----------------------------------------------------------------
      public void CTimeStart(double alpha)
      {  Alpha = alpha;                         // параметр Alpha
         base.TimeStart();     // равномерный вихревой генератор
         CStartInside();
      }
//-----------------------------------------------------------------
      void CStartInside()
      {  int wX = 200;
         pC = new double[wX];      // распределение вероятностей
         nuC = new long[wX];             // распределение частот
         cnuC = new long[wX];            // кумулятивные частоты
         emAlpha = Math.Exp(-Alpha);              // exp(-Alpha)
         N = (long)N1 + 1L;    // количество равномерных событий
         dN = (double)N;       // количество равномерных событий
         cEta = CPoissonDY();           // вероятности и частоты
         CVerifyProbability();           // проверка вероятности
      }
```

```
//-------------------------------------------------------------
      public int CNext()
      {  uint z =base.Next();  // равномерная случайная величина
         return CSearchEta(z);    // случайная величина Пуассона
      }
//-------------------------------------------------------------
      int CPoissonDY()
      {  double spC = 0.0;                 // сумма вероятностей
         long snuC = 0L;                        // сумма частот
         pC[0] = 1.0 * emAlpha;     // вероятность p(0) Пуассона
         spC += pC[0];                       // сумма вероятностей
         nuC[0] = (long)Math.Round(pC[0] * dN);//частота nu(0)
         snuC += nuC[0];                        // сумма частот
         cnuC[0] = snuC;          // кумулятивная частота cnu(0)
         double r = Alpha;           // первое слагаемое Тейлора
         pC[1] = r * emAlpha;       // вероятность p(1) Пуассона
         spC += pC[1];                       // сумма вероятностей
         nuC[1] = (long)Math.Round(pC[1] * dN); // частота nu(1)
         snuC += nuC[1];                        // сумма частот
         cnuC[1] = snuC;          // кумулятивная частота cnu(1)
         int Eta = 2;                        // случайная величина
         do
         {  r *= Alpha / (double)Eta;       // слагаемое для exp
            double p = r * emAlpha;        // вероятность p(Eta)
            long nu = (long)Math.Round(p * dN);
            if (nu == 0L) break;    // хвостовые нулевые частоты
            long sd = snuC + nu;
            if (nu == 0L || sd > N) break;              // хвост
            pC[Eta] = p;                    // вероятность p(Eta)
            spC += p;                       // сумма вероятностей
            nuC[Eta] = nu;                     // частота nu(Eta)
            snuC += nu;                           // сумма частот
            cnuC[Eta] = snuC;            // кумулятивная частота
            Eta++;           // следующая случайная величина Eta
         } while (snuC < N);
         Eta--;
         long d = N - snuC;                 // хвостовые частоты
         if (d == 0L) return Eta;
         double d1N = (1.0 - spC) / (double)d;
         do
         {  Eta++;
            pC[Eta] = d1N;     // вероятность хвостового события
            nuC[Eta] = 1;               // одночастотное событие
            snuC++;                               // сумма частот
            cnuC[Eta] = snuC;            // кумулятивная частота
         } while (snuC < N);
            return Eta;
      }
//-------------------------------------------------------------
      void CVerifyProbability()
      {  for (int i = 0; i <= cEta; i++)
            pC[i] = (double)nuC[i] / dN;
```

```
      }
//---------------------------------------------------------------
      int CSearchEta(uint z)
      {  int Eta = 0;
         for (; Eta <= cEta; Eta++)
            if (z < cnuC[Eta]) break;
         return Eta;
      }
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   }
}
```

В качестве примера с помощью следующего программного кода покажем полную генерацию случайных величин Пуассона на базовом пространстве равномерных случайных величин длиной, например, $w = 7$ бит (при произвольном $w \le 32$ листинг $N = 2^{w=32} = 4\,294\,967\,296$ случайных величин оказывается слишком длинным). Это позволяет в наглядной форме показать работу вихревого генератора *nsDeonYuliCPoisson-Twist32D* с соблюдением распределения Пуассона. Программные имена *P060302* и *cP060302* выбраны произвольным образом:

```
using nsDeonYuliCPoissonTwist32D; // вихревой генератор Пуассона
                                  // по технологии кумулятивных частот
namespace P060302
{  class cP060302
   {  static void Main(string[] args)
      {  cDeonYuliCPoissonTwist32D PT =
                        new cDeonYuliCPoissonTwist32D();
         int w = 7;          // битовая длина равномерных величин
         PT.SetW(w);         // битовая длина равномерных величин
         double Alpha = 2.0;                    // параметр Alpha
         PT.CStart(Alpha);      // старт генератора по умолчанию
//       PT.CTimeStart(Alpha);       // старт по датчику времени
         Console.WriteLine("w = {0}  N = {1}", PT.w, PT.N);
         Console.WriteLine("Alpha = {0:F2}", Alpha);
         Console.WriteLine("cEta = {0}", PT.cEta);
         int wX = 200;
         int[] nuG = new int[wX];          // частоты генератора
         for (int i = 0; i < wX; i++) nuG[i] = 0;
         for (int i = 0, j = 1; i < PT.N; i++, j++)
         {  int Eta = PT.CNext();      // пуассоновская величина
            Console.Write("{0,5}", Eta);
            if (j % 8 == 0) Console.WriteLine();
            nuG[Eta]++;             // счетчик случайных величин
         }
         Console.WriteLine();
         double spEta = 0.0;   // сумма вероятностей по Пуассону
         long snuEta = 0;            // сумма частот по Пуассону
         int snuG = 0;               // сумма частот генератора
         Console.Write("Eta      pC            nuC");
         Console.WriteLine("      cnuC     nuDYG");
```

```
         for (int Eta = 0; Eta <= PT.cEta; Eta++)
         {  Console.WriteLine(
                    "{0,2}   {1,12:F10}  {2,8}  {3,8}  {4,8}",
                     Eta, PT.pC[Eta], PT.nuC[Eta],
                     PT.cnuC[Eta], nuG[Eta]);
            spEta += PT.pC[Eta];
            snuEta += PT.nuC[Eta];
            snuG += nuG[Eta];
         }
         Console.Write("Sum    spC            snuC");
         Console.WriteLine("              snuDYG");
         Console.WriteLine(
                "     {0,12:F10}  {1,8}            {2,8}",
                     spEta, snuEta, snuG);
         Console.ReadKey();               // просмотр результата
        }
    }
}
```

После запуска программы *P060302* на мониторе появляется следующий результат:

```
w = 7  N = 128
Alpha = 2.00
cEta = 6
    1    6    3    1    3    1    1    2
    2    1    0    2    0    2    3    3
    3    2    1    4    2    3    0    0
    0    4    2    1    3    0    1    2
    1    1    5    2    0    1    2    3
    3    2    1    4    1    3    4    0
    0    4    2    1    2    0    1    1
    1    1    4    2    5    1    2    2
    2    2    1    3    1    2    4    5
    5    3    2    1    2    5    1    1
    1    0    3    2    4    1    2    2
    2    2    1    3    1    2    3    4
    4    3    2    0    2    4    0    1
    1    0    3    1    4    1    2    2
    2    1    0    3    1    2    3    3
    3    3    2    0    2    3    0    1
Eta      pC           nuC      cnuC    nuDYG
 0    0.1328125000       17        17       17
 1    0.2734375000       35        52       35
 2    0.2734375000       35        87       35
 3    0.1796875000       23       110       23
 4    0.0937500000       12       122       12
 5    0.0390625000        5       127        5
 6    0.0078125000        1       128        1
sum     spC           snuC             snuDYG
      1.0000000000      128               128
```

Этот результат показывает последовательность случайных величин, которые создает вихревой генератор *nsDeonYuliCPoissonTwist32D*. Непосредственный подсчет по листингу подтверждает, что случайные величины, их вероятности и частоты действительно обеспечивают распределение Пуассона. Аналогичные результаты можно получить при других базовых равномерных случайных величинах числом $N = 2^{32}$ и длиной $w \le 32$ бит.

Обсуждение. Как было отмечено выше, распределение Пуассона обладает основными свойствами начальных вероятностных моментов по математическому ожиданию и дисперсии. Математическое ожидание для такого распределения характеризует среднее число успешных результатов на каком-либо интервале. Оно определяется на основе опытно полученных данных для конкретной ситуации. Далее если математическое ожидание найдено, то дисперсия также будет известна в силу свойств распределения вероятностей Пуассона. Если окажется, что значения математического ожидания и дисперсии достаточно близки, то гипотеза о распределении случайной величины в соответствии с законом Пуассона верна. Однако если будет наблюдаться резкое различие в значениях этих характеристик, то это будет свидетельствовать против гипотезы о пуассоновском распределении этой случайной величины.

Таким образом, прежде всего необходимо подтвердить два основных указанных выше свойства распределения Пуассона о равенстве параметра α математическому ожиданию и дисперсии. Программный код, проверяющий это, представлен ниже (здесь генератор создает $N = 2^w = 2^{32} = 4\,294\,967\,296$ случайных величин; программные имена *P060402* и *cP060402* выбраны произвольно):

```
using nsDeonYuliCPoissonTwist32D; // вихревой генератор Пуассона
                                  // по технологии кумулятивных частот
namespace P060402
{  class cP060402
   {  static void Main(string[] args)
      {  cDeonYuliCPoissonTwist32D PT =
                                  new cDeonYuliCPoissonTwist32D();
         PT.SetW(32);             // битовая длина равномерных величин
         double Alpha = 2.0;                    // параметр Alpha
         PT.CStart(Alpha);                    // старт генератора
         Console.WriteLine("w = {0}  N = {1}", PT.w, PT.N);
         Console.WriteLine("Alpha = {0:F2}", Alpha);
         double p1 = 1.0 / (double)PT.N;  // вероятность события
         double m = 0.0;                // математическое ожидание
         double D = 0.0;                                // дисперсия
         for (long i = 0; i < PT.N; i++)
```

```
         {  int Eta = PT.CNext();          // случайная величина
            m += Eta * p1;              // математическое ожидание
            D += Eta * Eta * p1;                    // дисперсия
         }
         D = D - m * m;
         Console.WriteLine("m = {0:F10}", m);
         Console.WriteLine("D = {0:F10}", D);
         Console.ReadKey();               // просмотр результата
      }
   }
}
```

После запуска программы *P060401* на мониторе появляется листинг:

```
w = 32  N = 4294967296
Alpha = 2.00
m = 2.0000000014
D = 2.0000000116
```

Этот листинг показывает полученные значения математического ожидания m и дисперсии D. Их небольшое отличие от параметра $\alpha = 2{,}0$ связано с дискретностью модели Пуассона по вероятностям и частотам событий. Таким образом, полное дискретное моделирование случайных величин с распределением Пуассона на основе вихревого базового генератора равномерных случайных величин подтверждает важные свойства по математическому ожиданию и дисперсии. Кроме того, автоматическое продление серий базовых равномерных случайных величин значительно расширяет период неповторяемости и серий случайных величин Пуассона. Это значительно превосходит известные генераторы [19, 20] на основе алгоритма вероятностной сходимости.

Заключение. Анализ исходного материала показывает, что алгоритмы генерации случайных величин Пуассона по технологии вероятностной сходимости допускают различные реализации распределения. Кроме того, могут наблюдаться пропуски в таких распределениях, что неверно в теории процессов Пуассона. Поскольку качество работы генераторов сильно зависит от используемых базовых генераторов равномерных случайных величин, предложено применить равномерный вихревой генератор *nsDeonYuliCPoissonTwist32D*, чтобы обеспечить полноту технологии кумулятивного массива частот на пространстве Пуассона. Это позволяет сократить время вычислений и повысить качество генерации. Проведенные эксперименты реально подтверждают полное совпадение распределений получаемых случайных величин с теоретическими выводами на дискретном вероятностном пространстве Пуассона. Автоматическая настройка

параметров используемого базового вихревого равномерного генератора *nsDeonYuliTwist32D* позволяет значительно увеличить общий период неповторяемости генерации серий случайных величин Пуассона. Полученные результаты можно использовать во многих прикладных задачах.

ЛИТЕРАТУРА

[1] Feller W. An introduction to probability theory and its applications. Vol. 2. WSE Press, 2008.

[2] Gnedenko B. Theory of probability. CRC Press, 1998.

[3] Zhang H., Li B. Characterizations of discrete compound Poisson distribution. *Commun. Stat. — Theory Methods*, 2016, vol. 45, iss. 22, pp. 6789–6802. DOI: https://doi.org/10.1080/03610926.2014.901375

[4] Guerriero V. Power low distribution: method of multi-scale inferential statistics. *JMMF*, 2012, vol. 1, no. 1, pp. 21–28.

[5] Arkani M., Khalafi H., Vosoughi N. A flexible multichannel digital random pulse generator based on FPGA. *WJNST*, 2013, vol. 3, no. 4, pp. 109–116. DOI: https://doi.org/10.4236/wjnst.2013.34019

[6] Rasoanaivo A.N., Horowitz W.A. Medium-induced radiation beyond the Poisson approximation. *J. Phys.: Conf. Ser.*, 2017, vol. 878, art. 012029. DOI: https://doi.org/10.1088/1742-6596/878/1/012029

[7] Veiga A., Spinelli E. A pulse generator with poisson-exponential distribution for emulation of radioactive decay events. *Proc. 7th LASCAS*, 2016, pp. 31–34. DOI: https://doi.org/10.1109/LASCAS.2016.7451002

[8] Kirkpatrick J.M., Young B.M. Poisson statistical methods for the analysis of low-count gamma spectra. *IEEE Trans. Nucl. Sci.*, 2009, vol. 56, iss. 3, pp. 1278–1282. DOI: https://doi.org/10.1109/TNS.2009.2020516

[9] Marsaglia G., Tsang W.W., Wang J. Fast generation of discrete random variables. *J. Stat. Softw.*, 2004, vol. 11, iss. 3, pp. 1–11. DOI: https://doi.org/10.18637/jss.v011.i03

[10] Kumari S., Valarmathi M., Prince S. Generation of pseudorandom binary sequence using shot noise for optical encryption. *Proc. ICCSP,* 2016, pp. 0119–0122. DOI: https://doi.org/10.1109/ICCSP.2016.7754537

[11] Hosamo M. A study of the source traffic generator using Poisson distribution for ABR service. *Model. Simul. Eng.*, 2012, vol. 2012, art. 408395. DOI: https://doi.org/10.1155/2012/408395

[12] Zhang H., Liu Y., Li B. Notes on discrete compound Poisson model with applications to risk theory. *Insur.: Math. Econ.*, 2014, vol. 59, pp. 325–336. DOI: https://doi.org/10.1016/j.insmatheco.2014.09.012

[13] Shanmugam R. Informatics about fear to report rapes using bumped-up Poisson model. *Amer. J. Biostat.*, 2013, vol. 3, iss. 1, pp. 17–29. DOI: https://doi.org/10.3844/amjbsp.2013.17.29

[14] Menyaev Yu.A., Nedosekin D.A., Sarimollaoglu M., et al. Optical clearing in photoacoustic flow cytometry. *Biomed. Opt. Express*, 2013, vol. 4, iss. 12, pp. 3030–3041. DOI: https://doi.org/10.1364/BOE.4.003030

[15] Menyaev Yu.A., Carey K.A., Nedosekin D.A., et al. Preclinical photoacoustic models: application for ultrasensitive single cell malaria diagnosis in large vein and artery. *Biomed. Opt. Express*, 2016, vol. 7, iss. 9, pp. 3643–3658. DOI: https://doi.org/10.1364/BOE.7.003643

[16] Sitek A., Celler A.M. Limitations of Poisson statistics in describing radioactive decay. *PM*, 2015, vol. 31, iss. 8, pp. 1105–1107. DOI: https://doi.org/10.1016/j.ejmp.2015.08.015

[17] Menyaev Yu.A., Zharov V.P. Experience in development of therapeutic photomatrix equipment. *Biomed. Eng.*, 2006, vol. 40, iss. 2, pp. 57–63. DOI: https://doi.org/10.1007/s10527-006-0042-6

[18] Menyaev Yu.A., Zharov V.P. Experience in the use of therapeutic photomatrix equipment. *Biomed. Eng.*, 2006, vol. 40, iss. 3, pp. 144–147. DOI: https://doi.org/10.1007/s10527-006-0064-0

[19] Knuth D.E. Art of computer programming. Vol. 2. Seminumerical algorithms. Addison-Wesley, 1997.

[20] Knuth D.E. Art of computer programming. Vol. 4A. Combinatorial Algorithms. P. 1. Addison-Wesley, 2011.

[21] Kolmogorov A.N., Fomin S.V. Elements of the theory of functions and functional analysis. Dover Publ., 1999.

[22] Deon A.F., Menyaev Yu.A. The complete set simulation of stochastic sequences without repeated and skipped elements. *J. Univers. Comput. Sci.*, 2016, vol. 22, iss. 8, pp. 1023–1047. DOI: https://doi.org/10.3217/jucs-022-08-1023

[23] Deon A.F., Menyaev Yu.A. Parametrical tuning of twisting generators. *J. Comp. Sci.*, 2016, vol. 12, iss. 8, pp. 363–378. DOI: https://doi.org/10.3844/jcssp.2016.363.378

[24] Deon A.F., Menyaev Yu.A. Twister generator of arbitrary uniform sequences. *J. Univers. Comput. Sci.,* 2017, vol. 23, iss. 4, pp. 353–384. DOI: https://doi.org/10.3217/jucs-023-04-0353

[25] Deon A.F., Menyaev Yu.A. Uniform twister plane generator. *J. Comp. Sci.*, 2018, vol. 14, iss. 2, pp. 260–272. DOI: https://doi.org/10.3844/jcssp.2018.260.272

Деон Алексей Федорович — канд. техн. наук, доцент кафедры «Программное обеспечение ЭВМ и информационные технологии» (Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5, стр. 1).

Дмитриев Дмитрий Дмитриевич — канд. техн. наук, доцент кафедры «Технологии приборостроения» (Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5, стр. 1).

Меняев Юлиан Алексеевич — канд. техн. наук, сотрудник Института исследования рака им. Уинтропа Рокфеллера (США, Арканзас AR 72202, Литл-Рок, 4018 W Capitol Ave).

Просьба ссылаться на эту статью следующим образом:

Деон А.Ф., Дмитриев Д.Д., Меняев Ю.А. Вихревой генератор случайных величин Пуассона по технологии кумулятивных частот. *Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение*, 2020, № 1 (130), с. 101–123.
DOI: https://doi.org/10.18698/0236-3933-2020-1-101-123

TWISTER GENERATOR OF POISSON RANDOM NUMBERS WITH THE USE OF CUMULATIVE FREQUENCY TECHNOLOGY

A.F. Deon[1] deonalex@mail.ru
D.D. Dmitriev[1] ddd.1955@gmail.com
Yu.A. Menyaev[2] yamenyaev@uams.edu

[1] **Bauman Moscow State Technical University, Moscow, Russian Federation**
[2] **Winthrop P. Rockefeller Cancer Institute, Little Rock, AR, USA**

Abstract

The widely known generators of Poisson random variables are associated with different modifications of the algorithm based on the convergence in probability of a sequence of uniform random variables to the created stochastic number. However, in some situations, this approach yields different discrete Poisson probability distributions and skipping in the generated numbers. This paper offers a new approach for creating Poisson random variables based on the complete twister generator of uniform random variables, using cumulative frequency technology. The simulation results confirm that probabilistic and frequency distributions of the obtained stochastic numbers completely coincide with the theoretical Poisson distribution. Moreover, combining this new approach with the tuning algorithm of basic twister generation allows for a significant increase in length of the created sequences without using additional RAM of the computer

Keywords

Pseudorandom number generator, stochastic sequences, Poisson distribution, twister generator

Received 15.11.2019
Accepted 25.11.2019

REFERENCES

[1] Feller W. An introduction to probability theory and its applications. Vol. 2. WSE Press, 2008.

[2] Gnedenko B. Theory of probability. CRC Press, 1998.

[3] Zhang H., Li B. Characterizations of discrete compound Poisson distribution. *Commun. Stat. — Theory Methods*, 2016, vol. 45, iss. 22, pp. 6789–6802.
DOI: https://doi.org/10.1080/03610926.2014.901375

[4] Guerriero V. Power low distribution: method of multi-scale inferential statistics. *JMMF*, 2012, vol. 1, no. 1, pp. 21–28.

[5] Arkani M., Khalafi H., Vosoughi N. A flexible multichannel digital random pulse generator based on FPGA. *WJNST*, 2013, vol. 3, no. 4, pp. 109–116.
DOI: https://doi.org/10.4236/wjnst.2013.34019

[6] Rasoanaivo A.N., Horowitz W.A. Medium-induced radiation beyond the Poisson approximation. *J. Phys.: Conf. Ser.*, 2017, vol. 878, art. 012029.
DOI: https://doi.org/10.1088/1742-6596/878/1/012029

[7] Veiga A., Spinelli E. A pulse generator with Poisson-exponential distribution for emulation of radioactive decay events. *Proc. 7th LASCAS*, 2016, pp. 31–34.
DOI: https://doi.org/10.1109/LASCAS.2016.7451002

[8] Kirkpatrick J.M., Young B.M. Poisson statistical methods for the analysis of low-count gamma spectra. *IEEE Trans. Nucl. Sci.*, 2009, vol. 56, iss. 3, pp. 1278–1282.
DOI: https://doi.org/10.1109/TNS.2009.2020516

[9] Marsaglia G., Tsang W.W., Wang J. Fast generation of discrete random variables. *J. Stat. Softw.*, 2004, vol. 11, iss. 3, pp. 1–11. DOI: https://doi.org/10.18637/jss.v011.i03

[10] Kumari S., Valarmathi M., Prince S. Generation of pseudorandom binary sequence using shot noise for optical encryption. *Proc. ICCSP*, 2016, pp. 0119–0122.
DOI: https://doi.org/10.1109/ICCSP.2016.7754537

[11] Hosamo M. A study of the source traffic generator using Poisson distribution for ABR service. *Model. Simul. Eng.*, 2012, vol. 2012, art. 408395.
DOI: https://doi.org/10.1155/2012/408395

[12] Zhang H., Liu Y., Li B. Notes on discrete compound Poisson model with applications to risk theory. *Insur.: Math. Econ.*, 2014, vol. 59, pp. 325–336.
DOI: https://doi.org/10.1016/j.insmatheco.2014.09.012

[13] Shanmugam R. Informatics about fear to report rapes using bumped-up Poisson model. *Amer. J. Biostat.*, 2013, vol. 3, iss. 1, pp. 17–29.
DOI: https://doi.org/10.3844/amjbsp.2013.17.29

[14] Menyaev Yu.A., Nedosekin D.A., Sarimollaoglu M., et al. Optical clearing in photoacoustic flow cytometry. *Biomed. Opt. Express*, 2013, vol. 4, iss. 12, pp. 3030–3041.
DOI: https://doi.org/10.1364/BOE.4.003030

[15] Menyaev Yu.A., Carey K.A., Nedosekin D.A., et al. Preclinical photoacoustic models: application for ultrasensitive single cell malaria diagnosis in large vein and artery. *Biomed. Opt. Express*, 2016, vol. 7, iss. 9, pp. 3643–3658.
DOI: https://doi.org/10.1364/BOE.7.003643

[16] Sitek A., Celler A.M. Limitations of Poisson statistics in describing radioactive decay. *PM*, 2015, vol. 31, iss. 8, pp. 1105–1107. DOI: https://doi.org/10.1016/j.ejmp.2015.08.015

[17] Menyaev Yu.A., Zharov V.P. Experience in development of therapeutic photomatrix equipment. *Biomed. Eng.*, 2006, vol. 40, iss. 2, pp. 57–63.
DOI: https://doi.org/10.1007/s10527-006-0042-6

[18] Menyaev Yu.A., Zharov V.P. Experience in the use of therapeutic photomatrix equipment. *Biomed. Eng.*, 2006, vol. 40, iss. 3, pp. 144–147.
DOI: https://doi.org/10.1007/s10527-006-0064-0

[19] Knuth D.E. Art of computer programming. Vol. 2. Seminumerical algorithms. Addison-Wesley, 1997.

[20] Knuth D.E. Art of computer programming. Vol. 4A. Combinatorial Algorithms. P. 1. Addison-Wesley, 2011.

[21] Kolmogorov A.N., Fomin S.V. Elements of the theory of functions and functional analysis. Dover Publ., 1999.

[22] Deon A.F., Menyaev Yu.A. The complete set simulation of stochastic sequences without repeated and skipped elements. *J. Univers. Comput. Sci.*, 2016, vol. 22, iss. 8, pp. 1023–1047. DOI: https://doi.org/10.3217/jucs-022-08-1023

[23] Deon A.F., Menyaev Yu.A. Parametrical tuning of twisting generators. *J. Comp. Sci.*, 2016, vol. 12, iss. 8, pp. 363–378. DOI: https://doi.org/10.3844/jcssp.2016.363.378

[24] Deon A.F., Menyaev Yu.A. Twister generator of arbitrary uniform sequences. *J. Univers. Comput. Sci.*, 2017, vol. 23, iss. 4, pp. 353–384.
DOI: https://doi.org/10.3217/jucs-023-04-0353

[25] Deon A.F., Menyaev Yu.A. Uniform twister plane generator. *J. Comp. Sci.*, 2018, vol. 14, iss. 2, pp. 260–272. DOI: https://doi.org/10.3844/jcssp.2018.260.272

Deon A.F. — Cand. Sc. (Eng.), Assoc. Professor, Department of Computer Software and Information Technology, Bauman Moscow State Technical University (2-ya Baumanskaya ul. 5, str. 1, Moscow, 105005 Russian Federation).

Dmitriev D.D. — Cand. Sc. (Eng.), Assoc. Professor, Department of Instrumentation Technology, Bauman Moscow State Technical University (2-ya Baumanskaya ul. 5, str. 1, Moscow, 105005 Russian Federation).

Menyaev Yu.A. — Cand. Sc. (Eng.), employer, Winthrop P. Rockefeller Cancer Institute, University of Arkansas for Medical Science (4018 W Capitol Ave, Little Rock, AR 72202 USA).

Please cite this article in English as:

Deon A.F., Dmitriev D.D., Menyaev Yu.A. Twister generator of Poisson random numbers with the use of cumulative frequency technology. *Herald of the Bauman Moscow State Technical University, Series Instrument Engineering*, 2020, no. 1 (130), pp. 101–123 (in Russ.). DOI: https://doi.org/10.18698/0236-3933-2020-1-101-123