УДК 681.3.06 DOI: 10.18698/0236-3933-2019-3-27-45

ВИХРЕВОЙ ГЕНЕРАТОР СТОХАСТИЧЕСКИХ ПЛОСКОСТЕЙ

А.Ф. Деон[1] deonalex@mail.ru
В.А. Онучин[1] onuch-v@yandex.ru
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

Дискретную стохастическую плоскость можно создать с помощью различных алгоритмов генерации случайных величин. Если необходимо, чтобы такая плоскость обладала декартовым свойством полноты, то она должна быть равномерной. Дело в том, что использование концепции бесконтрольной генерации случайных величин может привести к результату низкого качества, поскольку исходные последовательности могут иметь как недостаточную равномерность, так и пропуск случайных величин. Предложен новый подход для создания стохастических декартовых плоскостей по модели полных вихревых последовательностей равномерных случайных величин без пропусков и повторений. Результаты моделирования подтверждают, что получаемые случайные плоскости действительно имеют абсолютную равномерность. Кроме того, комбинирование параметров исходных полных равномерных последовательностей позволяет существенно увеличить число создаваемых плоскостей без использования дополнительной оперативной памяти компьютера

Ключевые слова

Генератор случайных величин, стохастические последовательности, вихревые генераторы, стохастические плоскости

Поступила 29.01.2019

Введение. Генераторы равномерных плоскостей (*Random Plane, RP*) реализуют случайный процесс при создании точек, распределенных на *N*-мерной плоскости. В этом исследовании рассмотрены только двумерные плоскости. Плоскости других дискретных размерностей имеют те же начальные свойства. Каждая координата *RP* генерируемой точки на плоскости может принадлежать своему случайному полю (*Random Field*, *RF*). В современной литературе рассматривают условные [1, 2], марков-

ские [3, 4], гауссовы [5], равномерные [6, 7] и другие *RF* [8–11]. В прикладных областях *RP*-генераторы плоскостей применяют при анализе графических изображений [12], в звуковых системах [13], в рекламных приложениях и т. д., также их активно используют в плоской реализации метода Монте-Карло [14, 15], в факториальных разработках [16], в обучающих системах [17] вплоть до биологической инженерии [18, 19].

Принципы перечисленных исследований базируются на концепции случайных плоскостей, в которых особенности декартовой плоскости удовлетворяют следующим свойствам: 1) процесс генерации должен обеспечивать уникальность (без повторений) каждой точки на плоскости, 2) процесс генерации должен сохранять полноту (без пропусков) для всех создаваемых точек. Эти условия являются «естественным фильтром» при подборе генераторов равномерных случайных величин.

Рассмотрим этот вопрос подробно. Для этого используем вихревой генератор [10, 20], который может создавать полные равномерные последовательности произвольного размера. Программный код для моделирования сетки дискретной декартовой плоскости $U \times V$ приведен ниже. Алгоритм программы независимо генерирует целые случайные величины вдоль независимых осей U и V. Матрица A является индикатором дискретной плоскости. Ее клетки $a[u, v] \in A$ с индексами u и v соответствуют координатам дискретных точек $< u, v > \in U \times V$. Значение клетки $a[u, v] \in A$ показывает число попыток независимого создания соответствующей точки $< u, v >$ на генерируемой плоскости. Без потери общности в целях наглядного представления результата зададим число дискретных координат множествами $U = V = \{0, 1, 2, 3, 4, 5, 6, 7\}$, что в двоичном представлении соответствует длине $w = 3$ бита для каждой координаты. Для тестирующего моделирования выберем вихревой генератор *nsDeonYuliTwist32D* с константами $a = 5$, $c = 1$ [10, 20], поскольку другой современный генератор *MT19937* [19, 21–24] не имеет абсолютной полноты. Начальные значения вихревых последовательностей возьмем как $x_0 = 1 \in U$ и $x_0 = 4 \in V$. Возможен любой другой выбор параметров генерации, но суть и общность получаемых результатов не меняется. Программные имена *P040101* и *cP040101* выбраны произвольно. Язык программирования C# в среде *Microsoft Visual Studio*. Применение других диалектов исторического C (*Win32*) или C++ (*CLR*) дает те же результаты.

```
using nsDeonYuliTwist32D;   // генератор целых случайных величин
namespace P040101
{  class cP040101
   {  static void Main(string[] args)
```

```
      {  uint w = 3;          // битовая длина случайных величин
         cDeonYuliTwist32D GU = new cDeonYuliTwist32D();
         GU.x0 = 1;               // начало последовательности U
         GU.w = w;            // битовая длина случайных величин
         GU.Start();                    // старт генератора GU
         cDeonYuliTwist32D GV = new cDeonYuliTwist32D();
         GV.x0 = 4;               // начало последовательности V
         GV.w = w;            // битовая длина случайных величин
         GV.Start();                    // старт генератора GV
         int N = 1 << (int)w; // длина последовательностей U и V
         Console.WriteLine("w = {0}  N = {1}", w, N);
         uint[] U = new uint[N];         // последовательность U
         uint[] V = new uint[N];         // последовательность V
         int[,] A = new int[N,N];          // матрица результата
         for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++) A[i, j] = 0;
         for (int i = 0; i < N; i++)             // одна из осей
         {  Console.Write("i = {0,3} |   ", i);
            for (int j = 0; j < N; j++)           // другая ось
            {  uint u = GU.Next();       // случайная величина u
               U[j] = u;       // случайная последовательность U
               uint v = GV.Next();       // случайная величина v
               V[j] = v;       // случайная последовательность V
               A[u, v]++;       // счетчик генерации точки <u,v>
            }
            Console.Write("U = ");
            for (int m = 0; m < N; m++)
               Console.Write("{0,4}", U[m]);
            Console.WriteLine();
            Console.Write("        |   V = ");
            for (int m = 0; m < N; m++)
               Console.Write("{0,4}", V[m]);
            Console.WriteLine();
         }
         Console.WriteLine("Matrix A");
         for (int i = 0; i < N; i++)
         {  for (int j = 0; j < N; j++)
               Console.Write("{0,4}", A[i, j]);
            Console.WriteLine();
         }
         Console.ReadKey();              // просмотр результата
      }
   }
}
```

После выполнения этой программы на мониторе появляется следующий листинг:

```
w = 3   N = 8
i =    0 |   U =   1   6   7   4   5   2   3   0
         |   V =   4   5   2   3   0   1   6   7
i =    1 |   U =   3   5   7   1   2   4   6   0
         |   V =   1   2   4   6   0   3   5   7
i =    2 |   U =   7   3   6   2   5   1   4   0
```

```
          |   V =   2   5   1   4   0   7   3   6
i =     3 |   U =   6   7   4   5   2   3   0   1
          |   V =   5   2   3   0   1   6   7   4
i =     4 |   U =   5   7   1   2   4   6   0   3
          |   V =   2   4   6   0   3   5   7   1
i =     5 |   U =   3   6   2   5   1   4   0   7
          |   V =   5   1   4   0   7   3   6   2
i =     6 |   U =   7   4   5   2   3   0   1   6
          |   V =   2   3   0   1   6   7   4   5
i =     7 |   U =   7   1   2   4   6   0   3   5
          |   V =   4   6   0   3   5   7   1   2
Matrix A
   0   0   0   0   0   0   2   6
   0   0   0   0   3   0   3   2
   3   3   0   0   2   0   0   0
   0   3   0   0   0   2   3   0
   0   0   0   8   0   0   0   0
   5   0   3   0   0   0   0   0
   0   2   0   0   0   6   0   0
   0   0   5   0   3   0   0   0
```

Наличие в матрице A клеток со значениями, отличными от единицы, показывает, что независимая генерация координат точек на плоскости не обеспечивает равномерное распределение случайных точек $< u, v >$ на сетке дискретной плоскости. Некоторые точки отсутствуют вовсе (значения 0), другие — присутствуют несколько раз (значения > 1).

Этот пример показывает, что независимая генерация координат точек не обеспечивает выполнение декартовых свойств и, следовательно, не гарантирует получение равномерной стохастической плоскости. Следовательно, *цель работы* — поиск решения для генерации равномерных стохастических плоскостей, обладающих декартовым свойством одиночного присутствия случайных точек в узлах дискретной сетки.

Теория. Любая плоскость определяет множество принадлежащих ей точек. Проекции этих точек на координатные оси задают их координаты. В такой последовательности определений первичным является плоскость, вторичным — координаты ее точек. Эти плоскости обладают декартовыми свойствами: каждая точка на плоскости уникальна (без повторений), множество точек является полным относительно всех своих точек (без пропусков).

Если первичным принять задание произвольных парных координат в заданной координатной системе, то последовательное перечисление всех таких пар может характеризовать плоскость лишь в тех случаях, когда на ней выполняются декартовые свойства. В таком варианте используется алгоритмический подход, в котором применяется интуитивное понятие последовательного перечисления, поскольку алгоритм — последователь-

ность выполнимых действий. Стохастический алгоритм имеет стохастическую последовательность действий. Плоскость является стохастической, если указание всех ее точек выполняется по стохастическому алгоритму с соблюдением декартовых свойств уникальности и полноты.

Выполняя генерацию координат точек плоскости по независимым алгоритмам вдоль каждой из координатных осей, можно обеспечить лишь независимую случайность, но обеспечить полную декартовую стохастичность получаемой плоскости не всегда представляется возможным. Это подтвердил результат моделирования, полученный выше. Однако ситуацию можно исправить, если генерацию парных координат точек стохастической плоскости выполнять по стохастическим трекам.

Для того чтобы наглядно представить вихревой трек, возьмем минимальные последовательности случайных величин x длиной $w = 2$ бита, $x \in \{0,1,2,3\} = \{00_2, 01_2, 10_2, 11_2\}$. В этом случае каждая полная последовательность содержит $N = 2^w = 2^2 = 4$ элемента. Без потери общности предположим, что вихревые параметры [10, 23] имеют значения $a = 1 \in \overline{[1, N-1]}$ и $a = 3 \in \overline{[1, N-1]}$. Всего возможны четыре вихревые последовательности: $<0,3,2,1>$, $<1,0,3,2>$, $<2,1,0,3>$, $<3,2,1,0>$. Если в качестве начального значения выбрать случайную величину $x_0 = 1$, то генератор с обозначением *GU* создает вихревую последовательность $U =< 1,0,3,2 >$. Из этого следует, что начальная случайная вершина будет расположена на вертикальной части сетки с горизонтальной дискретной отметкой 1 на оси U (рисунок, *а*).

Если второй независимый генератор *GV* использует начальную случайную величину $x_0 = 3$, то создается последовательность $V =< 3,2,1,0 >$. Тогда вторая координата имеет значение 3 для начальной случайной точки $<1,3>$. Координаты следующей вершины $<0,2>$. Обе полученные вершины связывает дуга, образуя начало случайного трека. Затем на этом треке появляется вершина $<3,1>$. Завершает случайный трек вершина с координатами $<2,0>$. Вид этого трека показан на рисунке, *а*.

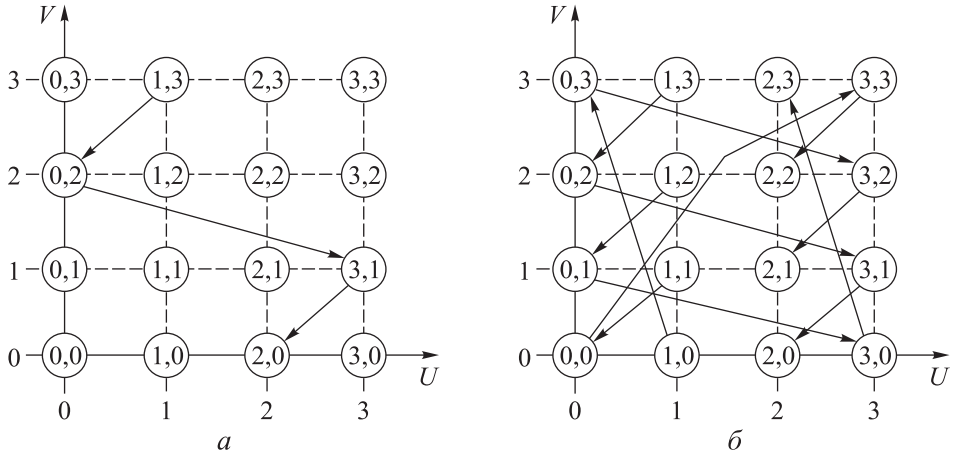
Относительно случайной последовательности $<1,0,3,2>$ вдоль оси U возможны четыре последовательности вдоль оси *V*:

1) $U =< 1,0,3,2 >$
   $V =< 3,2,1,0 >$;
2) $U =< 1,0,3,2 >$

$V =< 2,1,0,3 >;$

3) $U =< 1,0,3,2 >$
$V =< 1,0,3,2 >;$

4) $U =< 1,0,3,2 >$
$V =< 0,3,2,1 >.$



Вид начального случайного трека (*а*) и вид треков случайных последовательностей на осях U и V (*б*)

Эти последовательности V можно интерпретировать как левый кольцевой сдвиг исходной последовательности $< 3,2,1,0 >$. Четыре трека взаимодействия пар последовательностей на осях U и V показаны на рисунке, *б*. Эти треки обеспечивают однократную генерацию каждой точки на случайной плоскости.

Аналогичные испытания создания стохастических плоскостей с произвольной битовой длиной w случайных величин подтверждает равномерность получаемых случайных плоскостей.

Простой вихревой генератор. В предыдущем примере использованы две стохастические последовательности U и V в качестве координатных массивов. Такие массивы можно создать с помощью вихревого генератора *nsDeonYuliTwist32D* [10, 20]. Однако генератор *nsDeonYuliTwist32D* не предусматривает возможности тривиальных операций с вихревыми последовательностями. Чтобы исключить этот недостаток, необходимо иметь простой вихревой генератор с элементарными действиями.

Класс *cDeonYuliSTwist32D* представлен ниже. Буква *S* в имени соответствует слову *Simple* (простой). Этот класс обеспечивает элементарные операции над вихревыми последовательностями. Прообразом класса *cDeonYuliSTwist32D* служит класс *cDeonYuliTwist32D*, если исключить в

последнем возможности автоматической настройки конгруэнтных параметров *a* и *c*. Пример применения класса *cDeonYuliSTwist32D* будет рассмотрен далее в программе *P040302* при генерации вихревых плоскостей.

```
namespace nsDeonYuliSTwist32D
{  class cDeonYuliSTwist32D
   {  public uint w = 16U;       // битовая длина числа
      public uint N1 = 0U;       // максимальное число
      public uint x0 = 1U;       // начало последовательности
      uint xB = 1U;              // начало очередного вихря
      public uint xG = 0U;       // созданная случайная величина
      uint xL = 0U, xR = 1U;     // парные величины
      public uint a = 5U;        // конгруэнтная константа a
      public uint c = 1U;         // конгруэнтная константа c
      public uint maskW = 0U;     // маска числа
      public uint maskU = 0U;    // маска старшего бита
      public uint maskT = 0U;    // вихревые биты
      public uint nW = 0U;       // номер парного вихря в w
//-------------------------------------------------------------
      public cDeonYuliSTwist32D()
      {  N1 = 0xFFFFFFFF >> (32 - (int)w); // максимальное число
      }
//-------------------------------------------------------------
      public void StartCong(uint sxB)
      {  xB = x0;
         uint sxBe = sxB & maskW;
         for (int i = 0; i < sxBe; i++)
            xB = (a * xB + c) & maskW;     // сдвинутое начало
         xR = xB; // заготовка начала вихря
      }
//-------------------------------------------------------------
      public uint NextCong()
      {  xL = xR; // начало пары
         xR = (a * xL + c) & maskW; // окончание пары
         xG = xL; // сгенерированная величина
         return xG;
      }
//-------------------------------------------------------------
      public void RepeatCong()
      {  xR = xB;                              // повторить трек
      }
//-------------------------------------------------------------
      public void ShiftCong()
      {  xB = (a * xB + c) & maskW; // сдвинутое начало
         xR = xB; // заготовка начала вихря
      }
//-------------------------------------------------------------
      public void StartTwist(uint snW)
      {  nW = (uint)snW; // величина битового сдвига
         maskT = maskU;      //старшая 1 маски вихря
         for (int m = 1; m < nW; m++)
            maskT |= maskU >> m;    // маска вихря
         xL = xB;     // заготовка начала вихря
         xR = (a * xL + c) & maskW; // окончание пары xL,xR
```

```
      }
//-------------------------------------------------------------
      public uint NextTwist()
      {  uint g = (xR & maskT) >> (int)(w - nW);   // старшие
         xG = ((xL << (int)nW) & maskW) | g;       // младшие
         xL = xR;     // начало следующей пары
         xR = (a * xL + c) & maskW; // окончание пары xL,xR
         return xG;   // вихрь пары
      }
//-------------------------------------------------------------
      public void RepeatTwist()
      {  xL = xB; // заготовка начала вихря
         xR = (a * xL + c) & maskW; // окончание пары xL,xR
      }
//-------------------------------------------------------------
      public void ShiftTwist()
      {  xB = (a * xB + c) & maskW; // сдвинутое начало
         xL = xB;      // заготовка начала вихря
         xR = (a * xL + c) & maskW; // окончание пары xL,xR
      }
//-------------------------------------------------------------
      public void Start()
      {  N1 = 0xFFFFFFFF >> (32 - (int)w); // максимальное число
         maskW = 0xFFFFFFFF >> (32 - (int)w);  // маска числа
         maskU = 1U << ((int)w - 1);// маска старшего бита
         maskT = maskU;      // первый вихревой бит
         DeonYuli_PlusA();   // значение a
         DeonYuli_SetC(); // значение c
         x0 &= maskW;     // начало последовательности
         StartCong(0);   // для конгруэнтной генерации
      }
//-------------------------------------------------------------
      public void TimeStart()
      {  x0 = (uint)DateTie.Now.Millisecond;   // миллисекунды
         Start();                          // старт генератора
      }
//-------------------------------------------------------------
      public void SetW(uint sw)
      {  w = (uint)Math.Abs(sw);    // битовая длина числа
         DeonYuli_SetW();
      }
//-------------------------------------------------------------
      public void SetW(int sw)
      {  w = (uint)Math.Abs(sw);    // битовая длина числа
         DeonYuli_SetW();
      }
//-------------------------------------------------------------
      public void DeonYuli_SetW()
      {  if (w < 3U) w = 3U;     // минимальная длина
         else if (w > 32U) w = 32U;     // максимальная длина
         N1 = 0xFFFFFFFF >> (32 - (int)w); // максимальное число
         x0 = N1 / 7U;       // начало последовательности
      }
//-------------------------------------------------------------
```

```
      public void SetA(double sa)
      {  double ad = Math.Abs(sa);
         if (ad > 1.0) ad = 1.0;
         a = (uint)(N1 * ad);       // относительная установка a
      }
//-------------------------------------------------------------
      public void SetA(uint sa)
      {  a = (uint)Math.Abs(sa);
         if (a < 1) a = 1;          // минимальное значение
         if (a > N1) a = N1;        // максимальное значение
      }
//-------------------------------------------------------------

      void DeonYuli_PlusA()
      {  if (a < 1U) { a = 1U; return; }
         uint z = a;  // нижняя граница для a
         for (uint i = 0U; i < 3U; i++)
            if (a % 4U != 0U) a--;  // условие равномерности
            else break;
         a++;  // правильное значение константы a
         if (a < z) a += 4U; // справа от нижней границы
      if (a >= N1 - 1) a -= 4U;     // слева от верхней границы
      }
//-------------------------------------------------------------
      public void SetC(double sc)
      {  double cd = Math.Abs(sc);
         if (cd > 1.0) cd = 1.0;
         c = (uint)(N1 * cd);       // относительная установка c
      }
//-------------------------------------------------------------
      public void SetC(uint sc)
      {  c = (uint)Math.Abs(sc);
         if (c < 1) a = 1;          // минимальное значение
         if (c > N1) c = N1;        // максимальное значение
      }
//-------------------------------------------------------------
      void DeonYuli_SetC()
      {  if (c % 2U == 0U) c += 1;  // только нечетное c
         if (c > N1) c = N1;       // максимальное значение
      }
//-------------------------------------------------------------
      public void SetX0(double sx)
      {  double xd = Math.Abs(sx);
         if (xd > 1.0) xd = 1.0;
         x0 = (uint)(N1 * xd);   // начало последовательности
      }
//-------------------------------------------------------------
      public void SetX0(int sx)
      {  x0 = (uint)sx;          // начало последовательности
      }
//=============================================================
   }
}
```

Вихревой генератор равномерной плоскости. При построении полного генератора равномерной вихревой плоскости используем два генератора *GU* и *GV* представленного выше инструментального класса *nsDeonYuliSTwist32D*. Это позволяет организовать класс *cDeonYuliPlaneTwist32D* для генерации всех точек на квадратной сетке вихревой плоскости. По умолчанию начальным треком принимается диагональ точек сетки слева направо и снизу вверх, хотя это можно изменить, задав независимые начала для внутренних генераторов *GU* и *GV*. Пример применения класса *cDeonYuliPlaneTwist32D* будет рассмотрен в программе *P040302* при генерации вихревой плоскости.

```
using nsDeonYuliSTwist32D;         // простой вихревой генератор
namespace nsDeonYuliPlaneTwist32D
{  class cDeonYuliPlaneTwist32D
   {  public uint w = 16;        // битовая длина равномерных величин
      public uint N1 = 0;       // максимальное число в треке
      public uint a = 5U;       // конгруэнтная константа a
      public uint c = 1U;       // конгруэнтная константа c
      public uint x0 = 1U;      // константа начала трека
      public cDeonYuliSTwist32D GU;               // генератор 1
      public cDeonYuliSTwist32D GV;               // генератор 2
      public uint nWU = 0;        // номер вихревого сдвига в GU
      public uint nRU = 0;       // номер кольцевого сдвига в GU
      public uint nWV = 0;       // номер вихревого сдвига в GV
      public uint nRV = 0;       // номер кольцевого сдвига в GV
      public uint nG = 0;        // номер элементов в треках GU и GV
//-------------------------------------------------------------
      public cDeonYuliPlaneTwist32D()
      {  GU = new cDeonYuliSTwist32D();   // создать генератор 1
         GV = new cDeonYuliSTwist32D();   // создать генератор 2
      }
//-------------------------------------------------------------
      public void SetW(int sw)
      {  w = (uint)sw;       // битовая длина случайной величины
       DeonYuli_SetWN1ACX();
      }
//-------------------------------------------------------------
      public void SetW(uint sw)
      {  w = sw;             // битовая длина случайной величины
       DeonYuli_SetWN1ACX();
      }
//-------------------------------------------------------------
      void DeonYuli_SetWN1ACX()
      {  N1 = 0xFFFFFFFF >> (32 - (int)w);    // макс-число
         a = (uint)((double)N1 * 0.39);      // конг-константа a
         c = a / 2;                  // конгруэнтная константа c
         x0 = N1 / 2U;               // константа начала трека
       }
//-------------------------------------------------------------
      void DeonYuli_SetN1ACX()
```

```
      {  GU.w = w;       // битовая длина случайных величин в GU
         GU.a = a;            // конгруэнтная константа a для GU
         GU.c = c;            // конгруэнтная константа c для GU
         GU.x0 = x0;           // начало последовательности в GU
         GV.w = w;       // битовая длина случайных величин в GV
         GV.a = a;            // конгруэнтная константа a для GV
         GV.c = c;            // конгруэнтная константа c для GV
         GV.x0 = x0;           // начало последовательности в GV
      }
//-------------------------------------------------------------
      public void Start()
      {  DeonYuli_SetN1ACX();          // конгруэнтные константы
         GU.Start();                      // старт генератора GU
         GV.Start();                      // старт генератора GV
         a = GU.a;                   // конгруэнтная константа a
         c = GU.c;                   // конгруэнтная константа c
         x0 = GU.x0;         // начало последовательностей U и V
         nWU = 0;      // номер вихревого сдвига в генераторе GU
         nRU = 0;     // номер кольцевого сдвига в генераторе GU
         nWV = 0;      // номер вихревого сдвига в генераторе GV
         nRV = 0;     // номер кольцевого сдвига в генераторе GV
         nG = 0;             // номер элементов в треках GU и GV
         GU.StartCong(0);       // для конгруэнтной генерации GU
         GV.StartCong(0);       // для конгруэнтной генерации GV
      }
//-------------------------------------------------------------
      public void Next(ref uint u, ref uint v)
      {  if (nWU == 0) GU.NextCong(); else GU.NextTwist();
         if (nWV == 0) GV.NextCong(); else GV.NextTwist();
         u = GU.xG;                    // координаты точки <u,v>
         v = GV.xG;
         if (nG < N1) { nG++; return; }          // внутри трека
         nG = 0;                      // начало очередного трека
         if (DeonYuli_RingCongGV()) return;         // внутри GV
         if (DeonYuli_RingTwistGV()) return;        // внутри GV
         if (DeonYuli_RingCongGU()) return;         // внутри GU
         DeonYuli_RingTwistGU();                    // внутри GU
      }
//-------------------------------------------------------------
      bool DeonYuli_RingCongGV()
      {  if (nWV != 0) return false; // нет конгруэнтного кольца
         if (nWU == 0) GU.RepeatCong(); else GU.RepeatTwist();
         if (nRV < N1)  // возможность конгруэнтного кольца в GV
         {  GV.ShiftCong();           // конгруэнтный сдвиг в GV
            nRV++;                    // номер следующего кольца
            return true;      // внутри конгруэнтного трека в GV
         }
         nRV = 0;                       // начальное кольцо в GV
         nWV = 1;                           // первый вихрь в GV
         GV.StartTwist(nWV);                 // старт вихря в GV
         return true;                               // внутри GV
      }
//-------------------------------------------------------------
```

```
      bool DeonYuli_RingTwistGV()
      {  if (nWV == 0) return false;     // нет вихревого кольца
         if (nWU == 0) GU.RepeatCong(); else GU.RepeatTwist();
         if (nRV < N1)      // возможность вихревого кольца в GV
         {  GV.ShiftTwist();              // вихревой сдвиг в GV
            nRV++;                     // номер следующего кольца
            return true;          // внутри вихревого трека в GV
         }
         nRV = 0;                             // новое кольцо в GV
         if (nWV < w - 1)                     // продолжить вихри
         {  nWV++;               // следующий битовый вихрь в GV
            GV.StartTwist(nWV);               // старт вихря в GV
            return true;         // внутри вихревого режима в GV
         }
         nRV = 0;                             // новое кольцо в GV
         nWV = 0;          // конгруэнтное начало (вихрь 0) в GV
         GV.StartCong(nWV);              // начало генератора GV
         return false;     // оба кольца R2 и W2 прокручены в GV
      }
//-------------------------------------------------------------
      bool DeonYuli_RingCongGU()
      {  if (nWU != 0) return false; // нет конгруэнтного кольца
         if (nRU < N1)  // возможность конгруэнтного кольца в GU
         {  GU.ShiftCong();           // конгруэнтный сдвиг в GU
            nRU++;                    // номер следующего кольца
            return true;      // внутри конгруэнтного трека в GU
         }
         nRU = 0;                        // начальное кольцо в GU
         nWU = 1;                            // первый вихрь в GU
         GU.StartTwist(nWU);                  // старт вихря в GU
         return true;                                 // внутри GU
      }
//-------------------------------------------------------------
      bool DeonYuli_RingTwistGU()
      {  if (nWU == 0) return false;     // нет вихревого кольца
         if (nRU < N1)      // возможность вихревого кольца в GU
         {  GU.ShiftTwist();              // вихревой сдвиг в GU
            nRU++;                     // номер следующего кольца
            return true;          // внутри вихревого трека в GU
         }
         nRU = 0;                             // новое кольцо в GU
         if (nWU < w - 1)                     // продолжить вихри
         {  nWU++;               // следующий битовый вихрь в GU
            GU.StartTwist(nWU);               // старт вихря в GU
            return true;         // внутри вихревого режима в GU
         }
         nRU = 0;                             // новое кольцо в GU
         nWU = 0;          // конгруэнтное начало (вихрь 0) в GU
         GU.StartCong(nWU);              // начало генератора GU
         return false;    // все плоскости созданы; общее начало
      }
//=============================================================
   }
}
```

Для того чтобы проверить работу представленного генератора *nsDeonYuliPlaneTwist32D*, используем приведенный ниже программный код *P040302*, в котором генерируются точки начальной вихревой плоскости случайных величин длиной $w = 3$ бита. По умолчанию начальный трек генерации — диагональ сетки слева направо и снизу вверх. Равномерность точек на плоскости должна будет подтвердить матрица A, в которой значения клеток являются счетчиками генерации соответствующих точек $< u, v >$ на *RP*. Программные имена *P040302* и *cP040302* выбраны произвольно.

```
using nsDeonYuliPlaneTwist32D;   // генератор вихревой плоскости
namespace P040302
{  class cP040302
   {  static void Main(string[] args)
      {  cDeonYuliPlaneTwist32D TP =
                                 new cDeonYuliPlaneTwist32D();
         int w = 3;           // битовая длина случайных величин
         int N = 1 << w;                          // длина трека
         TP.SetW(w);
         TP.Start();                           // старт генератора
         Console.WriteLine("w = {0}   N = {1}", w, N);
         Console.WriteLine("a = {0}   c = {1}   x0 = {2}",
                           TP.a, TP.c, TP.x0);
         uint[] u = new uint[N];  // u-координаты точек на треке
         uint[] v = new uint[N];  // v-координаты точек на треке
         int[,] A = new int[N, N];         // матрица результата
         for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++) A[i, j] = 0;
         uint uu = 0;   // рабочие координаты точки на плоскости
         uint vv = 0;
         for (int i = 0; i < N; i++)//номера треков на плоскости
         {  Console.Write("i = {0,4}   ", i);
            Console.Write(" nWU = {0,3}", TP.nWU);
            Console.Write(" nRU = {0,3}", TP.nRU);
            Console.Write(" nWV = {0,3}", TP.nWV);
            Console.Write(" nRV = {0,3}", TP.nRV);
            Console.WriteLine();
            for (int j = 0; j < N; j++)
            {  TP.Next(ref uu, ref vv);        // точка на сетке
               u[j] = uu;
               v[j] = vv;
               A[uu, vv]++;           // счетчик генерации <u,v>
            }
            Console.Write("              U = ");
            for (int j = 0; j < N; j++)
               Console.Write("{0,4}", u[j]);
            Console.WriteLine();
            Console.Write("              V = ");
            for (int j = 0; j < N; j++)
               Console.Write("{0,4}", v[j]);
```

```
            Console.WriteLine();
         }
         Console.WriteLine("Matrix A");
         for (int i = 0; i < N; i++)
         {  for (int j = 0; j < N; j++)
               Console.Write("{0,4}", A[i, j]);
            Console.WriteLine();
         }
         Console.ReadKey();                // просмотр результата
      }
   }
}
```

После выполнения программы *P040302* на мониторе появляется следующий листинг (пропущенные строки обозначены прочерком):

```
w = 3   N = 8
a = 5   c = 1   x0 = 3
i =    0    nWU =   0 nRU =   0 nWV =   0 nRV = 0
              U =    3   0   1   6   7   4   5   2
              V =    3   0   1   6   7   4   5   2
i =    1    nWU =   0 nRU =   0 nWV =   0 nRV = 1
              U =    3   0   1   6   7   4   5   2
              V =    0   1   6   7   4   5   2   3
i =    2    nWU =   0 nRU =   0 nWV =   0 nRV = 2
              U =    3   0   1   6   7   4   5   2
              V =    1   6   7   4   5   2   3   0
- - - - -
i =    7    nWU =   0 nRU =   0 nWV =   0 nRV = 1
              U =    3   0   1   6   7   4   5   2
              V =    2   3   0   1   6   7   4   5
Matrix A
1   1   1   1   1   1   1   1
1   1   1   1   1   1   1   1
1   1   1   1   1   1   1   1
1   1   1   1   1   1   1   1
1   1   1   1   1   1   1   1
1   1   1   1   1   1   1   1
1   1   1   1   1   1   1   1
1   1   1   1   1   1   1   1
```

В этом листинге индикатор *nWU* показывает номер вихря на оси *U*, индикатор *nRU* — номер кольцевого сдвига на оси *U*. Значения $NWU = 0$, $NRU = 0$ соответствуют конгруэнтной исходной последовательности на оси *U*. Аналогичные значения индикаторов *NWV*, *nRV* показывают кольцевой сдвиг исходной последовательности на оси *V*.

Листинг единичной матрицы *A* подтверждает, что каждая случайная точка $< u, v >$ создается один раз. Такие же результаты верны для других $w \le 32$, что соответствует концепции равномерной декартовой стохастической плоскости.

Заключение. Анализ исходного материала показывает, что алгоритмы используемых генераторов равномерных плоскостей не учитывают возможности абсолютно равномерных последовательностей случайных величин. Такие технологии генерации не гарантируют абсолютную равномерность полных случайных плоскостей, в результате чего получаемые плоскости имеют низкое качество. Предлагаемая новая технология стохастических треков использует алгоритмы декомпозиции простейших операций абсолютно полных вихревых последовательностей. Это обеспечивает полноту и уникальность всех случайных величин на сетке декартовой плоскости. Выполненные тесты подтверждают абсолютное равномерное распределение получаемых случайных точечных величин на стохастической плоскости.

ЛИТЕРАТУРА

[1] Sutton C., McCallum A. An introduction to conditional random fields. Now Publ. Inc., 2012.

[2] Quattoni A., Collins M., Darrell T. Conditional random fields for object recognition. *Proc. NIPS'04*, 2004. URL: https://papers.nips.cc/paper/2652-conditional-random-fields-for-object-recognition.pdf

[3] Bekkerman R., Sahami M., Learned-Miller E. Combinatorial Markov random fields. In: Fürnkranz J., Scheffer T., Spiliopoulou M. (eds). Machine Learning: ECML 2006. ECML 2006. *Lecture Notes in Computer Science*, vol. 4212, 2006. Berlin, Heidelberg, Springer, pp. 30–41. DOI: https://doi.org/10.1007/11871842_8

[4] Sarawagi S., Cohen W.W. Semi-Markov conditional random fields for information extraction. *Proc. NIPS'04*, 2004. URL: https://papers.nips.cc/paper/2648-semi-markov-conditional-random-fields-for-information-extraction.pdf

[5] Rimstad K., Omre H. Skew-Gaussian random fields. *Spat. Stat.*, vol. 10, no. 11, pp. 43–62. DOI: 10.1016/j.spasta.2014.08.001

[6] Deon A.F., Menyaev Y.A. Uniform twister plane generator. *J. Comput. Sci.*, 2018, vol. 14, iss. 2, pp. 260–272. DOI: 10.3844/jcssp.2018.260.272

[7] Xiao Y. Uniform modulus of continuity of random fields. *Monatsh. Math.*, 2010, vol. 159, iss. 1-2, pp. 163–184. DOI: 10.1007/s00605-009-0133-z

[8] Deon A., Menyaev Y. 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: 10.3217/jucs-022-08-1023

[9] Deon A., Menyaev Y. Parametrical tuning of twisting generators. *J. Comput. Sci.*, 2016, vol. 12, iss. 8, pp. 363–378. DOI: 10.3844/jcssp.2016.363.378

[10] Deon A.F., Menyaev Y.A. Twister generator of arbitrary uniform sequences. *JUCS*, 2017, vol. 23, iss. 4, pp. 353–384. DOI: 10.3217/jucs-023-04-0353

[11] Qi Y., Szummer M., Minka T.P. Bayesian conditional random fields. *Proc. AISTATS'05*, 2005, pp. 269–276.

[12] Kumar S., Hebert M. Discriminative random fields: a discriminative framework for contextual interaction in classification. *Proc. ICCV'03*, 2003, pp. 1150–1157. DOI: 10.1109/ICCV.2003.1238478

[13] Sung Y., Jurafsky D. Hidden conditional random fields for phone recognition. *Proc. ASRU'09*, 2009, pp. 107–112. DOI: 10.1109/ASRU.2009.5373329

[14] Spanos P.D., Zeldin B.A. Monte Carlo treatment of random fields: a broad perspective. *Appl. Mech. Rev.*, 1998, vol. 51, iss. 3, pp. 219–237. DOI: 10.1115/1.3098999

[15] Newman M.E.J., Barkema G.T. Monte Carlo study of the random-field Ising model. *Phys. Rev. E*, 1996, vol. 53, iss. 1, pp. 393–404. DOI: 10.1103/PhysRevE.53.393

[16] Kim J., Zabih R. Factorial Markov random fields. In: Heyden A., Sparr G., Nielsen M., Johansen P. (eds). Computer Vision — ECCV 2002. ECCV 2002. *Lecture Notes in Computer Science*, vol. 2352. Berlin, Heidelberg, Springer, 2002, pp. 321–334. DOI: https://doi.org/10.1007/3-540-47977-5_21

[17] Sha F., Pereira F. Shallow parsing with conditional random fields. *Proc. NAACL'03*, 2003, vol. 1, pp. 134–141. DOI: 10.3115/1073445.1073473

[18] Menyaev Y.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: 10.1364/BOE.7.003643

[19] Matsumoto M., Nishimura T. Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator. *TOMACS*, 1998, vol. 8, iss. 1, pp. 3–30. DOI: 10.1145/272991.272995

[20] Деон А.Ф., Меняев Ю.А. Генератор равномерных вихревых последовательностей целых случайных величин без запоминающего массива. *Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение*, 2018, № 3, с. 51–69. DOI: 10.18698/0236-3933-2018-3-51-69

[21] Matsumoto M., Wada I., Kuramoto A., et al. Common defects in initialization of pseudorandom number generators. *TOMACS*, 2007, vol. 17, iss. 4, art. 15. DOI: 10.1145/1276927.1276928

[22] Saito M., Matsumoto M. SIMD-oriented fast Mersenne twister: a 128-bit pseudorandom number generator. In: Keller A., Heinrich S., Niederreiter H. (eds). *Monte Carlo and Quasi-Monte Carlo Methods 2006*. Berlin, Heidelberg, Springer, 2008, pp. 607–622. DOI: https://doi.org/10.1007/978-3-540-74496-2_36

[23] Деон А.Ф., Меняев Ю.А. Полное факториальное моделирование равномерных последовательностей целых случайных величин. *Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение*, 2017, № 5, с. 132–149. DOI: 10.18698/0236-3933-2017-5-132-149

[24] Деон А.Ф., Меняев Ю.А. Генератор равномерных случайных величин по технологии полного вихревого массива. *Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение*, 2017, № 2, с. 86–110. DOI: 10.18698/0236-3933-2017-2-86-110

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).

Онучин Вадим Александрович — аспирант кафедры «Радиоэлектронные системы и устройства» МГТУ им. Н.Э. Баумана (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., Онучин В.А., Меняев Ю.А. Вихревой генератор стохастических плоскостей. *Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение*, 2019, № 3, с. 27–45. DOI: 10.18698/0236-3933-2019-3-27-45

TWISTER GENERATOR OF STOCHASTIC PLANES

A.F. Deon[1] deonalex@mail.ru
V.A. Onuchin[1] onuch-v@yandex.ru
Yu.A. Menyaev[2] yamenyaev@uams.edu

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

Abstract

Various pseudorandom number generation algorithms may be used to create a discrete stochastic plane. If a Cartesian completeness property is required of the plane, it must be uniform. The point is, employing the concept of uncontrolled random number generation may yield low-quality results, since original sequences may omit random numbers or not be sufficiently uniform. We present a novel approach for generating stochastic Cartesian planes according to the model of complete twister sequences featuring uniform random numbers without omissions or repetitions. Simulation results confirm that the random planes obtained are indeed perfectly uniform. Moreover, recombining the original complete uniform sequence parameters allows the number of planes created to be significantly increased without using any extra random access memory

Keywords

Pseudorandom number generator, stochastic sequences, Mersenne Twister generators, stochastic planes

Received 29.01.2019

REFERENCES

[1] Sutton C., McCallum A. An introduction to conditional random fields. Now Publ. Inc., 2012.

[2] Quattoni A., Collins M., Darrell T. Conditional random fields for object recognition. *Proc. NIPS'04*, 2004. Available at: https://papers.nips.cc/paper/2652-conditional-random-fields-for-object-recognition.pdf

[3] Bekkerman R., Sahami M., Learned-Miller E. Combinatorial Markov random fields. In: Fürnkranz J., Scheffer T., Spiliopoulou M. (eds). Machine Learning: ECML 2006. ECML 2006. *Lecture Notes in Computer Science*, vol. 4212. Berlin, Heidelberg, Springer, 2006, pp. 30–41. DOI: https://doi.org/10.1007/11871842_8

[4] Sarawagi S., Cohen W.W. Semi-Markov conditional random fields for information extraction. *Proc. NIPS'04*, 2004. Available at: https://papers.nips.cc/paper/2648-semi-markov-conditional-random-fields-for-information-extraction.pdf

[5] Rimstad K., Omre H. Skew-Gaussian random fields. *Spat. Stat.*, vol. 10, no. 11, pp. 43–62. DOI: 10.1016/j.spasta.2014.08.001

[6] Deon A.F., Menyaev Y.A. Uniform twister plane generator. *J. Comput. Sci*., 2018, vol. 14, iss. 2, pp. 260–272. DOI: 10.3844/jcssp.2018.260.272

[7] Xiao Y. Uniform modulus of continuity of random fields. *Monatsh. Math.*, 2010, vol. 159, iss. 1-2, pp. 163–184. DOI: 10.1007/s00605-009-0133-z

[8] Deon A., Menyaev Y. 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: 10.3217/jucs-022-08-1023

[9] Deon A., Menyaev Y. Parametrical tuning of twisting generators. *J. Comput. Sci.,* 2016, vol. 12, iss. 8, pp. 363–378. DOI: 10.3844/jcssp.2016.363.378

[10] Deon A.F., Menyaev Y.A. Twister generator of arbitrary uniform sequences. *JUCS*, 2017, vol. 23, iss. 4, pp. 353–384. DOI: 10.3217/jucs-023-04-0353

[11] Qi Y., Szummer M., Minka T.P. Bayesian conditional random fields. *Proc. AISTATS'05*, 2005, pp. 269–276.

[12] Kumar S., Hebert M. Discriminative random fields: a discriminative framework for contextual interaction in classification. *Proc. ICCV'03*, 2003, pp. 1150–1157.

DOI: 10.1109/ICCV.2003.1238478

[13] Sung Y., Jurafsky D. Hidden conditional random fields for phone recognition. *Proc. ASRU'09*, 2009, pp. 107–112. DOI: 10.1109/ASRU.2009.5373329

[14] Spanos P.D., Zeldin B.A. Monte Carlo treatment of random fields: a broad perspective. *Appl. Mech. Rev*., 1998, vol. 51, no. 3, pp. 219–237. DOI: 10.1115/1.3098999

[15] Newman M.E.J., Barkema G.T. Monte Carlo study of the random-field Ising model. *Phys. Rev. E*, 1996, vol. 53, iss. 1, pp. 393–404. DOI: 10.1103/PhysRevE.53.393

[16] Kim J., Zabih R. Factorial Markov random fields. In: Heyden A., Sparr G., Nielsen M., Johansen P. (eds). Computer Vision — ECCV 2002. ECCV 2002. *Lecture Notes in Computer Science*, vol. 2352. Berlin, Heidelberg, Springer, 2002, pp. 321–334.
DOI: https://doi.org/10.1007/3-540-47977-5_21

[17] Sha F., Pereira F. Shallow parsing with conditional random fields. *Proc. NAACL'03*, 2003, vol. 1, pp. 134–141. DOI: 10.3115/1073445.1073473

[18] Menyaev Y.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: 10.1364/BOE.7.003643

[19] Matsumoto M. Nishimura T. Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator. *TOMACS*, 1998, vol. 8, iss. 1, pp. 3–30. DOI: 10.1145/272991.272995

[20] Deon F.F., Menyaev Yu.A. Generator of uniform twister sequences of random integer numbers without storage arrays. *Herald of the Bauman Moscow State Technical University, Series Instrument Engineering*, 2018, no. 3, pp. 51–69 (in Russ.).

DOI: 10.18698/0236-3933-2018-3-51-69

[21] Matsumoto M., Wada I., Kuramoto A., et al. Common defects in initialization of pseudorandom number generators. *TOMACS*, 2007, vol. 17, no. 4, art. 15.

DOI: 10.1145/1276927.1276928

[22] Saito M., Matsumoto M. SIMD-oriented fast Mersenne twister: a 128-bit pseudorandom number generator. In: Keller A., Heinrich S., Niederreiter H. (eds). *Monte Carlo and Quasi-Monte Carlo Methods 2006.* Berlin, Heidelberg, Springer, 2008, pp. 607–622. DOI: https://doi.org/10.1007/978-3-540-74496-2_36

[23] Deon A.F., Menyaev Yu.A. Complete factorial simulation of integer random number uniform sequences. *Herald of the Bauman Moscow State Technical University, Series Instrument Engineering*, 2017, no. 5, pp. 132–149 (in Russ.).

DOI: 10.18698/0236-3933-2017-5-132-149

[24] Deon A.F., Menyaev Yu.A. Uniform random quantity generator using complete vortex array technology. *Herald of the Bauman Moscow State Technical University, Series Instrument Engineering*, 2017, no. 2, pp. 86–110 (in Russ.).

DOI: 10.18698/0236-3933-2017-2-86-110

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).

Onuchin V.A. — Post-Graduate Student, Department of Radioelectronic Systems and Devices, Bauman Moscow State Technical University (2-ya Baumanskaya ul. 5, str. 1, Moscow, 105005 Russian Federation).

Menyaev Yu.A. — Cand. Sc. (Eng.), works in 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., Onuchin V.A., Menyaev Yu.A. Twister generator of stochastic planes. *Herald of the Bauman Moscow State Technical University, Series Instrument Engineering*, 2019, no. 3, pp. 27–45 (in Russ.). DOI: 10.18698/0236-3933-2019-3-27-45