ГЛАВА 2. ПРОГРАММИРОВАНИЕ БАЗОВЫХ ГЕНЕРАТОРОВ ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ

2.1. Общие сведения

При имитационном моделировании стохастических систем выполняется реализация алгоритмов, описывающих процесс функционирования системы во времени. При этом воспроизводятся элементарные явления, происходящие в системе, с учетом их логической и временной последовательности. Для моделирования таких явлений, обычно носящих случайный характер, используются случайные числа. Следовательно, при имитационном моделировании необходимо уметь генерировать случайные величины с требуемыми параметрами и характеристиками.

Большинство системных программных средств общего пользования имеют в своем составе специальные стандартные подпрограммы для генерации случайных величин. Однако во многих случаях оказывается, что нужные пользователю стандартные подпрограммы либо вообще отсутствуют, либо не отчевают предъявляемым требованиям. Поэтому необходимо уметь самостоятельно разрабатывать и проверять генераторы случайных величин.

Базовым генератором для получения последовательностей случайных величин с любым заданным распределением, является генератор равномерно распределенной случайной величины на интервале [0;1). Такая непрерывная случайная величина X имеет функцию распределения

$$F_x(x) = \begin{cases} 0, \text{ при } x < 0; \\ x, \text{ при } 0 \le x < 1; \\ 1, \text{ при } x \ge 1 \end{cases} \tag{2.1}$$

и функцию плотности распределения вероятности

$$f_x(x) = \begin{cases} 1, \text{ при } 0 \le x < 1; \\ 0, \text{ при других } x. \end{cases} \tag{2.2}$$

Математическое ожидание такой равномерно распределенной случайной величины X равно $M(X)=1/2$, дисперсия $D(X)=1/12$. Графики плотности $f_x(x)$ и функции распределения $F_x(x)$ изображены на рис. 2.1. В дальнейшем равномерно распреде-

ленные на интервале [0;1) случайные величины будем называть случайными числами.

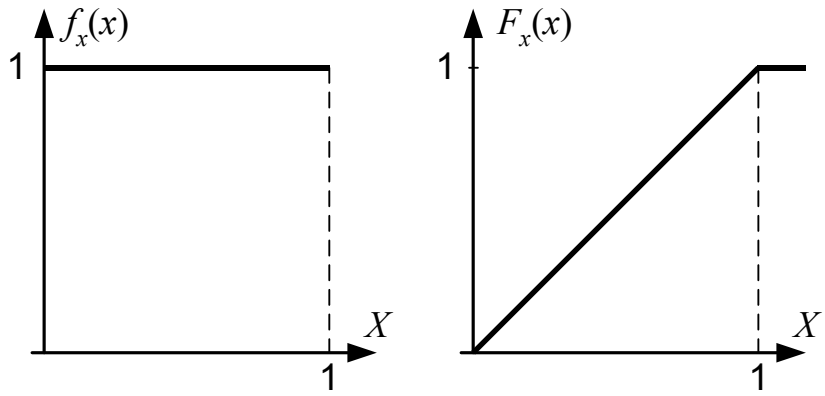

Рис. 2.1. Функция плотности вероятности и функция распределения для равномерного распределения на интервале [0;1)

Равномерное распределение на интервале [0;1) очень важно, поскольку случайные величины из всех других распределений могут быть получены путем преобразования независимых случайных чисел некоторым способом, определяемым нужным распределением. Некоторые из таких преобразований будут рассмотрены в четвертой главе.

Для генерации случайных чисел может применяться один из следующих способов:

1. *Аппаратный*. В основе работы аппаратного генератора лежит какой-либо физический эффект (например, шумы в электронных устройствах или другие явления). Достоинством такого генератора является то, что случайные числа, получаемые с его помощью, являются абсолютно случайными. Также запас чисел аппаратного генератора неограничен. В качестве недостатка аппаратного генератора можно отметить, что данный способ не гарантирует качество получаемой последовательности случайных чисел непосредственно во время моделирования. Кроме того, с помощью аппаратного генератора нельзя получать одинаковые последовательности случайных чисел, которые желательно использовать при отладке моделирующих программ и сравнении альтернативных вариантов организации системы. Поэтому при моделировании аппаратные генераторы практически не используются.

2. *Табличный*. При использовании табличного генератора заранее полученные и проверенные случайные числа оформлены в виде таблицы в памяти ЭВМ. Данный способ свободен от отмеченных недостатков аппаратного генератора. Однако при использовании табличного генератора запас случайных чисел ограничен размером таблицы. Кроме этого неэффективно используются ресурсы ЭВМ. Поэтому данный способ при моделировании используется сравнительно редко.

3. *Арифметический* (*численный*). При использовании арифметического генератора каждое новое число определяется одним или несколькими предшествующими числами в соответствии с заданной математической формулой. Данный способ свободен от описанных недостатков ранее рассмотренных генераторов. Поэтому при моделировании чаще всего используются именно арифметические генераторы.

Отметим, что при получении последовательности случайных чисел на ЭВМ с помощью арифметических генераторов используются алгоритмы. Поэтому такие последовательности, являющиеся по сути детерминированными, но для стороннего наблюдателя выглядящие как случайные, называются *псевдослучайными*. Также заметим, что любая ЭВМ оперирует только n-разрядными числами. Поэтому на ЭВМ вместо непрерывной совокупности равномерных случайных чисел из интервала [0,1) может получиться лишь дискретная последовательность не более чем 2^n случайных чисел из того же интервала. Поэтому закон распределения такой дискретной последовательности называется *квазиравномерным*.

Далее перечислим основные требования, предъявляемые к идеальному генератору случайных чисел [4]:

– последовательность должна состоять из квазиравномерно распределенных чисел;

– числа должны быть статистически независимыми;

– последовательности случайных чисел должны быть воспроизводимыми;

– последовательности должны быть непериодическими;

– последовательности должны получаться с минимальными затратами вычислительных ресурсов;

– генераторы должны занимать минимальный объем машинной памяти.

Далее рассмотрим основные типы арифметических генераторов случайных чисел. Заметим, что иногда генераторы случайных чисел в тексте учебника называются датчиками.

2.2. Арифметические генераторы случайных чисел

Известно большое число алгоритмов, процедур и методов получения случайных (псевдослучайных) чисел. В общем случае программные генераторы формируют последовательность чисел по некоторой рекуррентной формуле

$$x_{n+1} = \phi\left(x_n, x_{n-1}, ..., x_{n-r}\right) \tag{2.3}$$

где φ означает совокупность операций, которые необходимо проделать над числами x_n, x_{n-1}, …, x_{n-r}, чтобы получить x_{n+1}. При этом качество последовательности x_{r+1}, …, x_n в значительной мере зависит от начальных (исходных) чисел x_0, x_1, ..., x_r.

В результате многократного применения процедуры (2.3) получается последовательность чисел, которая носит детерминированный характер, но в определенных границах она удовлетворяет свойствам равномерного распределения и свойству случайности.

2.2.1. Конгруэнтный (линейный) метод

Наиболее широко применяемые в настоящее время программные генераторы равномерно распределенных чисел в общем виде можно представить с помощью следующей формулы:

$$y_{n+1} = \left(\sum_{i=0}^{r} a_i y_{n-i} + \mu \right) \bmod m, \tag{2.4}$$

где a_0, a_1, ..., a_r, μ и m, а также получаемые числа y_0, y_1, ... являются целыми числами. Образование равномерно распределенных псевдослучайных чисел на интервале [0;1) реализуется с помощью выражения

$$x_{n+1} = \frac{y_{n+1}}{m}. \tag{2.5}$$

Формулы (2.4) и (2.5) определяют конгруэнтный метод, частные случаи которого задают тот или иной датчик случайных чисел.

Алгоритм Лемера (мультипликативный датчик)

Датчик случайных чисел, основанный на использовании алгоритма Лемера, является частным случаем линейного конгруэнтного метода. Подставляя в (2.4)

$$\mu = a_1 = ... = a_j = 0$$

и полагая, что $a = a_0 > 0$, получаем

$$y_{n+1} = a \cdot y_n \bmod m. \qquad (2.6)$$

В алгоритме Лемера задаются два соответствующим образом подобранных целых числа: множитель a и модуль m, а также начальное значение y_0. Последовательность случайных чисел вычисляется следующим образом.

1) Число y_i известно с предыдущего шага. Вычисляется произведение $a \cdot y_i$.

2) Получается остаток y_{i+1} от деления $a \cdot y_i$ на m.

3) Так как y_{i+1} является целым числом из интервала [0, m), то нужно его разделить на m, чтобы получить число из интервала [0, 1), т.е.

$$x_{i+1} = \frac{y_{i+1}}{m}.$$

Следует отметить, что с помощью (2.6) можно получить всего m различных значений y_i. Поэтому на некотором шаге T обязательно получится y_T, которое уже было получено ранее. А так как следующее значение, формируемое генератором, зависит только от предшествующего, то формируемая далее последовательность начнет повторяться. Количество чисел T, после получения которых последовательность начинает повторяться, называется *периодом генератора*.

Значения y_0, a и m выбираются чаще всего исходя из требования получения максимально возможного периода.

Например, если $m=2^l$, где $l>2$ – число двоичных разрядов для задания максимального значения целой константы в ЭВМ (например l может быть равно 15 или 31). Данный выбор определяется тем, что при таком m вычислительно легко получить остаток от деления по модулю m. Максимальный период в этом случае достигается при нечетном y_0 и значении a mod 8, равном 3 или 5, что выполняется, когда

$a=5^{2p+1}$, p=0, 1, 2, … (или $a=2^s+3$, s=3, 4, 5, …). Значение максимально достижимого периода в данном случае $T_{макс}=2^{l-2}=m/4$.

В том случае, если в качестве m выбрать наибольшее простое число, меньшее 2^l, можно получить генератор с периодом $m-1$. Такой генератор называется мультипликативным линейным конгруэнтным генератором с простым модулем. Например, если b=31, то $m=2^{31}-1$=2147483647. Для такого m можно получить период $m-1$, если a – это первообразный элемент по модулю m, то есть наименьшее целое число l, для которого a^l-1 делится на m, составляет $l=m-1$.

Заметим, что при построении такого генератора сложно выбирать параметр a. Хорошие свойства имеют генераторы с $m=2^{31}-1$=2147483647 и $a=a_1$=16807 или $a=a_2$=630360016, причем генератор с $a=a_2$ лучше [4]. Также отметим, что данный генератор работает несколько медленнее из-за более сложного получения остатка от деления на $m-1$.

В качестве примера приведем консольную программу на языке C#, реализующую и использующую функцию для генерации случайных чисел. Данная функция реализует мультипликативный генератор с параметрами: с параметрами: y_0=2451, m=4096, a=5.

```
namespace Mult_RND
{
  class Program
  {
    // Количество случайных чисел
    private const int N = 1500;
    // Параметр мультипликативного генератора
    private const int A = 5;
    // Модуль
    private static int _Mm;
    // Случайное число
    private static int _Y;
    /// <summary> Генератор случайных чисел </summary>
    /// <returns> Случайное значение </returns>
    private static double Rnd()
    {
      _Y = (A * _Y) % _Mm;
      return (double)_Y / _Mm;
    }
```

```
    /// <summary> Основная программа </summary>
    /// <param name="args"> Аргументы </param>
    static void Main(string[] args)
    {
      _Mm = 4096;
      _Y = 2451;
      // Массив для случайных чисел
      double[] x = new double[N];
      for (int i = 0; i < N; i++)
      {
        // получение случайного числа
        x[i] = Rnd();
        ... // обработка случайного числа
      }
      ... // обработка массива случайных чисел
    }
  }
}
```

Смешанный генератор

Если в (2.4) положить $a_1 = a_2 = a_3 = \ldots = a_r = 0$ и $a = a_0 > 0$, $\mu > 0$, то получается выражение для смешанного генератора:

$$y_{n+1} = (ay_n + \mu) \bmod m . \tag{2.7}$$

Целые числа a, μ и m для получения максимального периода генератора, равного m, должны удовлетворять следующим условиям:

1) $a \bmod 8 = 5$;

2) $m/100 < a < m - \sqrt{m}$;

3) двоичные знаки числа a не должны иметь очевидного шаблона;

4) $\mu / m \approx 1/2 - \sqrt{3}/6$;

5) μ – нечетное;

6) m – целое число, максимальное для данной ЭВМ.

В случае 32-разрядного представления чисел можно использовать следующие значения параметров [4]:

$$a = 16070093;\ \mu = 453816693;\ m = 2^{31} = 2147483648 .$$

Запишем пример функции, реализующей смешанный генератор.

```
private double Rnd()
{
  _Y = (A * _Y + _Mu) % _Mm;
  return (double)_Y / _Mm;
}
```

Для использования этой функции в основной программе необходимо задать *A*, *_Mu*, *_Mm* и начальное значение *_Y*, например так:

```
...
_A = 165;
_Mu = 3463;
_Mm = 4096 * 4;
_Y = 3887;
double[] x = new double[N];
for (int i = 0; i < N; i++)
{
  x[i] = Rnd();
  ...
}
...
```

Аддитивные генераторы

Если в (2.4) положить $a_0 = a_1 = 1$ и $\mu = a_2 = ... = a_r = 0$, то полученный генератор называется аддитивным или генератором Фибоначчи:

$$y_{n+1} = (y_n + y_{n-1}) \bmod m. \qquad (2.8)$$

В отличие от ранее рассмотренных генераторов, в генераторе Фибоначчи задаются два начальных значения y_0 и y_1.

Для аддитивного генератора можно использовать следующую функцию.

```
private double Rnd()
{
  int y = (_Y0 + _Y1) % _Mm;
  _Y0 = _Y1;
  _Y1 = y;
  return (double)y / _Mm;
}
```

При этом *_Mm* и начальные значения *_Y*0, *_Y*1 должны быть определены в основной программе, например так:

```
...
_Mm = 4096 * 4;
_Y0 = 3971;
_Y1 = 1013;
```

```
double[] x = new double[N];
for (int i = 0; i < N; i++)
{
  x[i] = Rnd();
  ...
}
...
```

Обобщенный аддитивный генератор

Генератор вида (2.4), где все $a_i = 1$, $i = 0, 1, ..., r$, называется обобщенным аддитивным генератором. Для получения псевдослучайной последовательности чисел $x_0, x_1, ..., x_N$, равномерно распределенных на интервале [0;1), обобщенный аддитивный генератор реализуется в виде следующего алгоритма:

$$x_{n+1} = \{x_n + x_{n-1} + ... + x_{n-r}\}, \tag{2.9}$$

где $\{\psi\}$ означает дробную часть числа ψ, т.е. $\{\psi\} = \psi - \lfloor\psi\rfloor$, $\lfloor\psi\rfloor$ – целая часть ψ. Начальные значения $x_0, x_1, ..., x_r$ выбираются из таблицы случайных чисел. Количество слагаемых в формуле (2.9) может быть любым. В практических случаях ограничиваются r=6…8. Рассмотрим основные фрагменты программы, реализующей формулу (2.9).

```
...
// Количество случайных чисел, необходимых для
// получения нового случайного числа
private const int R = 6;
...
private static double Rnd()
{
  double s = 0.0;
  for (int k = 0; k < R; k++)
  {
    s += _Rand[k]; // сумма случайных чисел
  }
  for (int k = 1; k < R; k++)
  {
    // сдвиг случайных чисел
    _Rand[k - 1] = _Rand[k];
  }
  s -= Math.Truncate(s);
  // новое случайное число
  _Rand[R - 1] = s;
```

```
  return s;
}
...
static void Main(string[] args)
{
  // инициализация стандартного датчика
  Random random = new Random();

  for (int i = 0; i < R; i++)
  {
    // генерация первых случайных чисел
    _Rand[i] = random.NextDouble();
  }
  for (int i = 0; i < N_MAX; i++)
  {
    double x = Rnd();
    ...
  }
  ...
}
```

2.2.2. Комбинации генераторов случайных чисел

С целью предотвращения использования при моделировании циклично повторяющихся последовательностей (для увеличения длины периода) широко используют комбинацию генераторов случайных чисел.

Рассмотрим простейший пример такого генератора, использующего два простых генератора. С помощью них получают две последовательности случайных чисел, одну с периодом N_1, другую с периодом N_2. Обозначим через y_{ij} j-й член (j=1, 2, …) i-й последовательности (i=1, 2) и через z_j – j-й член новой последовательности, образующейся по формуле

$$z_j = (y_{1j} + y_{2j}) \bmod m. \qquad (2.10)$$

Если периоды N_1 и N_2 – взаимнопростые числа, то результирующая последовательность z_j – периодическая с периодом $N=N_1 \cdot N_2$.

Процедуру (2.10) можно обобщить путем увеличения числа первоначальных последовательностей. Например, если число первоначальных последовательностей равно r, то j-й член результирующей последовательности получается по формуле

$$z_j = \left(\sum_{i=1}^{r} y_{ij} \right) \bmod m . \tag{2.11}$$

Еще в качестве одного примера рассмотрим универсальный датчик, в котором для вычисления случайного числа используются три идентичных мультипликативных датчика. Каждый из первоначальных датчиков имеет вид:

$$y_{k,n+1} = \left| a_k \left(y_{k,n} \bmod B_k \right) - \frac{c_k y_{k,n}}{B_k} \right|, \tag{2.12}$$

$$x_{k,n+1} = \frac{y_{k,n+1}}{d_k},$$

где k=1, 2, 3 – номер первоначального датчика; a_k, b_k, c_k, d_k – целые числа; $y_{k,n}$ – n-е случайное число, генерируемое k-м датчиком и изменяющееся от 0 до $(d_k - 1)$; $x_{k,n}$ – случайное число, принадлежащее интервалу [0;1).

Результирующее случайное число вычисляется по формуле

$$z_n = \left\{ x_{1,n} + x_{2,n} + x_{3,n} \right\}. \tag{2.13}$$

В качестве исходных данных можно взять следующие целые константы:

$$a_1 = 171;\ b_1 = 177;\ c_1 = 2;\ d_1 = 30269; \tag{2.14}$$

$$a_2 = 172;\ b_2 = 176;\ c_2 = 35;\ d_2 = 30307; \tag{2.15}$$

$$a_3 = 170;\ b_3 = 178;\ c_3 = 63;\ d_3 = 30323. \tag{2.16}$$

Функция, реализующая универсальный датчик, имеет вид.

```
private double Rnd()
{
  double s = 0.0;
  double[] x = new double[K];
  int[] yN = new int[K];
  for (int i = 0; i < K; i++)
  {
    yN[i] = (int) Math.Abs(_A[i] * (_Y[i] % _B[i]) -
                           _C[i] * (double)_Y[i] / _B[i]);
    _Y[i] = yN[i];
    x[i] = (double) yN[i] / _D[i];
    s += x[i];
  }
```

```
  return Math.Abs(s - Math.Truncate(s));
}
```

При этом в основной программе должны быть объявлены и определены вспомогательные массивы и константа *K*.

```
...
// Количество первоначальных датчиков
private const int K = 3;
// Вспомогательные массивы
private int[] _A = new int[K];
private int[] _B = new int[K];
private int[] _C = new int[K];
private int[] _D = new int[K];
...
```

Метод Макларена – Марсальи

Метод Макларена – Марсальи основан на использовании двух последовательностей случайных чисел $x_{1,1}, x_{1,2}, ..., x_{1,n}, ...$ и $x_{2,1}, x_{2,2}, ..., x_{2,n}, ...$, получаемых с помощью двух начальных датчиков.

Первоначально с помощью первого генератора вырабатывают последовательность $x_{1,1}, x_{1,2}, ..., x_{1,k}$ из k чисел, обычно k = 64, 128, 256.

В дальнейшем с помощью первого и второго датчиков генерируют очередные $x_{1,n+k}$ и $x_{2,n}$ числа. Число z_n комбинированной последовательности получают в результате выполнения операций

$$\begin{aligned} M &= [x_{2,n} \cdot k] + 1; \\ z_n &= x_{1,M}; \\ x_{1,M} &= x_{1,n+k}. \end{aligned} \qquad (2.17)$$

Операции (2.17) повторяются необходимое число раз. В результате с помощью значений, полученных вторым генератором, реализуется случайный выбор числа из последовательности k чисел $x_{1,1}, x_{1,2}, ..., x_{1,k}$, а также ее случайное заполнение новыми числами.

Рассмотрим основные фрагменты программы, необходимые для реализации метода Макларена – Марсальи.

```
// Объем выборки
private const int N_MAX = 4000;
```

```
// Размер 1-ой последовательности случайных чисел
private const int K = 64;
// 1-ая случайная последовательность
private static double[] _Z1 = new double[K];
// Стандартный датчик
private static Random _Random = new Random();
...
private static double Rnd()
{
  // случайное число 1-ой последовательности
  double g1 = _Random.NextDouble();
  // случайное число 2-ой последовательности
  double g2 = _Random.NextDouble();
  int m = (int) (g2 * K);
  double res = _Z1[m];
  // заполнение элемента первой последовательности
  // новым числом
  _Z1[m] = g1;
  return res;
}
...
static void Main(string[] args)
{
  for (int i = 0; i < K; i++)
  {
    // генерация первых K случайных чисел
    _Z1[i] = _Random.NextDouble();
  }
  double x;
  for (int i = 0; i < N_MAX; i++)
  {
    x = Rnd();
    ...
  }
  ...
}
```

2.2.3. Арифметические процедуры

В ряде случаев для генерации случайных чисел бывает целесообразно использовать последовательность цифр разложения в десятичную дробь иррациональных чисел, в частности числа $\pi = 3.1415926...$, $\sqrt{3} = 1.7320508...$, $e = 2.7182818...$ и др. Одним из главных достоинств таких последовательностей десятичных цифр является отсутствие периодичности.

2.2.4. Алгоритмы на основе нелинейных формул

Квадратичный конгруэнтный метод

Обобщением линейного конгруэнтного метода является квадратичный конгруэнтный метод формирования случайной последовательности чисел

$$y_{n+1} = (Ay_n^2 + By_n + C) \bmod m, \tag{2.18}$$

где $m = 2^l$. Если $l \geq 2$, то наибольшее значение периода квадратического конгруэнтного датчика составляет $T_{\max} = 2^l$, что достигается при четном A, нечетном C и если нечетное B удовлетворяет условию $B \bmod 4 = (A+1) \bmod 4$.

Функция, реализующая формулу (2.18), имеет вид.

```
private double Rnd()
{
  _Y = (_A * _Y * _Y + _B * _Y + _C) % _Mm;
  return (double) _Y / _Mm;
}
```

При этом в основной программе должны быть заданы *_A*, *_B*, *_C*, *_Mm* и начальное значение *_Y*, например:

```
...
_A = 6;
_B = 7;
_C = 3;
_Mm = 4096;
_Y = 4001;
for (int i = 0; i < N; i++)
{
  double x = Rnd();
  ...
}
...
```

Квадратичный метод Ковэю

При использовании данного метода вычисление очередного числа осуществляется по формуле

$$y_{n+1} = y_n (y_n + 1) \bmod m, \tag{2.19}$$

где $m = 2^l$, а начальное значение y_0 удовлетворяет равенству $y_0 \bmod 4 = 2$. Длина периода случайной последовательности такого датчика равна 2^{l-2}.

Функция, реализующая формулу (2.19), имеет вид.

```
private double Rnd()
{
  _Y = _Y * (_Y + 1) % _Mm;
  return (double)_Y / _Mm;
}
```

При этом в основной программе должны быть заданы *_Mm* и *_Y*, например:

```
...
_Mm = 512;
_Y = 2135;
for (int i = 0; i < N; i++)
{
  double x = Rnd();
  ...
}
...
```

2.3. Контрольные вопросы

1. Каким образом задается равномерно распределенная случайная величина?
2. Какие основные способы генерации случайных чисел Вы знаете?
3. В чем заключается конгруэнтный метод генерации равномерно распределенных случайных чисел?
4. Какие разновидности конгруэнтного метода генерации вы знаете?
5. Что такое период псевдослучайной последовательности? Опишите способы увеличения длины периода.
6. Сформулируйте основные принципы выбора составляющих датчиков случайных чисел для формирования сложного датчика.

ГЛАВА 11.ЛАБОРАТОРНЫЙ ПРАКТИКУМ

В данной главе представлено девять лабораторных работ, снабженных индивидуальными вариантами заданий, которые рекомендуется выполнить учащимся, изучающим дисциплину «Компьютерное моделирование»

11.1. Лабораторная работа №1. Изучение базовых генераторов псевдослучайных чисел

Составить и отладить программу (подпрограмму) генерирования псевдослучайных чисел с равновероятным распределением на интервале [0;1). Вариант выполняемого задания выбирается из таблицы 10.1, в которой указаны тип генератора случайных чисел, начальные условия и пр. Для заданных объема выборки и числа участков разбиения интервала [0;1) построить гистограмму частот и статистическую функцию распределения, получить программным способом оценки математического ожидания, дисперсии, второго и третьего моментов. Выполните анализ полученных результатов.

Таблица 10.1. Варианты заданий к лабораторной работе №1

№ вар.	Тип датчика	Начальные данные	Объем выборки	Число участков разбиения
1.	Мультипликативный, формула (2.6)	$Y_0 = 4003$, $M = 4096$	256	16
2.	Универсальный, формула (2.12), $k = 3$	Y_k – любые	500	16
3.	Аддитивный, формула (2.8)	$Y_1 = 3215$, $Y_2 = 4073$, $m = 4096*4$	5000	10
4.	Универсальный, формула (2.12), $k = 1$	Y – любое	1600	18
5.	Квадратичный конгруэнтный метод, формула (2.18)	Y – любое $I = 12$	5000	25
6.	Квадратичный метод Ковэю, формула (2.19)	$I = 9$, Y_1 из условия $Y_1 \bmod 4 = 2$	5000	12
7.	Квадратичный конгруэнтный метод, формула (2.18)	Y – любое $I = 12$	7000	16
8.	Метод Макларена-Марсальи	$k = 256$	1000	21

№ вар.	Тип датчика	Начальные данные	Объем выборки	Число участков разбиения
9.	Смешанный, формула (2.7)	Y – любое	4000	16
10.	Аддитивный, формула (2.8)	$Y_1 = 4091$, $Y_2 = m - 5$ $m = 4096*4$	1000	16
11.	Обобщенный аддитивный, формула (2.9)	$r = 6$; $x_1, ..., x_6$ из таблицы случайных чисел	6000	16
12.	Обобщенный аддитивный, формула (2.9)	$r = 10$; $x_1, ..., x_{10}$ из таблицы случайных чисел	6000	16
13.	Аддитивный, формула (2.8)	$Y_1 = 3971$, $Y_2 = 1013$ $m = 4096*4$	2000	21
14.	Метод Макларена-Марсальи	$k = 64$	2000	16
15.	Смешанный, формула (2.7)	$Y = 3845$	1000	16
16.	Обобщенный аддитивный, формула (2.9)	$r = 8$; $x_1, ..., x_8$ из таблицы случайных чисел	6000	26
17.	Метод Макларена-Марсальи	$k = 128$	5000	26
18.	Смешанный, формула (2.7)	Y =4001	1500	16
19.	Универсальный, формула (2.12), $k = 2$	Y_k – любое	4000	21
20.	Мультипликативный, формула (2.6)	$Y_0 = 3091$, $M = 4096$	2000	21