УДК 681.3.069

Ю. В. Щербина, к.т.н., **С. Л. Волков,** к.т.н., **Н. Ф. Казакова,** д.т.н.

Одесская государственная академия технического регулирования и качества, г. Одесса

ВЫЧИСЛЕНИЕ СПЕЦИАЛЬНЫХ ФУНКЦИЙ *igam*(*a*, *x*) И *igamc*(*a*, *x*) НА ЯЗЫКЕ C# В СТАТИСТИЧЕСКОМ МОДЕЛИРОВАНИИ

Предложен компактный вариант вычисления на языке C# *таких специальных функций как дополнительная гамма-функция, неполная дополнительная гамма-функция, натуральный логарифм от гамма-функции и квантили распределения хи-квадрат, применяемые в ходе решения задач статистического моделирования.*

Ключевые слова*: специальные функции, вычислительные алгоритмы, статистическое моделирование.*

Постановка проблемы в общем виде и ее связь с важными научными или практическими задачами.

Построение современных автоматизированных измерительных систем невозможно представить без широкого применения статистического математического моделирования. Основу программной реализации статистических моделей, применяемых в измерительных системах, как правило, составляют датчики равномерно распределенных в заданном диапазоне случайных чисел. Из псевдослучайных последовательностей, формируемых на их выходе, достаточно просто формируются последовательности с иными видами распределения вероятностей следования символов. Проблемой создания таких генераторов является требование высокой степени равномерности распределения символов. До настоящего времени единственным надежным методом испытания такой равномерности остается статистическое тестирование с помощью тестовых пакетов [1 – 3]. Суть этого тестирования сводится к оценке соответствия распределения вероятностей символов, формируемых псевдослучайных последовательностей, равномерному закону распределения. В основу этого тестирования заложен сформулированный Д. Кнутом принцип, предполагающий многократное испытание шифрующих псевдослучайных последовательностей (ПСП) большим числом независимых тестов. Чем большее число тестов дает положительный результат, тем больше доверие к проектируемому шифру [4]. Пакет [1] был опубликован как стандарт NIST, а остальные пакеты доступны в сети Internet. Однако, учитывая высокие требования к результатам тестирования, вошедшие в пакет тесты, предъявляют серьезные требования к точности измерения параметров, на основе которых определяется критерий пригодности испытуемой ПСП для криптографических нужд.

Подходы и способы тестирования были сформулированы и разработаны достаточно давно. Эта работа активно проводилась всю вторую половину прошлого века и, к 2000 году, было создано и стандартизировано программное обеспечение, позволяющее решать задачи анализа криптографической стойкости симметричных шифров.

Постановка задачи.

Проблема заключается в том, что практически все известные пакеты тестирования предполагают вычисление специальных функций, значение которых не может быть рассчитано аналитическим путем. В частности, к числу таких функций относят гамма-функцию, дополнительную гамма-функцию и неполную дополнительную гамма-функцию. Вычисление этих функций требует, в свою очередь, расчета иных дополнительных специальных функций. В принципе, их вычисление не является новой задачей, и к настоящему времени в сети Internet можно найти множество свободно распространяемых и даже стандартизированных программных пакетов, которые позволяют производить необходимые вычисления. Широко известны такие прикладные программные пакеты как Matlab, Mathematica, Mathcad, Maple, Statistica, Tabula и многие им подобные. Однако они рассчитаны не только на вычисление специальных функций, а также и на решение целого ряда иных традиционных математических задач, требующих сложных математических расчетов. Они громоздки и их разработчики, как правило, не предоставляют доступа к исходным файлам программного обеспечения, поэтому вычисления с помощью таких пакетов можно производить, только обращаясь к исполняемым файлам, что не всегда удобно.

В реальной жизни разработчики статистических моделей создают свое программное обеспе-

чение, которое, порой, требует многократного вычисления значений специальных функций. Поэтому было бы предпочтительней «зашить» эти расчеты в исходный код программы.

Сложность вычисления специальных функций заключается в том, что описывающие их аналитические выражения, как правило, не раскладываются на элементарные функции и, поэтому, их вычисляют, разлагая в ряды. При этом вид ряда в каждом конкретном случае определяется требуемой скоростью и точностью вычислений. В прошлом, было написано достаточно много программ для вычисления этих функций таким способом. Доступными сайтами, где можно получить исходные коды программ для решения подобных задач являются сайты AlgoList (http://algolist.manual.ru/maths/) и AlgLib (http://www.alglib.net/). Эти коды написаны либо на языке C++, либо на Delphi, либо на более ранних версиях иных языков высокого уровня, которые применялись до появления платформы .NET. К настоящему моменту времени существенно обновились и системное обеспечение современных компьютеров, и языки программирования. Сегодня наиболее популярным является новый язык программирования C# от компании Microsoft. Он, как и языки Visual C++ и Visual Basic.NET, входит в пакет Visual Studio.NET, которые являются компонентно-ориентированными языками для новой платформы .NET и которые разрабатывались как альтернатива языку Java. С учетом этого *целью статьи является* описание класса, содержащего программный код, написанный на языке C# для вычисления дополнительной гамма-функции и неполной дополнительной гамма-функции. Кроме того, в этот класс включена процедура для расчета квантилей распределения χ^2.

Изложение основного материала исследования.

Предлагаемый программный модуль включает класс `SpFunct`, состоящий из следующих общедоступных процедур: вычисления неполной гамма-функции **`igam(a, x)`**, вычисления дополнительной неполной гамма-функции **`igamc(a, x)`**, вычисления натурального логарифма от гамма-функции **`lngamma(x)`** и вычисления обратной неполной гамма-функции **`invigam(a, x)`**. Ниже приводится тест самого программного модуля, который может быть включен в проект, созданный в среде Visual C#.

```
   Using System;
   namespace CalcSpFunct
{
    class SpFunct
  {
// ======Incomplete gamma integral
igam(a,x)
   public double igam(double a, double
x)
   {
    double result = 0; double ans = 0;
    double igammaepsilon = 0; double
ax = 0;
    double c = 0; double r=0; double
tmp=0;
        igammaepsilon =
0.000000000000001;
        if ((double)(x) <= (double)(0)
| (double)(a) <= (double)(0))
           { result = 0; return re-
sult; }
        if ((double)(x) > (double)(1)
& (double)(x) > (double)(a))
 { result = 1-igamc(a, x); return re-
sult; }
   ax = a * Math.Log(x) – x – lngam-
ma(a, ref tmp);
if ((double)(ax) < (double)(-
709.78271289338399))
 { result = 0; return result; }
   ax = Math.Exp(ax); r = a; c = 1;
ans = 1;
do
   { r=r+1; c=c*x/r; ans = ans + c; }
While
  ((dou-
ble)(c/ans)>(double)(igammaepsilon));
   result = ans * ax / a; return re-
sult;
 }
// ===== Complemented incomplete gamma
integral igamc(a,x) = 1 – igam(a,x)
public double igamc(double a, double
x)
{
  double result = 0; double ans = 0;
double ax = 0;
double c=0; double yc=0; double r=0;
double t=0; double y = 0; double z =
0;
double pk=0; double pkm1=0; double
pkm2=0;
double qk=0;double qkm1=0; double
qkm2=0;
double tmp=0; double igammabignumber-
inv=0;
double igammaepsilon=0; double igam-
mabignumber=0;
igammaepsilon = 0.000000000000001;
igammabignumber = 4503599627370496.0;
```

```
        igammabignumberinv =
2.22044604925031308085 *
0.0000000000000001;
        if ((double)(x) <= (double)(0)
| (double)(a) <= (double)(0))
           { result = 1; return re-
sult; }
        if ((double)(x) < (double)(1)
| (double)(x) < (double)(a))
   { result=1 – igam(a, x); return re-
sult; }
        ax = a * Math.Log(x) – x –
lngamma(a, ref tmp);
        if ((double)(ax) < (double)(-
709.78271289338399))
           { result = 0; return re-
sult; }
        ax = Math.Exp(ax); y = 1 – a;
z = x + y + 1; c = 0;
        pkm2 = 1; qkm2 = x; pkm1 = x +
1; qkm1 = z * x; ans = pkm1 / qkm1;
        do
          {  c = c + 1; y = y + 1; z =
z + 2; yc = y * c;
             pk = pkm1 * z – pkm2 *
yc; qk = qkm1 * z – qkm2 * yc;
             if ((double)(qk) != (dou-
ble)(0))
                { r = pk / qk; t =
Math.Abs((ans – r) / r); ans = r; }
             else { t = 1; }
             pkm2 = pkm1; pkm1 = pk;
qkm2 = qkm1; qkm1 = qk;
             if ((dou-
ble)(Math.Abs(pk)) > (dou-
ble)(igammabignumber))
                { pkm2 = pkm2 * igam-
mabignumberinv; pkm1 = pkm1 * igam-
mabignumberinv;
                qkm2 = qkm2 * igam-
mabignumberinv; qkm1 = qkm1 * igam-
mabignumberinv; }
          } while ((double)(t) > (dou-
ble)(igammaepsilon));
           result = ans * ax; return
result;
}
// ===== Natural logarithm of gamma
function
   public static double lngamma(double
x, ref double sngam)
{
  double result=0; double a=0; double
b=0;
  double c=0; double p=0; double q=0;
  double u=0; double w=0; double z=0;
  double logpi=0; double ls2pi=0; dou-
ble tmp=0;
   int i=0; sngam=0; sngam=1;
       logpi = 1.1447298858494001741;
ls2pi = 0.91893853320467274178;
       if ((double)(x) < (double)(-
34.0))
       {
         q = -x; w = lngamma(q, ref
tmp);
         p = (int)Math.Floor(q); I =
(int)Math.Round(p);
         if (I % 2 == 0)
              { sngam = -1; }
         else
              { sngam = 1; }
         z = q – p;
         if ((double)(z) > (dou-
ble)(0.5))
            { p = p + 1; z = p – q; }
         z = q * Math.Sin(Math.PI *
z);
         result = logpi – Math.Log(z)
– w; return result;
       }
       if ((double)(x) < (double)(13))
       {
          z = 1; p = 0; u = x;
 while ((double)(u) >= (double)(3))
       { p = p – 1; u = x + p; z = z *
u; }
 while ((double)(u) < (double)(2))
       { z = z / u; p = p + 1; u = x +
p; }
          if ((double)(z) < (dou-
ble)(0))
             { sngam = -1; z = -z; }
          else
             { sngam = 1; }
          if ((double)(u) == (dou-
ble)(2))
    { result = Math.Log(z); return re-
sult; }
       p = p – 2; x = x + p;
       b = -1378.25152569120859100;
       b = -38801.6315134637840924 + x
* b;
       b = -331612.992738871184744 + x
* b;
       b = -1162370.97492762307383 + x
* b;
       b = -1721737.00820839662146 + x
* b;
```

```
       b = -853555.664245765465627 + x
* b;
       c = 1;
       c = -351.815701436523470549 + x
* c;
       c = -17064.2106651881159223 + x
* c;
       c = -220528.590553854454839 + x
* c;
       c = -1139334.44367982507207 + x
* c;
       c = -2532523.07177582951285 + x
* c;
       c = -2018891.41433532773231 + x
* c;
       p = x*b/c; result = Math.Log(z)
+ p;
       return result;
      }
    q = (x - 0.5) * Math.Log(x) - x +
ls2pi;
   if ((double)(x) > (dou-
ble)(100000000))
        { result = q; return result; }
          p = 1 / (x * x);
   if ((double)(x) >= (dou-
ble)(1000.0))
   {
q=q+((7.936507936507936507936*0.0001
*
     p - 2.777777777777777777778 *
0.001) *
     p + 0.083333333333333333333) /
x; }
   else
   { a = 8.11614167470508450300 *
0.0001;
     a=-
(5.950619042843014383240*0.0001)+p*a;

a=7.936503404577169439450*0.0001+p*a;
     a=-
(2.777777773009968720550*0.001)+p*a;

a=8.333333333333192772200*0.01+p*a;
     q = q + a / x; }
       result = q; return result;
   }
// Inversion incomplete gamma integral
invigam(a,x)
   public double invigam(double a,
double q)
   {
       double fx = 0, l = 0, r = 1, x
= 0;
       if (q == 0) return 0;
       if (q > 0 && q < 1)
       {
         for (l = 0, r = a / 2;
igam(a, r) < q; r += a / 2)
               l = r; x = (l + r) / 2;
         do
           { fx = igam(a, x);
             if (fx > q)  r = x;
             else if (fx < q) l = x;
                  else break; x = (l +
r) * 0.5;
           } while ((l != x) && (r !=
x));
           return x;
       }   else return 0;
   }
//
=====================================
===
  }
}
```

Здесь неполная гамма-функция **igam(a, x)** рассчитывается как

$$igam(a,x)=\frac{1}{\Gamma(a)}\int_0^x t^{a-1}e^{-t}dt\ .$$

Здесь Γ(*a*) – гамма-функция Эйлера от аргумента *a*, принимающего положительные значения.

$$\Gamma(a)=\int_0^\infty t^{a-1}e^{-t}dt\ .$$

В этой реализации оба аргумента *a* и *x* должны быть положительными. Интеграл вычисляется либо путем разложения в степенной ряд или непрерывную дробь, в зависимости от относительных значений *a* и *x* [5].

Дополнительная неполная гамма-функция определяется как **igamc(a,x) = 1 – igam(a,x)**. При некоторых значениях аргументов *a* и *x* ее можно именно так и вычислять. Однако в некоторых случаях, когда значение **igam(a,x)** велико, программа может возвращать значение **igamc(a,x)** равное нулю, а это не всегда допустимо. Результат вычисления этой функции должен быть всегда хоть и небольшим, но положительным. Поэтому ее вычисляют точно также, как и неполную гамма-функцию, путем разложения в степенной ряд или непрерывную дробь, в зависимости от относительных значений *a* и *x*, по формуле

$$igamc(a,x) = \frac{1}{\Gamma(a)}\int_0^x t^{-t}e^{a-1}dt \ .$$

При этом, как и в предыдущем случае, оба аргумента *a* и *x* должны быть положительными.

Натуральный логарифм от гамма-функции **lngamma(x)** вычисляется с помощью аппроксимирующей формулы [6]

$$\ln(gamma(x)) = (x+0{,}5)*\ln(x+5{,}5) - (x+5{,}5) + \ln(C_0(C_1 + C_2/(x+1) + C_3/(x+2) + \ldots + C_7/(x+8))/x) \ .$$

Значения коэффициентов C_k являются табличными данными, и содержаться в тексте программного кода.

Обратная неполная гамма-функция **invigam(a, x)** ищет такое значение *x*, для которого **igam(a, x)** равна *p*, т.е., вероятность того, что случайная величина, подчиняющаяся центральному гамма-распределению с параметром *a*, меньше или равна *x*. Значение этой функции необходимо для вычисления квантилей распределения χ^2 [5].

$$\chi^2_{a,x} = 2*invigam\ (0.5*a,x) \ .$$

Описанный модуль может быть включен либо в консольный проект, либо в проект, содержащий форму для вызова результатов выполнения содержащихся в нем процедур. Например, в случае консольного приложения, это будет выглядеть так:

```
static void Main()
{
  Console.Write(“ a = “); var sa =
Console.ReadLine(); double a = Dou-
ble.Parse(sa);
  Console.Write(“ x = “); var sx =
Console.ReadLine(); double q = Dou-
ble.Parse(sx);
  Console.WriteLine(“ a = {0}, q =
{1}”, a, q); Console.WriteLine();
  SpFunct igm = new SpFunct(); double
c = igm.igam(a, q);
  Console.WriteLine(“ Igam(x) = {0}”,
c);
  SpFunct igmc = new SpFunct(); c =
igmc.igamc(a, q);
  Console.WriteLine(“ Igamc(x) = {0}”,
c);
  SpFunct kv = new SpFunct(); c = 2 *
kv.invigam(a / 2.0, q);
  Console.WriteLine(“ KvHiKv(x) =
{0}”, c);
  Console.ReadKey();
}
```

Выводы

Задача, которую ставили перед собой авторы, сводилась к созданию инструментального средства для расчета некоторых, часто вычисляемых в процессе статистического моделирования, специальных функций. В настоящее время известно множество способов их расчета и программных реализаций этих способов. Здесь приводится наиболее компактный, на наш взгляд, вариант программного кода, созданный в среде Visual Studio.NET. Испытания предлагаемого модуля показали, что результаты вычислений значений гамма-функции, дополнительной гамма-функции, неполной дополнительной гамма-функции и квартилей распределения χ^2, не отличаются от результатов, которые дают вычисления с помощью таких пакетов как Matlab, Mathematica, Mathcad, Statistica и Excel.

Список использованных источников

1. A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications. NIST Special Publication 800-22. May 15, 2001.

2. The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness//http://www.stat.fsu.edu/pub/diehard/

3. Д. Кнут. Искусство программирования для ЭВМ. Т.2. – М.: Мир, 1977. – 727 с.

4. Statistical test suite Crypt-X //http://www.isi.qut.edu.au/resources/cryptx/

5. Попов Б. А., Теслер Г. С. Жанр: Справочник. Издательство: Наукова думка, 1984. – 600 с.

6. Абрамовиц М. Справочник по специальным функциям / М. Абрамовиц, И. Стиган. – М.: Наука, 1979. – 486 с.

Поступила в редакцию 18.11.2016

Рецензент: д.т.н., профессор Скачков В. В., Одесская государственная академия технического регулирования и качества, г. Одесса.

Ю. В. Щербина, к.т.н., **С. Л. Волков,** к.т.н., **Н. Ф. Казакова,** д.т.н.

ОБЧИСЛЕННЯ МОВОЮ С # СПЕЦІАЛЬНИХ ФУНКЦІЙ *igam*(*a*, *x*) ТА *igamc*(*a*, *x*) В СТАТИСТИЧНОМУ МОДЕЛЮВАННІ

Запропоновано компактний варіант обчислення на мові C# *таких спеціальних функцій як додаткова гама-функція, неповна додаткова гама-функція, натуральний логарифм від гама-функції та квантелі розподілення хі-квадрат, що використовуються під час вирішення задач статистичного моделювання.*

Ключові слова*: спеціальні функції, обчислювальні алгоритми, статистичне моделювання.*

Y. Scherbina, PhD, **S. Volkov,** PhD, **N. Kazakova,** DSc

CALCULATION IN C # SPECIAL FUNCTIONS *igam*(*a*, *x*) AND *igamc*(*a*, *x*) FOR THE NEEDS OF STATISTICAL MODELING

A compact version of the calculations on the C # language such special functions such as: additional gamma function, incomplete gamma function additional, natural logarithm of the gamma function and the quantile of chi-square distribution used when solving problems of statistical modeling.

Keywords*: special functions, numerical algorithms, statistical modeling.*

УДК 621.396.6:621.317

H. D. Bratchenko, DSc, **H. H. Smagliuk**, **D. V. Grygoriev**

Odesa State Academy of Technical Regulation and Quality, c. Odesa

METHOD FOR ISAR IMAGING OBJECTS WITH 3D ROTATIONAL MOTION

The influence of random components of spatial target movement on phase dependencies of signals from different scatterers is analyzed. Imaging method, which along with the regular trajectory component of target movement takes into account the random spatial components of rotational motion, is developed. These components are measured by phase changes of bright points at radar image sequence obtained by the discrete Fourier transform, coordinates of points being estimated using parametric spectral analysis methods.

Keywords: *radar high range resolution profile, ISAR imaging, cross-range, radar image, 3D rotational motion.*

Introduction

One of the ways of increasing the reliability of object recognition is to find and use highly informative signal features. Employment of wideband probing signals by monostatic active radar permits to resolve the elements of targets along the line-of-sight (LOS) and observe their high range resolution profiles (HRRP). The so called cross-range profiles are similar to HRRP but unlike the latter they show the location of the radar scatterers projected onto the line perpendicular to the LOS. Such profiles may actually be considered as one-dimensional radar images of targets.

However if an aerial target is observed from its side (aspect angles of 70º ... 110º), its HRRP becomes uninformative because fuselage obstructs the view of some its elements [1-3]. Additionally, the jet-engine modulation signatures are almost undetectable [4, 5]. In such circumstances, the interest shifts to the two-dimensional (2D) radar images, for which the side observation aspects are the most favorable [1]. In works by H. Safronov [6], J. Zinoviev and A. Pasmurov [7, 8], D.R. Wehner [9], A. Rihaczek and S. Hershkowitz [10], B. Steinberg [11], M. Prickett and C. Chen [12], V. Chen and H. Ling [13], V. Chen and M. Martorella [14] and by others the theoretical framework and examples of inverse synthetics aperture radar (ISAR) imaging of aerial objects have been given, including data on field experiments [15-17 and other]. Examples of ISAR imaging by Beijing Institute of Radio Measurement in C-band (4-8 GHz) and by Japanese researchers for radar operation frequency of 9.65 GHz were presented in [15, 16]. They illustrate a possibility of ISAR images being successfully obtained for air targets.

Some variants of solving the problem of ISAR imaging in case of non-uniform rotation of the object around single axis were proposed in [5, 18]. However, for more complex conditions of target flight in a turbulent atmosphere the decision hasn't been received. Irregular spatial movement of the