NAG C Library Function Document

nag_rngs_compd_poisson (g05mec)

1 Purpose

nag_rngs_compd_poisson (g05mec) generates a vector of pseudo-random integers, each from a discrete Poisson distribution with differing parameter λ.

2 Specification

```
void nag_rngs_compd_poisson (Integer m, const double vlamda[], Integer x[],
     Integer igen, Integer iseed[], NagError *fail)
```

3 Description

nag_rngs_compd_poisson (g05mec) generates m integers x_j, each from a discrete Poisson distribution with mean λ_j, where the probability of $x_j = I$ is

$$P(x_j = I) = \frac{\lambda_j^I \times e^{-\lambda_j}}{I!}, \quad I = 0, 1, \ldots,$$

where

$$0 \leq \lambda_j, \quad j = 1, 2, \ldots, m.$$

The methods used by this function have low set up times and are designed for efficient use when the value of the parameter λ changes during the simulation. For large samples from a distribution with fixed λ using nag_rngs_poisson (g05mkc) to set up and use a reference vector may be more efficient.

When $\lambda < 7.5$ the product of uniforms method is used, see for example Dagpunar (1988). For larger values of λ an envelope rejection method is used with a target distribution:

$$f(x) = \tfrac{1}{3} \qquad \text{if } |x| \leq 1,$$

$$f(x) = \tfrac{1}{3}|x|^{-3} \quad \text{otherwise.}$$

This distribution is generated using a ratio of uniforms method. A similar approach has also been suggested by Ahrens and Dieter (1989). The basic method is combined with quick acceptance and rejection tests given by Maclaren (1990). For values of $\lambda \geq 87$ Stirling's approximation is used in the computation of the Poisson distribution function, otherwise tables of factorials are used as suggested by Maclaren (1990).

One of the initialisation functions nag_rngs_init_repeatable (g05kbc) (for a repeatable sequence if computed sequentially) or nag_rngs_init_nonrepeatable (g05kcc) (for a non-repeatable sequence) must be called prior to the first call to nag_rngs_compd_poisson (g05mec).

4 References

Ahrens J H and Dieter U (1989) A convenient sampling method with bounded computation times for Poisson distributions *Amer. J. Math. Management Sci.* 1–13

Dagpunar J (1988) *Principles of Random Variate Generation* Oxford University Press

Maclaren N M (1990) A Poisson random number generator *Personal Communication*

5 Parameters

1: **m** – Integer *Input*

On entry: the number, m, of Poisson distributions for which pseudo-random variates are required.

Constraint: $\mathbf{m} \geq 1$.

2: **vlamda**[**m**] – const double *Input*

On entry: the means, λ_j, for $j = 1, 2, \ldots, \mathbf{m}$, of the Poisson distributions.

Constraint: $0.0 \leq \mathbf{vlamda}[j] \leq maxint/2$, where $maxint$ is the largest integer representable on the machine.

3: **x**[**m**] – Integer *Output*

On exit: the m pseudo-random numbers from the specified Poisson distributions.

4: **igen** – Integer *Input*

On entry: must contain the identification number for the generator to be used to return a pseudo-random number and should remain unchanged following initialisation by a prior call to one of the functions nag_rngs_init_repeatable (g05kbc) or nag_rngs_init_nonrepeatable (g05kcc).

5: **iseed**[4] – Integer *Input/Output*

On entry: contains values which define the current state of the selected generator.

On exit: contains updated values defining the new state of the selected generator.

6: **fail** – NagError * *Input/Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

On entry, $\mathbf{m} = \langle value \rangle$.
Constraint: $\mathbf{m} \geq 1$.

NE_REAL_ARRAY_ELEM_CONS

On entry, $2 \times \mathbf{vlamda}[i-1] > maxint$ for at least one value of $i = 1, 2, \ldots, \mathbf{m}$.

On entry, at least one element of **vlamda** < 0.0.

NE_BAD_PARAM

On entry, parameter $\langle value \rangle$ had an illegal value.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please consult NAG for assistance.

7 Accuracy

Not applicable.

8 Further Comments

None.

9 Example

The example program prints ten pseudo-random integers from five Poisson distributions with means $\lambda_1 = 0.5$, $\lambda_2 = 5$, $\lambda_3 = 10$, $\lambda_4 = 50$ and $\lambda_5 = 100$. These are generated by ten calls to nag_rngs_compd_poisson (g05mec), after initialisation by nag_rngs_init_repeatable (g05kbc).

9.1 Program Text

```
/* nag_rngs_compd_poisson(g05mec) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg05.h>

int main(void)
{
  /* Scalars */
  Integer  i, igen, j, m, n;
  Integer  exit_status=0;
  NagError fail;

  /* Arrays */
  double   *vlamda=0;
  Integer  *x=0;
  Integer  iseed[4];

  INIT_FAIL(fail);
  Vprintf("g05mec Example Program Results\n\n");
  m = 5;
  n = 10;

  /* Allocate memory */
  if ( !(vlamda = NAG_ALLOC(m, double)) ||
       !(x = NAG_ALLOC(m, Integer)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Set the distribution parameter LAMBDA */
  vlamda[0] = 0.5;
  vlamda[1] = 5.0;
  vlamda[2] = 10.0;
  vlamda[3] = 500.0;
  vlamda[4] = 1e3;
  /* Initialise the seed to a repeatable sequence */
  iseed[0] = 1762543;
  iseed[1] = 9324783;
  iseed[2] = 423442;
  iseed[3] = 742355;
  /* igen identifies the stream. */
  igen = 1;
  g05kbc(&igen, iseed);

  /* Generate integers and store in X */
  for (i = 0; i < n; ++i)
    {
      g05mec(m, vlamda, x, igen, iseed, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from g05mec.\n%s\n", fail.message);
          exit_status = 1;
```

```
          goto END;
        }
      for (j = 0; j < m; ++j)
        {
          Vprintf("%12ld%s", x[j], (j+1)%5 == 0 || j == 4 ?"\n":" ");
        }
    }
 END:
  if (vlamda) NAG_FREE(vlamda);
  if (x) NAG_FREE(x);
  return exit_status;
}
```

9.2 Program Data

None.

9.3 Program Results

```
g05mec Example Program Results

           1            3           13          482         1001
           1            4           12          494         1046
           1            2           11          548          941
           0            6            8          518          977
           0            6            8          504          943
           1            6            8          502          991
           0           11            7          475          991
           1            4            5          507         1012
           0            4           13          537         1016
           0            4            7          492         1072
```