

## NAG Library Function Document

### nag\_rand\_compdpoisson (g05tkc)

#### 1 Purpose

nag\_rand\_compdpoisson (g05tkc) generates a vector of pseudorandom integers, each from a discrete Poisson distribution with differing parameter.

#### 2 Specification

```
#include <nag.h>
#include <nagg05.h>

void nag_rand_compdpoisson (Integer m, const double vlamda[],
    Integer state[], Integer x[], NagError *fail)
```

#### 3 Description

nag\_rand\_compdpoisson (g05tkc) generates  $m$  integers  $x_j$ , each from a discrete Poisson distribution with mean  $\lambda_j$ , where the probability of  $x_j = I$  is

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

where

$$\lambda_j \geq 0, \quad j = 1, 2, \dots, m.$$

The methods used by this function have low set up times and are designed for efficient use when the value of the parameter  $\lambda$  changes during the simulation. For large samples from a distribution with fixed  $\lambda$  using nag\_rand\_poisson (g05tjc) 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  $\lambda$  an envelope rejection method is used with a target distribution:

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

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

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 initialization functions nag\_rand\_init\_repeatable (g05kfc) (for a repeatable sequence if computed sequentially) or nag\_rand\_init\_nonrepeatable (g05kge) (for a non-repeatable sequence) must be called prior to the first call to nag\_rand\_compdpoisson (g05tkc).

#### 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 Arguments

- 1: **m** – Integer *Input*  
*On entry:*  $m$ , the number of Poisson distributions for which pseudorandom variates are required.  
*Constraint:*  $m \geq 1$ .
- 2: **vlambda[m]** – const double *Input*  
*On entry:* the means,  $\lambda_j$ , for  $j = 1, 2, \dots, m$ , of the Poisson distributions.  
*Constraint:*  $0.0 \leq \mathbf{vlambda}[j - 1] \leq \text{nag\_max\_integer}/2.0$ , for  $j = 1, 2, \dots, m$ .
- 3: **state[dim]** – Integer *Communication Array*  
**Note:** the dimension,  $dim$ , of this array is dictated by the requirements of associated functions that must have been previously called. This array **MUST** be the same array passed as argument **state** in the previous call to `nag_rand_init_repeatable` (g05kfc) or `nag_rand_init_nonrepeatable` (g05kge).  
*On entry:* contains information on the selected base generator and its current state.  
*On exit:* contains updated information on the state of the generator.
- 4: **x[m]** – Integer *Output*  
*On exit:* the  $m$  pseudorandom numbers from the specified Poisson distributions.
- 5: **fail** – NagError \* *Input/Output*  
The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

See Section 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

### NE\_BAD\_PARAM

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

### NE\_INT

On entry,  $m = \langle value \rangle$ .

Constraint:  $m \geq 1$ .

### 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 contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.

See Section 3.6.6 in How to Use the NAG Library and its Documentation for further information.

### NE\_INVALID\_STATE

On entry, **state** vector has been corrupted or not initialized.

**NE\_NO\_LICENCE**

Your licence key may have expired or may not have been installed correctly.  
See Section 3.6.5 in How to Use the NAG Library and its Documentation for further information.

**NE\_REAL\_ARRAY**

On entry, at least one element of **vlambda** is less than zero.

On entry, at least one element of **vlambda** is too large.

**7 Accuracy**

Not applicable.

**8 Parallelism and Performance**

nag\_rand\_compdpoisson (g05tkc) is not threaded in any implementation.

**9 Further Comments**

None.

**10 Example**

This example prints ten pseudorandom integers from five Poisson distributions with means  $\lambda_1 = 0.5$ ,  $\lambda_2 = 5$ ,  $\lambda_3 = 10$ ,  $\lambda_4 = 500$  and  $\lambda_5 = 1000$ . These are generated by ten calls to nag\_rand\_compdpoisson (g05tkc), after initialization by nag\_rand\_initrepeatable (g05kfc).

**10.1 Program Text**

```

/* nag_rand_compdpoisson (g05tkc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg05.h>

int main(void)
{
    /* Integer scalar and array declarations */
    Integer exit_status = 0;
    Integer i, j, lstate;
    Integer *state = 0, *x = 0;

    /* NAG structures */
    NagError fail;

    /* Set the distribution parameters */
    Integer m = 5;
    double lambda[] = { 0.5e0, 5.0e0, 10.0e0, 500.0e0, 1000.0e0 };

    /* Set the sample size */
    Integer n = 10;

    /* Choose the base generator */
    Nag_BaseRNG genid = Nag_Basic;

```

```

Integer subid = 0;

/* Set the seed */
Integer seed[] = { 1762543 };
Integer lseed = 1;

/* Initialize the error structure */
INIT_FAIL(fail);

printf("nag_rand_compdpoisson (g05tkc) Example Program Results\n\n");

/* Get the length of the state array */
lstate = -1;
nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_rand_init_repeatable (g05tkc).\n%s\n",
        fail.message);
    exit_status = 1;
    goto END;
}

/* Allocate arrays */
if (!(state = NAG_ALLOC(lstate, Integer)) || !(x = NAG_ALLOC(m, Integer)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Initialize the generator to a repeatable sequence */
nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_rand_init_repeatable (g05tkc).\n%s\n",
        fail.message);
    exit_status = 1;
    goto END;
}

/* Generate and display n sets of the m variates */
for (i = 0; i < n; i++) {
    /* Generate the variates */
    nag_rand_compdpoisson(m, lambda, state, x, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_rand_compdpoisson (g05tkc).\n%s\n",
            fail.message);
        exit_status = 1;
        goto END;
    }

    /* Display the variates */
    printf("%12" NAG_IFMT " ", i + 1);
    for (j = 0; j < m; j++)
        printf("%12" NAG_IFMT " ", x[j]);
    printf("\n");
}

END:
NAG_FREE(state);
NAG_FREE(x);

return exit_status;
}

```

## 10.2 Program Data

None.

### 10.3 Program Results

nag\_rand\_compd\_poisson (g05tkc) Example Program Results

1	1	6	12	507	1003
2	0	9	11	520	1028
3	1	3	7	483	1041
4	0	3	11	513	1012
5	1	5	9	496	940
6	0	6	17	548	990
7	1	9	8	512	1035
8	0	4	10	458	1029
9	1	6	13	523	971
10	0	9	16	519	999

---