NAG Library Function Document

nag_prob_binomial_vector (g01sjc)

1 Purpose

nag_prob_binomial_vector (g01sjc) returns a number of the lower tail, upper tail and point probabilities for the binomial distribution.

2 Specification

```
#include <nag.h>
#include <nagg01.h>
void nag_prob_binomial_vector (Integer ln, const Integer n[], Integer lp,
     const double p[], Integer lk, const Integer k[], double plek[],
     double pgtk[], double peqk[], Integer ivalid[], NagError *fail)
```

3 Description

Let $X = \{X_i : i = 1, 2, \ldots, m\}$ denote a vector of random variables each having a binomial distribution with parameters n_i and p_i ($n_i \geq 0$ and $0 < p_i < 1$). Then

$$\text{Prob}\{X_i = k_i\} = \binom{n_i}{k_i} {p_i}^{k_i} (1 - p_i)^{n_i - k_i}, \quad k_i = 0, 1, \ldots, n_i.$$

The mean of the each distribution is given by $n_i p_i$ and the variance by $n_i p_i (1 - p_i)$.

nag_prob_binomial_vector (g01sjc) computes, for given n_i, p_i and k_i, the probabilities: Prob$\{X_i \leq k_i\}$, Prob$\{X_i > k_i\}$ and Prob$\{X_i = k_i\}$ using an algorithm similar to that described in KnÏsel (1986) for the Poisson distribution.

The input arrays to this function are designed to allow maximum flexibility in the supply of vector arguments by re-using elements of any arrays that are shorter than the total number of evaluations required. See Section 2.6 in the g01 Chapter Introduction for further information.

4 References

KnÏsel L (1986) Computation of the chi-square and Poisson distribution *SIAM J. Sci. Statist. Comput.* **7** 1022–1036

5 Arguments

1: **ln** – Integer *Input*

On entry: the length of the array **n**.

Constraint: **ln** > 0.

2: **n**[**ln**] – const Integer *Input*

On entry: n_i, the first parameter of the binomial distribution with $n_i =$ **n**$[j]$, $j = (i - 1) \bmod$ **ln**, for $i = 1, 2, \ldots, \max($**ln**, **lp**, **lk**$)$.

Constraint: **n**$[j - 1] \geq 0$, for $j = 1, 2, \ldots,$ **ln**.

3: **lp** – Integer *Input*

On entry: the length of the array **p**.

Constraint: **lp** > 0.

4: **p**[**lp**] – const double *Input*

On entry: p_i, the second parameter of the binomial distribution with $p_i = \mathbf{p}[j]$, $j = (i-1) \bmod \mathbf{lp}$.

Constraint: $0.0 < \mathbf{p}[j-1] < 1.0$, for $j = 1, 2, \ldots, \mathbf{lp}$.

5: **lk** – Integer *Input*

On entry: the length of the array **k**.

Constraint: **lk** > 0.

6: **k**[**lk**] – const Integer *Input*

On entry: k_i, the integer which defines the required probabilities with $k_i = \mathbf{k}[j]$, $j = (i-1) \bmod \mathbf{lk}$.

Constraint: $0 \leq k_i \leq n_i$.

7: **plek**[dim] – double *Output*

Note: the dimension, *dim*, of the array **plek** must be at least $\max(\mathbf{ln}, \mathbf{lp}, \mathbf{lk})$.

On exit: $\text{Prob}\{X_i \leq k_i\}$, the lower tail probabilities.

8: **pgtk**[dim] – double *Output*

Note: the dimension, *dim*, of the array **pgtk** must be at least $\max(\mathbf{ln}, \mathbf{lp}, \mathbf{lk})$.

On exit: $\text{Prob}\{X_i > k_i\}$, the upper tail probabilities.

9: **peqk**[dim] – double *Output*

Note: the dimension, *dim*, of the array **peqk** must be at least $\max(\mathbf{ln}, \mathbf{lp}, \mathbf{lk})$.

On exit: $\text{Prob}\{X_i = k_i\}$, the point probabilities.

10: **ivalid**[dim] – Integer *Output*

Note: the dimension, *dim*, of the array **ivalid** must be at least $\max(\mathbf{ln}, \mathbf{lp}, \mathbf{lk})$.

On exit: **ivalid**$[i-1]$ indicates any errors with the input arguments, with

ivalid$[i-1] = 0$
No error.

ivalid$[i-1] = 1$

On entry, $n_i < 0$.

ivalid$[i-1] = 2$

On entry, $p_i \leq 0.0$,
or $p_i \geq 1.0$.

ivalid$[i-1] = 3$

On entry, $k_i < 0$,
or $k_i > n_i$.

ivalid$[i-1] = 4$

On entry, n_i is too large to be represented exactly as a real number.

ivalid$[i-1] = 5$

On entry, the variance ($= n_i p_i(1-p_i)$) exceeds 10^6.

11: **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 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_ARRAY_SIZE

On entry, array size $= \langle value \rangle$.
Constraint: **lk** > 0.

On entry, array size $= \langle value \rangle$.
Constraint: **ln** > 0.

On entry, array size $= \langle value \rangle$.
Constraint: **lp** > 0.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ 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 contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NW_IVALID

On entry, at least one value of **n**, **p** or **k** was invalid.
Check **ivalid** for more information.

7 Accuracy

Results are correct to a relative accuracy of at least 10^{-6} on machines with a precision of 9 or more decimal digits, and to a relative accuracy of at least 10^{-3} on machines of lower precision (provided that the results do not underflow to zero).

8 Parallelism and Performance

nag_prob_binomial_vector (g01sjc) is not threaded in any implementation.

9 Further Comments

The time taken by nag_prob_binomial_vector (g01sjc) to calculate each probability depends on the variance ($= n_i p_i(1-p_i)$) and on k_i. For given variance, the time is greatest when $k_i \approx n_i p_i$ ($=$ the mean), and is then approximately proportional to the square-root of the variance.

10 Example

This example reads a vector of values for n, p and k, and prints the corresponding probabilities.

10.1 Program Text

```
/* nag_prob_binomial_vector (g01sjc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg01.h>

int main(void)
{
  /* Integer scalar and array declarations */
  Integer lk, lp, ln, i, lout;
  Integer *ivalid = 0, *k = 0, *n = 0;
  Integer exit_status = 0;

  /* NAG structures */
  NagError fail;

  /* Double scalar and array declarations */
  double *peqk = 0, *pgtk = 0, *plek = 0, *p = 0;

  /* Initialize the error structure to print out any error messages */
  INIT_FAIL(fail);

  printf("nag_prob_binomial_vector (g01sjc) Example Program Results\n\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n] ", &ln);
#else
  scanf("%" NAG_IFMT "%*[^\n] ", &ln);
#endif
  if (!(n = NAG_ALLOC(ln, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  for (i = 0; i < ln; i++)
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &n[i]);
#else
    scanf("%" NAG_IFMT "", &n[i]);
#endif
#ifdef _WIN32
```

```
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n] ", &lp);
#else
  scanf("%" NAG_IFMT "%*[^\n] ", &lp);
#endif
  if (!(p = NAG_ALLOC(lp, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  for (i = 0; i < lp; i++)
#ifdef _WIN32
    scanf_s("%lf", &p[i]);
#else
    scanf("%lf", &p[i]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n] ", &lk);
#else
  scanf("%" NAG_IFMT "%*[^\n] ", &lk);
#endif
  if (!(k = NAG_ALLOC(lk, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  for (i = 0; i < lk; i++)
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &k[i]);
#else
    scanf("%" NAG_IFMT "", &k[i]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Allocate memory for output */
  lout = MAX(ln, MAX(lp, lk));
  if (!(peqk = NAG_ALLOC(lout, double)) ||
      !(pgtk = NAG_ALLOC(lout, double)) ||
      !(plek = NAG_ALLOC(lout, double)) ||
      !(ivalid = NAG_ALLOC(lout, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Calculate probability */
  nag_prob_binomial_vector(ln, n, lp, p, lk, k,
                           plek, pgtk, peqk, ivalid, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_prob_binomial_vector (g01sjc).\n%s\n",
           fail.message);
    exit_status = 1;
    if (fail.code != NW_IVALID)
```

```
      goto END;
  }

  /* Display title */
  printf("     n         p         k       ");
  printf("plek      pgtk      peqk  ivalid\n");
  printf(" ---------------------------------");
  printf("---------------------------------\n");

  /* Display results */
  for (i = 0; i < lout; i++) {
    printf(" %6" NAG_IFMT "    %6.3f    %6" NAG_IFMT "",
           n[i % ln], p[i % lp], k[i % lk]);
    printf("    %6.3f    %6.3f    %6.3f   %3" NAG_IFMT "\n",
           plek[i], pgtk[i], peqk[i], ivalid[i]);
  }
END:
  NAG_FREE(n);
  NAG_FREE(p);
  NAG_FREE(k);
  NAG_FREE(plek);
  NAG_FREE(pgtk);
  NAG_FREE(peqk);
  NAG_FREE(ivalid);

  return (exit_status);
}
```

10.2 Program Data

```
nag_prob_binomial_vector (g01sjc) Example Program Data
4                                                                    :: ln
4 19 100 2000                                                        :: n
4                                                                    :: lp
0.500 0.440 0.750 0.330                                              :: p
4                                                                    :: lk
2 13 67 700                                                          :: k
```

10.3 Program Results

```
nag_prob_binomial_vector (g01sjc) Example Program Results

     n         p         k       plek      pgtk      peqk  ivalid
 ---------------------------------------------------------------------
     4     0.500         2     0.688     0.312     0.375     0
    19     0.440        13     0.991     0.009     0.019     0
   100     0.750        67     0.045     0.955     0.017     0
  2000     0.330       700     0.973     0.027     0.003     0
```