NAG Library Function Document

nag_prob_gamma_vector (g01sfc)

1 Purpose

nag_prob_gamma_vector (g01sfc) returns a number of lower or upper tail probabilities for the gamma distribution.

2 Specification

```
#include <nag.h>
#include <nagg01.h>
void nag_prob_gamma_vector (Integer ltail, const Nag_TailProbability tail[],
     Integer lg, const double g[], Integer la, const double a[], Integer lb,
     const double b[], double p[], Integer ivalid[], NagError *fail)
```

3 Description

The lower tail probability for the gamma distribution with parameters α_i and β_i, $P(G_i \leq g_i)$, is defined by:

$$P\left(G_i \leq g_i : \alpha_i, \beta_i\right) = \frac{1}{{\beta_i}^{\alpha_i}\Gamma(\alpha_i)}\int_0^{g_i} {G_i}^{\alpha_i - 1} e^{-G_i/\beta_i}\, dG_i, \quad \alpha_i > 0.0,\ \beta_i > 0.0.$$

The mean of the distribution is $\alpha_i\beta_i$ and its variance is $\alpha_i{\beta_i}^2$. The transformation $Z_i = \dfrac{G_i}{\beta_i}$ is applied to yield the following incomplete gamma function in normalized form,

$$P\left(G_i \leq g_i : \alpha_i, \beta_i\right) = P\left(Z_i \leq g_i/\beta_i : \alpha_i, 1.0\right) = \frac{1}{\Gamma(\alpha_i)}\int_0^{g_i/\beta_i} {Z_i}^{\alpha_i - 1} e^{-Z_i}\, dZ_i.$$

This is then evaluated using nag_incomplete_gamma (s14bac).

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

Hastings N A J and Peacock J B (1975) *Statistical Distributions* Butterworth

5 Arguments

1: **ltail** – Integer *Input*

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

Constraint: **ltail** > 0.

2: **tail**[**ltail**] – const Nag_TailProbability *Input*

On entry: indicates whether a lower or upper tail probability is required. For $j = (i-1) \bmod$ **ltail**, for $i = 1, 2, \ldots, \max($**ltail**, **lg**, **la**, **lb**$)$:

tail$[j] =$ Nag_LowerTail
The lower tail probability is returned, i.e., $p_i = P\left(G_i \leq g_i : \alpha_i, \beta_i\right)$.

tail$[j]$ = Nag_UpperTail
The upper tail probability is returned, i.e., $p_i = P\left(G_i \geq g_i : \alpha_i, \beta_i\right)$.

Constraint: **tail**$[j-1]$ = Nag_LowerTail or Nag_UpperTail, for $j = 1, 2, \ldots,$ **ltail**.

3: **lg** – Integer *Input*

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

Constraint: **lg** > 0.

4: **g**[**lg**] – const double *Input*

On entry: g_i, the value of the gamma variate with $g_i =$ **g**$[j]$, $j = (i-1) \bmod$ **lg**.

Constraint: **g**$[j-1] \geq 0.0$, for $j = 1, 2, \ldots,$ **lg**.

5: **la** – Integer *Input*

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

Constraint: **la** > 0.

6: **a**[**la**] – const double *Input*

On entry: the parameter α_i of the gamma distribution with $\alpha_i =$ **a**$[j]$, $j = (i-1) \bmod$ **la**.

Constraint: **a**$[j-1] > 0.0$, for $j = 1, 2, \ldots,$ **la**.

7: **lb** – Integer *Input*

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

Constraint: **lb** > 0.

8: **b**[**lb**] – const double *Input*

On entry: the parameter β_i of the gamma distribution with $\beta_i =$ **b**$[j]$, $j = (i-1) \bmod$ **lb**.

Constraint: **b**$[j-1] > 0.0$, for $j = 1, 2, \ldots,$ **lb**.

9: **p**$[dim]$ – double *Output*

Note: the dimension, *dim*, of the array **p** must be at least max(**lg**, **la**, **lb**, **ltail**).

On exit: p_i, the probabilities of the beta distribution.

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

Note: the dimension, *dim*, of the array **ivalid** must be at least max(**lg**, **la**, **lb**, **ltail**).

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, invalid value supplied in **tail** when calculating p_i.

ivalid$[i-1] = 2$
On entry, $g_i < 0.0$.

ivalid$[i-1] = 3$
On entry, $\alpha_i \leq 0.0$,
or $\beta_i \leq 0.0$.

ivalid$[i-1] = 4$
The solution did not converge in 600 iterations, see nag_incomplete_gamma (s14bac) for details. The probability returned should be a reasonable approximation to the solution.

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

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_ARRAY_SIZE

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

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

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

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

NW_IVALID

On entry, at least one value of **g**, **a**, **b** or **tail** was invalid, or the solution did not converge. Check **ivalid** for more information.

7 Accuracy

The result should have a relative accuracy of ***machine precision***. There are rare occasions when the relative accuracy attained is somewhat less than ***machine precision*** but the error should not exceed more than 1 or 2 decimal places.

8 Parallelism and Performance

Not applicable.

9 Further Comments

The time taken by nag_prob_gamma_vector (g01sfc) to calculate each probability varies slightly with the input arguments g_i, α_i and β_i.

10 Example

This example reads in values from a number of gamma distributions and computes the associated lower tail probabilities.

10.1 Program Text

```
/* nag_prob_gamma_vector (g01sfc) Example Program.
 *
 * Copyright 2011, Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg01.h>

int main(void)
{
  /* Integer scalar and array declarations */
  Integer ltail, lg, la, lb, i, lout;
  Integer *ivalid = 0;
  Integer exit_status = 0;

  /* NAG structures */
  NagError fail;
  Nag_TailProbability *tail = 0;

  /* Double scalar and array declarations */
  double *g = 0, *a = 0, *b = 0, *p = 0;

  /* Character scalar and array declarations */
  char ctail[40];

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

  printf("nag_prob_gamma_vector (g01sfc) Example Program Results\n\n");

  /* Skip heading in data file*/
  scanf("%*[^\n] ");

  /* Read in the input vectors */
  scanf("%ld%*[^\n] ", &ltail);
  if (!(tail = NAG_ALLOC(ltail, Nag_TailProbability))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  for (i = 0; i < ltail; i++) {
    scanf("%39s", ctail);
    tail[i] = (Nag_TailProbability) nag_enum_name_to_value(ctail);
  }
  scanf("%*[^\n] ");
  scanf("%ld%*[^\n] ", &lg);
  if (!(g = NAG_ALLOC(lg, double))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  for (i = 0; i < lg; i++)
    scanf("%lf", &g[i]);
  scanf("%*[^\n] ");
  scanf("%ld%*[^\n] ", &la);
  if (!(a = NAG_ALLOC(la, double))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  for (i = 0; i < la; i++)
    scanf("%lf", &a[i]);
  scanf("%*[^\n] ");
  scanf("%ld%*[^\n] ", &lb);
  if (!(b = NAG_ALLOC(lb, double))) {
    printf("Allocation failure\n");
```

```
    exit_status = -1;
    goto END;
  }
  for (i = 0; i < lb; i++)
    scanf("%lf", &b[i]);
  scanf("%*[^\n] ");

  /* Allocate memory for output */
  lout = MAX(ltail,MAX(lg,MAX(la,lb)));
  if (!(p = NAG_ALLOC(lout, double)) ||
      !(ivalid = NAG_ALLOC(lout, Integer))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Calculate probability */
  nag_prob_gamma_vector(ltail, tail, lg, g, la, a, lb, b,
                        p, ivalid, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_prob_gamma_vector (g01sfc).\n%s\n",
            fail.message);
    exit_status = 1;
    if (fail.code != NW_IVALID) goto END;
  }

  /* Display title */
  printf("        tail           g          a         b        ");
  printf("p     ivalid\n");
  printf(" ------------------------------------------");
  printf("----------------------\n");

  /* Display results */
  for (i = 0; i < lout; i++)
    printf(" %15s    %6.2f    %6.2f    %6.2f    %6.3f    %3ld\n",
           nag_enum_value_to_name(tail[i%ltail]), g[i%lg], a[i%la], b[i%lb],
           p[i], ivalid[i]);

 END:
  NAG_FREE(tail);
  NAG_FREE(g);
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(p);
  NAG_FREE(ivalid);

  return(exit_status);
}
```

10.2 Program Data

```
nag_prob_gamma_vector (g01sfc) Example Program Data
1                                                                    :: ltail
Nag_LowerTail                                                        :: tail
4                                                                    :: lg
15.5 0.5 10.0 5.0                                                    :: g
4                                                                    :: la
4.0 4.0 1.0 2.0                                                      :: a
4                                                                    :: lb
2.0 1.0 2.0 2.0                                                      :: b
```

10.3 Program Results

```
nag_prob_gamma_vector (g01sfc) Example Program Results

        tail           g          a         b         p     ivalid
 ------------------------------------------------------------------
   Nag_LowerTail     15.50       4.00      2.00     0.950       0
   Nag_LowerTail      0.50       4.00      1.00     0.002       0
   Nag_LowerTail     10.00       1.00      2.00     0.993       0
   Nag_LowerTail      5.00       2.00      2.00     0.713       0
```