NAG Library Function Document

nag_gamma_pdf_vector (g01kkc)

1 Purpose

nag_gamma_pdf_vector (g01kkc) returns a number of values of the probability density function (PDF), or its logarithm, for the gamma distribution.

2 Specification

```
#include <nag.h>
#include <nagg01.h>
void nag_gamma_pdf_vector (Nag_Boolean ilog, Integer lx, const double x[],
     Integer la, const double a[], Integer lb, const double b[],
     double pdf[], Integer ivalid[], NagError *fail)
```

3 Description

The gamma distribution with shape parameter α_i and scale parameter β_i has PDF

$$f(x_i,\alpha_i,\beta_i) = \frac{1}{{\beta_i}^{\alpha_i}\Gamma(\alpha_i)}{x_i}^{\alpha_i-1}e^{-x_i/\beta_i} \quad \text{if } x_i \geq 0; \quad \alpha_i,\beta_i > 0$$

$$f(x_i,\alpha_i,\beta_i) = 0 \qquad \text{otherwise.}$$

If $0.01 \leq x_i,\alpha_i,\beta_i \leq 100$ then an algorithm based directly on the gamma distribution's PDF is used. For values outside this range, the function is calculated via the Poisson distribution's PDF as described in Loader (2000) (see Section 9).

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

Loader C (2000) Fast and accurate computation of binomial probabilities **(not yet published)**

5 Arguments

1: **ilog** – Nag_Boolean *Input*

On entry: the value of **ilog** determines whether the logarithmic value is returned in **pdf**.

ilog = Nag_FALSE
$f(x_i,\alpha_i,\beta_i)$, the probability density function is returned.

ilog = Nag_TRUE
$\log(f(x_i,\alpha_i,\beta_i))$, the logarithm of the probability density function is returned.

2: **lx** – Integer *Input*

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

Constraint: **lx** > 0.

3: **x**[**lx**] – const double *Input*

On entry: x_i, the values at which the PDF is to be evaluated with $x_i = \mathbf{x}[j]$, $j = (i-1) \bmod \mathbf{lx}$, for $i = 1, 2, \ldots, \max(\mathbf{lx}, \mathbf{la}, \mathbf{lb})$.

4: **la** – Integer *Input*

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

Constraint: **la** > 0.

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

On entry: α_i, the shape parameter with $\alpha_i = \mathbf{a}[j]$, $j = (i-1) \bmod \mathbf{la}$.

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

6: **lb** – Integer *Input*

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

Constraint: **lb** > 0.

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

On entry: β_i, the scale parameter with $\beta_i = \mathbf{b}[j]$, $j = (i-1) \bmod \mathbf{lb}$.

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

8: **pdf**[dim] – double *Output*

Note: the dimension, *dim*, of the array **pdf** must be at least $\max(\mathbf{lx}, \mathbf{la}, \mathbf{lb})$.

On exit: $f(x_i, \alpha_i, \beta_i)$ or $\log(f(x_i, \alpha_i, \beta_i))$.

9: **ivalid**[dim] – Integer *Output*

Note: the dimension, *dim*, of the array **ivalid** must be at least $\max(\mathbf{lx}, \mathbf{la}, \mathbf{lb})$.

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

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

ivalid$[i-1] = 1$
$\alpha_i \leq 0.0$.

ivalid$[i-1] = 2$
$\beta_i \leq 0.0$.

ivalid$[i-1] = 3$
$\frac{x_i}{\beta_i}$ overflows, the value returned should be a reasonable approximation.

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

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

6 Error Indicators and Warnings

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: **lx** > 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 **x**, **a** or **b** was invalid.
Check **ivalid** for more information.

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

Due to the lack of a stable link to Loader (2000) paper, we give a brief overview of the method, as applied to the Poisson distribution. The Poisson distribution has a continuous mass function given by,

$$p(x;\lambda) = \frac{\lambda^x}{x!}e^{-\lambda}. \tag{1}$$

The usual way of computing this quantity would be to take the logarithm and calculate,

$$\log\left(p(x;\lambda)\right) = x\log\lambda - \log\left(x!\right) - \lambda.$$

For large x and λ, $x\log\lambda$ and $\log\left(x!\right)$ are very large, of the same order of magnitude and when calculated have rounding errors. The subtraction of these two terms can therefore result in a number, many orders of magnitude smaller and hence we lose accuracy due to subtraction errors. For example for $x = 2 \times 10^6$ and $\lambda = 2 \times 10^6$, $\log\left(x!\right) \approx 2.7 \times 10^7$ and $\log\left(p(x;\lambda)\right) = -8.17326744645834$. But calculated with the method shown later we have $\log\left(p(x;\lambda)\right) = -8.1732674441334492$. The difference between these two results suggests a loss of about 7 significant figures of precision.

Loader introduces an alternative way of expressing (1) based on the saddle point expansion,

$$\log\left(p(x;\lambda)\right) = \log\left(p(x;x)\right) - D(x;\lambda), \tag{2}$$

where $D(x;\lambda)$, the deviance for the Poisson distribution is given by,

$$\begin{aligned} D(x;\lambda) &= \log\left(p(x;x)\right) - \log\left(p(x;\lambda)\right), \\ &= \lambda D_0\left(\tfrac{x}{\lambda}\right), \end{aligned} \tag{3}$$

and

$$D_0(\epsilon) = \epsilon\log\epsilon + 1 - \epsilon.$$

For ϵ close to 1, $D_0(\epsilon)$ can be evaluated through the series expansion

$$\lambda D_0\left(\frac{x}{\lambda}\right) = \frac{(x-\lambda)^2}{x+\lambda} + 2x\sum_{j=1}^{\infty}\frac{v^{2j+1}}{2j+1}, \qquad \text{where } v = \frac{x-\lambda}{x+\lambda},$$

otherwise $D_0(\epsilon)$ can be evaluated directly. In addition, Loader suggests evaluating $\log\left(x!\right)$ using the Stirling–De Moivre series,

$$\log(x!) = \frac{1}{2}\log(2\pi x) + x\log(x) - x + \delta(x), \tag{4}$$

where the error $\delta(x)$ is given by

$$\delta(x) = \frac{1}{12x} - \frac{1}{360x^3} + \frac{1}{1260x^5} + \mathcal{O}\left(x^{-7}\right).$$

Finally $\log(p(x;\lambda))$ can be evaluated by combining equations (1)–(4) to get,

$$p(x;\lambda) = \frac{1}{\sqrt{2\pi x}} e^{-\delta(x) - \lambda D_0(x/\lambda)}.$$

10 Example

This example prints the value of the gamma distribution PDF at six different points x_i with differing α_i and β_i.

10.1 Program Text

```
/* nag_gamma_pdf_vector (g01kkc) 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 lx, la, lb, i, lout;
  Integer *ivalid = 0;
  Integer exit_status = 0;

  /* NAG structures */
  NagError fail;
  Nag_Boolean ilog;

  /* Double scalar and array declarations */
  double *x = 0, *a = 0, *b = 0, *pdf = 0;

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

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

  printf("nag_gamma_pdf_vector (g01kkc) Example Program Results\n\n");

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

  /* Read in the flag indicating whether logs are required */
  scanf("%39s%*[^\n] ", cilog);
  ilog = (Nag_Boolean) nag_enum_name_to_value(cilog);

  /* Read in the input vectors */
  scanf("%ld%*[^\n] ", &lx);
  if (!(x = NAG_ALLOC(lx, double))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
```

```
  for (i = 0; i < lx; i++)
    scanf("%lf", &x[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(lx,MAX(la,lb));
  if (!(pdf = NAG_ALLOC(lout, double)) ||
      !(ivalid = NAG_ALLOC(lout, Integer))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Calculate probability */
  nag_gamma_pdf_vector(ilog,lx,x,la,a,lb,b,pdf,ivalid,&fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gamma_pdf_vector (g01kkc).\n%s\n",
            fail.message);
    exit_status = 1;
    if (fail.code != NW_IVALID) goto END;
  }

  /* Display title */
  printf("   x          a         b        pdf      ivalid\n");
  printf(" ------------------------------------------------\n");

  /* Display results */
  for (i = 0; i < lout; i++)
    printf("%6.2f    %6.2f    %6.2f    %9.3e    %3ld\n",
           x[i%lx], a[i%la], b[i%lb], pdf[i], ivalid[i]);

 END:
  NAG_FREE(x);
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(pdf);
  NAG_FREE(ivalid);

  return(exit_status);
}
```

10.2 Program Data

```
nag_gamma_pdf_vector (g01kkc) Example Program Data
Nag_FALSE                    :: ILOG
6                            :: LX
0.1 3.0 6.0 4.0 9.0 16.0     :: X
6                            :: LA
3.0 10.0 5.0 10.0 9.0 3.5    :: A
6                            :: LB
2.0 11.0 1.0 0.1 0.5 2.5     :: B
```

10.3 Program Results

```
nag_gamma_pdf_vector (g01kkc) Example Program Results

   x          a          b         pdf       ivalid
 -------------------------------------------------
  0.10       3.00       2.00    5.945e-04       0
  3.00      10.00      11.00    1.592e-12       0
  6.00       5.00       1.00    1.339e-01       0
  4.00      10.00       0.10    3.069e-08       0
  9.00       9.00       0.50    8.325e-03       0
 16.00       3.50       2.50    2.072e-02       0
```