NAG Library Function Document

nag_deviates_beta_vector (g01tec)

1 Purpose

nag_deviates_beta_vector (g01tec) returns a number of deviates associated with given probabilities of the beta distribution.

2 Specification

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

3 Description

The deviate, β_{p_i}, associated with the lower tail probability, p_i, of the beta distribution with parameters a_i and b_i is defined as the solution to

$$P\left(B_i \leq \beta_{p_i} : a_i, b_i\right) = p_i = \frac{\Gamma(a_i + b_i)}{\Gamma(a_i)\Gamma(b_i)} \int_0^{\beta_{p_i}} {B_i}^{a_i - 1}(1 - B_i)^{b_i - 1}\, dB_i, \quad 0 \leq \beta_{p_i} \leq 1;\ a_i, b_i > 0.$$

The algorithm is a modified version of the Newton–Raphson method, following closely that of Cran *et al.* (1977).

An initial approximation, β_{i0}, to β_{p_i} is found (see Cran *et al.* (1977)), and the Newton–Raphson iteration

$$\beta_k = \beta_{k-1} - \frac{f_i(\beta_{k-1})}{{f_i}'(\beta_{k-1})},$$

where $f_i(\beta_k) = P\left(B_i \leq \beta_k : a_i, b_i\right) - p_i$ is used, with modifications to ensure that β_k remains in the range $(0, 1)$.

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

Cran G W, Martin K J and Thomas G E (1977) Algorithm AS 109. Inverse of the incomplete beta function ratio *Appl. Statist.* **26** 111–114

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 which tail the supplied probabilities represent. For $j = (i-1) \bmod \mathbf{ltail}$, for $i = 1, 2, \ldots, \max(\mathbf{ltail}, \mathbf{lp}, \mathbf{la}, \mathbf{lb})$:

tail$[j]$ = Nag_LowerTail
The lower tail probability, i.e., $p_i = P\left(B_i \leq \beta_{p_i} : a_i, b_i\right)$.

tail$[j]$ = Nag_UpperTail
The upper tail probability, i.e., $p_i = P\left(B_i \geq \beta_{p_i} : a_i, b_i\right)$.

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

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 probability of the required beta distribution as defined by **tail** with $p_i =$ **p**$[j]$, $j = (i-1) \bmod$ **lp**.

Constraint: $0.0 \leq$ **p**$[j-1] \leq 1.0$, for $j = 1, 2, \ldots,$ **lp**.

5: **la** – Integer *Input*

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

Constraint: **la** > 0.

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

On entry: a_i, the first parameter of the required beta distribution with $a_i =$ **a**$[j]$, $j = (i-1) \bmod$ **la**.

Constraint: $0.0 <$ **a**$[j-1] \leq 10^6$, 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: b_i, the second parameter of the required beta distribution with $b_i =$ **b**$[j]$, $j = (i-1) \bmod$ **lb**.

Constraint: $0.0 <$ **b**$[j-1] \leq 10^6$, for $j = 1, 2, \ldots,$ **lb**.

9: **tol** – double *Input*

On entry: the relative accuracy required by you in the results. If nag_deviates_beta_vector (g01tec) is entered with **tol** greater than or equal to 1.0 or less than $10 \times$ ***machine precision*** (see nag_machine_precision (X02AJC)), then the value of $10 \times$ ***machine precision*** is used instead.

10: **beta**$[dim]$ – double *Output*

Note: the dimension, *dim*, of the array **beta** must be at least $\max($**ltail**, **lp**, **la**, **lb**$)$.

On exit: β_{p_i}, the deviates for the beta distribution.

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

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

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 β_{p_i}.

ivalid$[i-1] = 2$

On entry, $p_i < 0.0$,
or $p_i > 1.0$.

ivalid$[i-1] = 3$

On entry, $a_i \leq 0.0$,
or $a_i > 10^6$,
or $b_i \leq 0.0$,
or $b_i > 10^6$.

ivalid$[i-1] = 4$
The solution has not converged but the result should be a reasonable approximation to the solution.

ivalid$[i-1] = 5$
Requested accuracy not achieved when calculating the beta probability. The result should be a reasonable approximation to the correct solution.

12: **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 = ⟨*value*⟩.
Constraint: **la** > 0.

On entry, array size = ⟨*value*⟩.
Constraint: **lb** > 0.

On entry, array size = ⟨*value*⟩.
Constraint: **lp** > 0.

On entry, array size = ⟨*value*⟩.
Constraint: **ltail** > 0.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ 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 **tail**, **p**, **a**, or **b** was invalid, or the solution failed to converge. Check **ivalid** for more information.

7 Accuracy

The required precision, given by **tol**, should be achieved in most circumstances.

8 Parallelism and Performance

nag_deviates_beta_vector (g01tec) is not threaded in any implementation.

9 Further Comments

The typical timing will be several times that of nag_prob_beta_vector (g01sec) and will be very dependent on the input argument values. See nag_prob_beta_vector (g01sec) for further comments on timings.

10 Example

This example reads lower tail probabilities for several beta distributions and calculates and prints the corresponding deviates.

10.1 Program Text

```
/* nag_deviates_beta_vector (g01tec) 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 ltail, lp, 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 tol;
  double *p = 0, *a = 0, *b = 0, *beta = 0;

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

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

```
  printf("nag_deviates_beta_vector (g01tec) Example Program Results\n\n");

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

  /* Read in the tolerance */
#ifdef _WIN32
  scanf_s("%lf%*[^\n] ", &tol);
#else
  scanf("%lf%*[^\n] ", &tol);
#endif

  /* Read in the input vectors */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n] ", &ltail);
#else
  scanf("%" NAG_IFMT "%*[^\n] ", &ltail);
#endif
  if (!(tail = NAG_ALLOC(ltail, Nag_TailProbability))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  for (i = 0; i < ltail; i++) {
#ifdef _WIN32
    scanf_s("%39s", ctail, (unsigned)_countof(ctail));
#else
    scanf("%39s", ctail);
#endif
    tail[i] = (Nag_TailProbability) nag_enum_name_to_value(ctail);
  }
#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] ", &la);
#else
  scanf("%" NAG_IFMT "%*[^\n] ", &la);
#endif
  if (!(a = NAG_ALLOC(la, double)))
```

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

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

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

  /* Calculate probability */
  nag_deviates_beta_vector(ltail, tail, lp, p, la, a, lb, b, tol, beta,
                           ivalid, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_deviates_beta_vector (g01tec).\n%s\n",
            fail.message);
    exit_status = 1;
    if (fail.code != NW_IVALID)
      goto END;
  }

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

  /* Display results */
  for (i = 0; i < lout; i++)
    printf(" %15s  %6.3f    %6.2f    %6.2f   %7.3f    %3" NAG_IFMT "\n",
            nag_enum_value_to_name(tail[i % ltail]), p[i % lp], a[i % la],
            b[i % lb], beta[i], ivalid[i]);
```

```
END:
  NAG_FREE(tail);
  NAG_FREE(p);
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(beta);
  NAG_FREE(ivalid);

  return (exit_status);
}
```

10.2 Program Data

```
nag_deviates_beta_vector (g01tec) Example Program Data
0.0                                                                    :: tol
1                                                                      :: ltail
Nag_LowerTail                                                          :: tail
3                                                                      :: lp
0.50 0.99 0.25                                                         :: p
3                                                                      :: la
1.0 1.5 20.0                                                           :: a
3                                                                      :: lb
2.0 1.5 10.0                                                           :: b
```

10.3 Program Results

```
nag_deviates_beta_vector (g01tec) Example Program Results

        tail         p          a         b       beta   ivalid
 ------------------------------------------------------------------
   Nag_LowerTail   0.500       1.00      2.00     0.293       0
   Nag_LowerTail   0.990       1.50      1.50     0.967       0
   Nag_LowerTail   0.250      20.00     10.00     0.611       0
```