nag_prob_beta_vector (g01sec)

1 Purpose

nag_prob_beta_vector (g01sec) computes a number of lower or upper tail probabilities for the beta distribution.

2 Specification

```c
#include <nag.h>
#include <nag01.h>

void nag_prob_beta_vector (Integer ltail, const Nag_TailProbability tail[],
                        Integer lbeta, const double beta[], Integer la, const double a[],
                        Integer lb, const double b[], double p[], Integer ivalid[],
                        NagError *fail)
```

3 Description

The lower tail probability, \( P(B_i \leq \beta_i : a_i, b_i) \) is defined by

\[
P(B_i \leq \beta_i : a_i, b_i) = \frac{I(a_i + b_i)}{I(a_i, b_i)} \int_0^{\beta_i} B_i^{a_i-1}(1-B_i)^{b_i-1} dB_i = I_{\beta_i}(a_i, b_i), \quad 0 \leq \beta_i \leq 1; \quad a_i, b_i > 0.
\]

The function \( I_{\beta_i}(a_i, b_i) \), also known as the incomplete beta function is calculated using nag_incomplete_beta (s14ccc).

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


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 probabilities are required. For \( j = (i - 1) \mod ltail \), for \( i = 1, 2, \ldots, \max(ltail, lbeta, la, lb) \):
   
   `tail[j] = Nag_LowerTail`
   
   The lower tail probability is returned, i.e., \( p_i = P(B_i \leq \beta_i : a_i, b_i) \).
The upper tail probability is returned, i.e., \( p_i = P_B (B_i \geq \beta_i : a_i, b_i) \).

**Constraint:** \( \text{tail}[j] = \text{Nag\_LowerTail} \) or \( \text{Nag\_UpperTail} \), for \( j = 1, 2, \ldots, \text{ltail} \).

3: \text{ibeta} – Integer  
**Input**  
*On entry:* the length of the array \text{beta}.  
**Constraint:** \( \text{ibeta} > 0 \).

4: \( \text{beta}[\text{ibeta}] \) – const double  
**Input**  
*On entry:* \( \beta_i \), the value of the beta variate with \( \beta_i = \text{beta}[j] \), \( j = (i - 1) \mod \text{ibeta} \).  
**Constraint:** \( 0.0 \leq \text{beta}[j - 1] \leq 1.0 \), for \( j = 1, 2, \ldots, \text{ibeta} \).

5: \text{la} – Integer  
**Input**  
*On entry:* the length of the array \text{a}.  
**Constraint:** \( \text{la} > 0 \).

6: \( \text{a}[\text{la}] \) – const double  
**Input**  
*On entry:* \( a_i \), the first parameter of the required beta distribution with \( a_i = \text{a}[j] \), \( j = (i - 1) \mod \text{la} \).  
**Constraint:** \( \text{a}[j - 1] > 0.0 \), for \( j = 1, 2, \ldots, \text{la} \).

7: \text{lb} – Integer  
**Input**  
*On entry:* the length of the array \text{b}.  
**Constraint:** \( \text{lb} > 0 \).

8: \( \text{b}[\text{lb}] \) – const double  
**Input**  
*On entry:* \( b_i \), the second parameter of the required beta distribution with \( b_i = \text{b}[j] \), \( j = (i - 1) \mod \text{lb} \).  
**Constraint:** \( \text{b}[j - 1] > 0.0 \), for \( j = 1, 2, \ldots, \text{lb} \).

9: \( \text{p}[\text{dim}] \) – double  
**Output**  
*Note:* the dimension, \( \text{dim} \), of the array \text{p} must be at least \( \max(\text{ltail}, \text{ibeta}, \text{la}, \text{lb}) \).  
*On exit:* \( p_i \), the probabilities for the beta distribution.

10: \( \text{invalid}[\text{dim}] \) – Integer  
**Output**  
*Note:* the dimension, \( \text{dim} \), of the array \text{invalid} must be at least \( \max(\text{ltail}, \text{ibeta}, \text{la}, \text{lb}) \).  
*On exit:* \( \text{invalid}[i - 1] \) indicates any errors with the input arguments, with  
\( \text{invalid}[i - 1] = 0 \)  
No error.  
\( \text{invalid}[i - 1] = 1 \)  
On entry, invalid value supplied in \text{tail} when calculating \( p_i \).  
\( \text{invalid}[i - 1] = 2 \)  
On entry, \( \beta_i < 0.0 \),  
or \( \beta_i > 1.0 \).
ivalid \[i - 1\] = 3

On entry, \(a_i \leq 0.0\),

or \(b_i \leq 0.0\).

On entry, array size = \(\text{value}\).

Constraint: \(la > 0\).

On entry, array size = \(\text{value}\).

Constraint: \(lb > 0\).

On entry, array size = \(\text{value}\).

Constraint: \(lbeta > 0\).

On entry, array size = \(\text{value}\).

Constraint: \(ltail > 0\).

On entry, at least one value of beta, a, b or tail was invalid.

Check invalid for more information.

6 Error Indicators and Warnings

NE_ALLOC_FAIL
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_ARRAY_SIZE
On entry, array size = \(\text{value}\).

Constraint: \(la > 0\).

On entry, array size = \(\text{value}\).

Constraint: \(lb > 0\).

On entry, array size = \(\text{value}\).

Constraint: \(lbeta > 0\).

On entry, array size = \(\text{value}\).

Constraint: \(ltail > 0\).

NE_BAD_PARAM
On entry, argument \(\text{value}\) 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 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE
Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

NW_INVALID
On entry, at least one value of beta, a, b or tail was invalid.

Check invalid for more information.

7 Accuracy

The accuracy is limited by the error in the incomplete beta function. See Section 7 in nag_incomplete_beta (s14ccc) for further details.

8 Parallelism and Performance

Not applicable.
9 Further Comments

None.

10 Example

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

10.1 Program Text

/* nag_prob_beta_vector (g01sec) Example Program. */
* Copyright 2014 Numerical Algorithms Group.
* Mark 23, 2011. */
#include <stdio.h>
#include <nag.h>
#include <naq_stdlib.h>
#include <naqg01.h>

int main(void)
{
  /* Integer scalar and array declarations */
  Integer ltaill, lbeta, 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 *beta = 0, *a = 0, *b = 0, *p = 0;
  /* Character scalar and array declarations */
  char ctaill[40];
  /* Initialise the error structure to print out any error messages */
  INIT_FAIL(fail);
  printf("nag_prob_beta_vector (g01sec) Example Program Results\n\n");
  /* Skip heading in data file*/
  #ifdef _WIN32
    scanf_s("%*[\n] ");
  #else
    scanf("%*[\n] ");
  #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(ltaill, Nag_TailProbability))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  for (i = 0; i < ltaill; i++) {
    #ifdef _WIN32
      scanf_s("%39s", ctaill, _countof(ctaill));
    #else
      scanf("%39s", ctaill);
    #endif
    /* Example: */
    *p = nag_prob_beta_vector(ltaill, beta, a, b, &p, fail, tail, ivalid);
    printf("%f\n", *p);
  }
  NAG_FREE(tail);
  NAG_FREE(ivalid);
  return exit_status;
}

```c
    tail[i] = (Nag_TailProbability) nag_enum_name_to_value(ctail);
}
#endif
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

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

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

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

```c
scanf("%*[\n\"]");
#endif
/* Allocate memory for output */
lout = MAX(ltail,MAX(lbeta,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_beta_vector(ltail, tail, lbeta, beta, la, a, lb, b,
    p, ivalid, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_prob_beta_vector (g01sec).\n%s\n", fail.message);
    exit_status = 1;
    if (fail.code != NW_IVALID) goto END;
}
/* Display title */
printf(" tail beta a b p ivalid\n");
printf(" ----------------------------------------------------\n");
/* Display results */
for (i = 0; i < lout; i++)
    printf(" %15s %6.2f %6.2f %6.2f %6.3f %3"NAG_IFMT"
        , nag_enum_value_to_name(tail[i%ltail]), beta[i%lbeta], a[i%la],
        b[i%lb], p[i], ivalid[i]);
END:
NAG_FREE(tail);
NAG_FREE(beta);
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(p);
NAG_FREE(ivalid);
return(exit_status);
}

10.2 Program Data

nag_prob_beta_vector (g01sec) Example Program Data
1 :: ltail
Nag_LowerTail :: tail
3 :: lbeta
0.26 0.75 0.5 :: beta
3 :: la
1.0 1.5 2.0 :: a
3 :: lb
2.0 1.5 1.0 :: b

10.3 Program Results

nag_prob_beta_vector (g01sec) Example Program Results

<table>
<thead>
<tr>
<th>tail</th>
<th>beta</th>
<th>a</th>
<th>b</th>
<th>p</th>
<th>ivalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nag_LowerTail</td>
<td>0.26</td>
<td>1.00</td>
<td>2.00</td>
<td>0.452</td>
<td>0</td>
</tr>
<tr>
<td>Nag_LowerTail</td>
<td>0.75</td>
<td>1.50</td>
<td>1.50</td>
<td>0.804</td>
<td>0</td>
</tr>
<tr>
<td>Nag_LowerTail</td>
<td>0.50</td>
<td>2.00</td>
<td>1.00</td>
<td>0.250</td>
<td>0</td>
</tr>
</tbody>
</table>
```