NAG Library Function Document

nag_prob_f_vector (g01sdc)

1 Purpose

nag_prob_f_vector (g01sdc) returns a number of lower or upper tail probabilities for the $F$ or variance-ratio distribution with real degrees of freedom.

2 Specification

```c
#include <nag.h>
#include <nagg01.h>

void nag_prob_f_vector (Integer ltail, const Nag_TailProbability tail[],
  Integer lf, const double f[], Integer ldf1, const double df1[],
  Integer ldf2, const double df2[], double p[], Integer ivalid[],
  NagError *fail)
```

3 Description

The lower tail probability for the $F$, or variance-ratio, distribution with $u_i$ and $v_i$ degrees of freedom, $P(F_i \leq f_i : u_i, v_i)$, is defined by:

$$P(F_i \leq f_i : u_i, v_i) = \frac{\Gamma\left(\frac{u_i + v_i}{2}\right)}{\Gamma\left(\frac{u_i}{2}\right)\Gamma\left(\frac{v_i}{2}\right)} \int_0^{f_i} F_i^{(u_i-2)/2}(u_iF_i + v_i)^{-\left(u_i+v_i\right)/2} dF_i,$$

for $u_i, v_i > 0$, $f_i \geq 0$.

The probability is computed by means of a transformation to a beta distribution, $P_{\beta_i}(B_i \leq \beta_i : a_i, b_i)$:

$$P(F_i \leq f_i : u_i, v_i) = P_{\beta_i}\left(\frac{f_i}{u_iF_i + v_i}, \frac{u_i}{2}, \frac{v_i}{2}\right)$$

and using a call to nag_prob_beta_dist (g01eec).

For very large values of both $u_i$ and $v_i$, greater than $10^5$, a normal approximation is used. If only one of $u_i$ or $v_i$ is greater than $10^5$ then a $\chi^2$ approximation is used, see Abramowitz and Stegun (1972).

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 $> 0$. 

Mark 25
2: tail[ltail] – const Nag_TailProbability

Input

On entry: indicates whether the lower or upper tail probabilities are required. For
\[ j = (i - 1) \mod \text{ltail}, \text{for } i = 1, 2, \ldots, \max(\text{ltail, lf, ldf1, ldf2}) \):

\[
\text{tail}[j] = \text{Nag}\_\text{LowerTail}
\]

The lower tail probability is returned, i.e., \( p_i = P(F_i \leq f_i : u_i, v_i) \).

\[
\text{tail}[j] = \text{Nag}\_\text{UpperTail}
\]

The upper tail probability is returned, i.e., \( p_i = P(F_i \geq f_i : u_i, v_i) \).

Constraint: \( \text{tail}[j - 1] = \text{Nag}\_\text{LowerTail} \) or \( \text{Nag}\_\text{UpperTail} \), for \( j = 1, 2, \ldots, \text{ltail} \).

3: If – Integer

Input

On entry: the length of the array \( f \).

Constraint: \( \text{If} > 0 \).

4: f[lf] – const double

Input

On entry: \( f_j \), the value of the \( F \) variate with \( f_i = f[j], j = (i - 1) \mod \text{lf} \).

Constraint: \( f[j - 1] \geq 0.0 \), for \( j = 1, 2, \ldots, \text{lf} \).

5: ldf1 – Integer

Input

On entry: the length of the array \( \text{df1} \).

Constraint: \( \text{ldf1} > 0 \).

6: df1[ldf1] – const double

Input

On entry: \( u_j \), the degrees of freedom of the numerator variance with \( u_i = \text{df1}[j], j = (i - 1) \mod \text{ldf1} \).

Constraint: \( \text{df1}[j - 1] > 0.0 \), for \( j = 1, 2, \ldots, \text{ldf1} \).

7: ldf2 – Integer

Input

On entry: the length of the array \( \text{df2} \).

Constraint: \( \text{ldf2} > 0 \).

8: df2[ldf2] – const double

Input

On entry: \( v_j \), the degrees of freedom of the denominator variance with \( v_i = \text{df2}[j], j = (i - 1) \mod \text{ldf2} \).

Constraint: \( \text{df2}[j - 1] > 0.0 \), for \( j = 1, 2, \ldots, \text{ldf2} \).

9: p[dim] – double

Output

Note: the dimension, \( \text{dim} \), of the array \( p \) must be at least \( \max(\text{ltail, lf, ldf1, ldf2}) \).

On exit: \( p_i \), the probabilities for the \( F \)-distribution.

10: invalid[dim] – Integer

Output

Note: the dimension, \( \text{dim} \), of the array \( \text{invalid} \) must be at least \( \max(\text{ltail, lf, ldf1, ldf2}) \).

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 \( p_i \).
ivalid[i − 1] = 2
On entry, fi < 0.0.
ivalid[i − 1] = 3
On entry, ui ≤ 0.0,
or vi ≤ 0.0.
ivalid[i − 1] = 4
The solution has failed to converge. The result returned should represent an 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.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_ARRAY_SIZE
On entry, array size = (value).
Constraint: ldf1 > 0.
On entry, array size = (value).
Constraint: ldf2 > 0.
On entry, array size = (value).
Constraint: if > 0.
On entry, array size = (value).
Constraint: itail > 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 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 f, df1, df2 or tail was invalid, or the solution failed to converge.
Check ivalid for more information.

7 Accuracy
The result should be accurate to five significant digits.
8 Parallelism and Performance
Not applicable.

9 Further Comments
For higher accuracy nag_prob_beta_vector (g01sec) can be used along with the transformations given in Section 3.

10 Example
This example reads values from, and degrees of freedom for, a number of F-distributions and computes the associated lower tail probabilities.

10.1 Program Text
/* nag_prob_f_vector (g01sdc) Example Program.
* * Copyright 2014 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, lf, ldf1, ldf2, i, lout;
    Integer *ivalid = 0;
    Integer exit_status = 0;
    /* NAG structures */
    NagError fail;
    Nag_TailProbability *tail = 0;
    /* Double scalar and array declarations */
    double *f = 0, *df1 = 0, *df2 = 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_f_vector (g01sdc) 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(ltail, Nag_TailProbability))) {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}
for (i = 0; i < ltail; i++) {
    #ifdef _WIN32
        scanf_s("%39s", ctail, _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 ] ", &lf);
    #else
        scanf("%"NAG_IFMT"%*[\n ] ", &lf);
    #endif
    if (!(f = NAG_ALLOC(lf, double))) {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    for (i = 0; i < lf; i++)
        #ifdef _WIN32
            scanf_s("%lf", &f[i]);
        #else
            scanf("%lf", &f[i]);
        #endif
        #ifdef _WIN32
            scanf_s("%*[\n ] ");
        #else
            scanf("%*[\n ] ");
        #endif
        #ifdef _WIN32
            scanf_s("%"NAG_IFMT"%*[\n ] ", &ldf1);
        #else
            scanf("%"NAG_IFMT"%*[\n ] ", &ldf1);
        #endif
        if (!(df1 = NAG_ALLOC(ldf1, double))) {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
        for (i = 0; i < ldf1; i++)
            #ifdef _WIN32
                scanf_s("%lf", &df1[i]);
            #else
                scanf("%lf", &df1[i]);
            #endif
            #ifdef _WIN32
                scanf_s("%*[\n ] ");
            #else
                scanf("%*[\n ] ");
            #endif
            #ifdef _WIN32
                scanf_s("%"NAG_IFMT"%*[\n ] ", &ldf2);
            #else
                scanf("%"NAG_IFMT"%*[\n ] ", &ldf2);
            #endif
            if (!(df2 = NAG_ALLOC(ldf2, double))) {
                printf("Allocation failure\n");
                exit_status = -1;
                goto END;
            }
            for (i = 0; i < ldf2; i++)
                #ifdef _WIN32
                    scanf_s("%lf", &df2[i]);
                #else
                    scanf("%lf", &df2[i]);
                #endif

Mark 25
```c
#if defined _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

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

/* Calculate probability */
nag_prob_f_vector(ltail, tail, lf, ldf1, df1, ldf2, df2,
    p, ivalid, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_prob_f_vector (g01sdc).\n", fail.message);
    exit_status = 1;
    if (fail.code != NW_IVALID) goto END;
}

/* Display title */
printf(" tail f df1 df2 ");
printf("p ivalid\n");
printf("----------------------------------");

/* 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]), f[i%lf], df1[i%ldf1],
        df2[i%ldf2], p[i], ivalid[i]);

END:
NAG_FREE(tail);
NAG_FREE(f);
NAG_FREE(df1);
NAG_FREE(df2);
NAG_FREE(p);
NAG_FREE(ivalid);
return(exit_status);
}
```

### 10.2 Program Data

nag_prob_f_vector (g01sdc) Example Program Data

```plaintext
1
Nag_LowerTail
3
5.5 39.9 2.5
3
1.5 1.0 20.25
3
25.5 1.0 1.0
```
### 10.3 Program Results

#### nag_prob_f_vector (g01sdc) Example Program Results

<table>
<thead>
<tr>
<th>tail</th>
<th>f</th>
<th>df1</th>
<th>df2</th>
<th>p</th>
<th>ivalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nag_LowerTail</td>
<td>5.5</td>
<td>1.5</td>
<td>25.5</td>
<td>0.984</td>
<td>0</td>
</tr>
<tr>
<td>Nag_LowerTail</td>
<td>39.9</td>
<td>1.0</td>
<td>1.0</td>
<td>0.900</td>
<td>0</td>
</tr>
<tr>
<td>Nag_LowerTail</td>
<td>2.5</td>
<td>20.25</td>
<td>1.0</td>
<td>0.534</td>
<td>0</td>
</tr>
</tbody>
</table>