NAG Library Function Document

nag_deviates_f_vector (g01tdc)

1 Purpose

nag_deviates_f_vector (g01tdc) returns a number of deviates associated with given probabilities of the F or variance-ratio distribution with real degrees of freedom.

2 Specification

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

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

3 Description

The deviate, \( f_{pi} \), associated with the lower tail probability, \( pi \), of the F-distribution with degrees of freedom \( ui \) and \( vi \) is defined as the solution to

\[
P(F_i \leq f_{pi} : u_i, v_i) = p_i = \frac{\Gamma(\frac{u_i + v_i}{2})}{\Gamma(\frac{u_i}{2})\Gamma(\frac{v_i}{2})} \int_{0}^{f_{pi}} F_i^{\frac{1}{2}(u_i-2)}(v_i + u_i F_i)^{-\frac{1}{2}(u_i+v_i)} dF_i,
\]

where \( u_i, v_i > 0; 0 \leq f_{pi} < \infty \).

The value of \( f_{pi} \) is computed by means of a transformation to a beta distribution, \( P(\beta_i : a_i, b_i) \):

\[
P(F_i \leq f_{pi} : u_i, v_i) = P(\beta_i : a_i, b_i) \leq \frac{u_i f_{pi}}{u_i f_{pi} + v_i ; u_i/2, v_i/2}
\]

and using a call to nag_deviates_beta_vector (g01tec).

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. **Itail** – Integer  
   
   *Input*
   
   *On entry*: the length of the array *tail*.
   
   *Constraint*: *Itail* > 0.
2: tail[ltail] – const Nag_TailProbability
   
   On entry: indicates which tail the supplied probabilities represent. For \( j = (i - 1) \mod ltail \), for \( i = 1, 2, \ldots, \max(\text{ltail}, \text{lp}, \text{ldf1}, \text{ldf2}) \):
   
   \( \text{tail}[j] = \text{Nag_LowerTail} \)
   
   The lower tail probability, i.e., \( p_i = P(F_i \leq f_{p_i} : u_i, v_i) \).
   
   \( \text{tail}[j] = \text{Nag_UpperTail} \)
   
   The upper tail probability, i.e., \( p_i = P(F_i \geq f_{p_i} : u_i, v_i) \).
   
   Constraint: \( \text{tail}[j - 1] = \text{Nag_LowerTail} \) or \( \text{Nag_UpperTail} \), for \( j = 1, 2, \ldots, \text{ltail} \).

3: lp – Integer
   
   On entry: the length of the array \( p \).
   
   Constraint: \( \text{lp} > 0 \).

4: p[lp] – const double
   
   On entry: \( p_i \), the probability of the required \( F \)-distribution as defined by \( \text{tail} \) with \( p_i = \text{p}[j] \), \( j = (i - 1) \mod \text{lp} \).
   
   Constraints:
   
   if \( \text{tail}[k] = \text{Nag_LowerTail} \), \( 0.0 \leq \text{p}[j] < 1.0 \);
   
   otherwise \( 0.0 < \text{p}[j] \leq 1.0 \).
   
   Where \( k = (i - 1) \mod \text{ltail} \) and \( j = (i - 1) \mod \text{lp} \).

5: ldf1 – Integer
   
   On entry: the length of the array \( df1 \).
   
   Constraint: \( \text{ldf1} > 0 \).

6: df1[ldf1] – const double
   
   On entry: \( u_i \), 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
   
   On entry: the length of the array \( df2 \).
   
   Constraint: \( \text{ldf2} > 0 \).

8: df2[ldf2] – const double
   
   On entry: \( v_i \), 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: f[dim] – double
   
   Note: the dimension, \( \text{dim} \), of the array \( f \) must be at least \( \max(\text{ltail}, \text{lp}, \text{ldf1}, \text{ldf2}) \).
   
   On exit: \( f_{p_i} \), the deviates for the \( F \)-distribution.

10: invalid[dim] – Integer
    
    Note: the dimension, \( \text{dim} \), of the array \( \text{invalid} \) must be at least \( \max(\text{ltail}, \text{lp}, \text{ldf1}, \text{ldf2}) \).
On exit: \( \text{ivalid}[i - 1] \) indicates any errors with the input arguments, with
\[
\text{ivalid}[i - 1] = 0
\]
No error.
\[
\text{ivalid}[i - 1] = 1
\]
On entry, invalid value supplied in \text{tail} when calculating \( f_{pi} \).
\[
\text{ivalid}[i - 1] = 2
\]
On entry, invalid value for \( p_i \).
\[
\text{ivalid}[i - 1] = 3
\]
On entry, \( u_i \leq 0.0 \),
or\[ v_i \leq 0.0. \]
\[
\text{ivalid}[i - 1] = 4
\]
The solution has not converged. The result should still be a reasonable approximation to the solution.
\[
\text{ivalid}[i - 1] = 5
\]
The value of \( p_i \) is too close to 0.0 or 1.0 for the result to be computed. This will only occur when the large sample approximations are used.

11: fail – NagError *

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 = \( \langle \text{value} \rangle \).
Constraint: \( \text{ldf1} > 0. \)
On entry, array size = \( \langle \text{value} \rangle \).
Constraint: \( \text{ldf2} > 0. \)
On entry, array size = \( \langle \text{value} \rangle \).
Constraint: \( \text{lp} > 0. \)
On entry, array size = \( \langle \text{value} \rangle \).
Constraint: \( \text{ltail} > 0. \)

**NE_BAD_PARAM**
On entry, argument \( \langle \text{value} \rangle \) 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.
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.
On entry, at least one value of tail, p, df1, df2 was invalid, or the solution failed to converge. Check invalid 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_deviates_beta_vector (g01tec) can be used along with the transformations given in Section 3.

10 Example
This example reads the lower tail probabilities for several F-distributions, and calculates and prints the corresponding deviates.

10.1 Program Text
/* nag_deviates_f_vector (g01tdc) 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, lp, 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 *p = 0, *df1 = 0, *df2 = 0, *f = 0;
    /* Character scalar and array declarations */
    char ctail[40];
    /* Initialise the error structure to print out any error messages */
    INIT_FAIL(fail);
    printf("nag_deviates_f_vector (g01tdc) Example Program Results\n\n");
    /* Skip heading in data file*/
    #ifdef _WIN32
        scanf_s("%*[\n] ");
    #else
        scanf("%*[\n] ");
    #endif

    /* Read the data file */
    /* Process the data */
    /* Calculate the deviates */
    /* Print the results */
    return 0;
}

/* 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] ", &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] ", &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
#include

```c
    scanf("%"NAG_IFMT"%[\n] ", &ldf2);
#endif
    if (df2 = NAG_ALLOC(ldf2, double)) {
        exit_status = -1;
        goto END;
    }
    for (i = 0; i < ldf2; i++)
        #ifdef _WIN32
            scanf_s("%lf", &df2[i]);
        #else
            scanf("%lf", &df2[i]);
        #endif
        #ifdef _WIN32
            scanf_s("%*[\n] ");
        #else
            scanf("%*[\n] ");
        #endif
/* Allocate memory for output */
    lout = MAX(ltail, MAX(lp, MAX(ldf1, ldf2)));
    if (!(f = NAG_ALLOC(lout, double) ||
          !(ivalid = NAG_ALLOC(lout, Integer)))) {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
/* Calculate probability */
    nag_deviates_f_vector(ltail, tail, lp, ldf1, df1, ldf2, df2, f,
        ivalid, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_deviates_f_vector (g01tdc)\n%s\n",
            fail.message);
        exit_status = 1;
        if (fail.code != NW_IVALID) goto END;
    }
/* Display title */
    printf(" tail p df1 df2 f 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"
",
            nag_enum_value_to_name(tail[i%ltail]),p[i%lp], df1[i%ldf1],
            df2[i%ldf2], f[i], ivalid[i]);
END:
    NAG_FREE(tail);
    NAG_FREE(p);
    NAG_FREE(df1);
    NAG_FREE(df2);
    NAG_FREE(f);
    NAG_FREE(ivalid);
    return(exit_status);
}
```
10.2 Program Data

```
nag_deviates_f_vector (g01tdc) Example Program Data
1 :: ltail
Nag_LowerTail :: tail
3 :: lp
0.984 0.9 0.534 :: p
3 :: ldf1
10.0 1.0 20.25 :: df1
3 :: ldf2
25.5 1.0 1.0 :: df2
```

10.3 Program Results

```
nag_deviates_f_vector (g01tdc) Example Program Results

<table>
<thead>
<tr>
<th>tail</th>
<th>p</th>
<th>df1</th>
<th>df2</th>
<th>f</th>
<th>invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nag_LowerTail</td>
<td>0.984</td>
<td>10.00</td>
<td>25.50</td>
<td>2.847</td>
<td>0</td>
</tr>
<tr>
<td>Nag_LowerTail</td>
<td>0.900</td>
<td>1.00</td>
<td>1.00</td>
<td>39.863</td>
<td>0</td>
</tr>
<tr>
<td>Nag_LowerTail</td>
<td>0.534</td>
<td>20.25</td>
<td>1.00</td>
<td>2.498</td>
<td>0</td>
</tr>
</tbody>
</table>
```