

## 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

```
#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:

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

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

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

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

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

Abramowitz M and Stegun I A (1972) *Handbook of Mathematical Functions* (3rd Edition) Dover Publications

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 whether the lower or upper tail probabilities are required. For  $j = (i - 1) \bmod \mathbf{ltail}$ , for  $i = 1, 2, \dots, \max(\mathbf{ltail}, \mathbf{lf}, \mathbf{ldf1}, \mathbf{ldf2})$ :  
**tail**[ $j$ ] = Nag\_LowerTail  
The lower tail probability is returned, i.e.,  $p_i = P(F_i \leq f_i : u_i, v_i)$ .  
**tail**[ $j$ ] = Nag\_UpperTail  
The upper tail probability is returned, i.e.,  $p_i = P(F_i \geq f_i : u_i, v_i)$ .  
*Constraint:* **tail**[ $j - 1$ ] = Nag\_LowerTail or Nag\_UpperTail, for  $j = 1, 2, \dots, \mathbf{ltail}$ .
- 3: **lf** – Integer *Input*  
*On entry:* the length of the array **f**.  
*Constraint:* **lf** > 0.
- 4: **f**[**lf**] – const double *Input*  
*On entry:*  $f_i$ , the value of the  $F$  variate with  $f_i = \mathbf{f}[j]$ ,  $j = (i - 1) \bmod \mathbf{lf}$ .  
*Constraint:* **f**[ $j - 1$ ]  $\geq 0.0$ , for  $j = 1, 2, \dots, \mathbf{lf}$ .
- 5: **ldf1** – Integer *Input*  
*On entry:* the length of the array **df1**.  
*Constraint:* **ldf1** > 0.
- 6: **df1**[**ldf1**] – const double *Input*  
*On entry:*  $u_i$ , the degrees of freedom of the numerator variance with  $u_i = \mathbf{df1}[j]$ ,  $j = (i - 1) \bmod \mathbf{ldf1}$ .  
*Constraint:* **df1**[ $j - 1$ ] > 0.0, for  $j = 1, 2, \dots, \mathbf{ldf1}$ .
- 7: **ldf2** – Integer *Input*  
*On entry:* the length of the array **df2**.  
*Constraint:* **ldf2** > 0.
- 8: **df2**[**ldf2**] – const double *Input*  
*On entry:*  $v_i$ , the degrees of freedom of the denominator variance with  $v_i = \mathbf{df2}[j]$ ,  $j = (i - 1) \bmod \mathbf{ldf2}$ .  
*Constraint:* **df2**[ $j - 1$ ] > 0.0, for  $j = 1, 2, \dots, \mathbf{ldf2}$ .
- 9: **p**[*dim*] – double *Output*  
**Note:** the dimension, *dim*, of the array **p** must be at least  $\max(\mathbf{ltail}, \mathbf{lf}, \mathbf{ldf1}, \mathbf{ldf2})$ .  
*On exit:*  $p_i$ , the probabilities for the  $F$ -distribution.
- 10: **ivalid**[*dim*] – Integer *Output*  
**Note:** the dimension, *dim*, of the array **ivalid** must be at least  $\max(\mathbf{ltail}, \mathbf{lf}, \mathbf{ldf1}, \mathbf{ldf2})$ .  
*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  $p_i$ .

**ivalid**[ $i - 1$ ] = 2

On entry,  $f_i < 0.0$ .

**ivalid**[ $i - 1$ ] = 3

On entry,  $u_i \leq 0.0$ ,  
or  $v_i \leq 0.0$ .

**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 =  $\langle value \rangle$ .

Constraint: **ldf1** > 0.

On entry, array size =  $\langle value \rangle$ .

Constraint: **ldf2** > 0.

On entry, array size =  $\langle value \rangle$ .

Constraint: **If** > 0.

On entry, array size =  $\langle value \rangle$ .

Constraint: **Itail** > 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.

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
#ifdef _WIN32
    scanf_s("%*[^\\n] ");
#else
    scanf("%*[^\\n] ");
#endif

/* Allocate memory for output */
lout = MAX(ltail,MAX(lf,MAX(ldf1,ldf2)));
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, f, ldf1, df1, ldf2, df2,
    p, ivalid, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_prob_f_vector (g01sdc).\\n%s\\n",
        fail.message);
    exit_status = 1;
    if (fail.code != NW_INVALID) goto END;
}

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

/* Display results */
for (i = 0; i < lout; i++)
    printf(" %15s %6.2f %6.2f %6.2f %6.3f %3\"NAG_IFMT\"\\n",
        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

1		:: ltail
Nag_LowerTail		:: tail
3		:: lf1
5.5 39.9 2.5		:: f
3		:: ldf1
1.5 1.0 20.25		:: df1
3		:: ldf2
25.5 1.0 1.0		:: df2

### 10.3 Program Results

nag\_prob\_f\_vector (g01sdc) Example Program Results

tail	f	df1	df2	p	ivalid
Nag_LowerTail	5.50	1.50	25.50	0.984	0
Nag_LowerTail	39.90	1.00	1.00	0.900	0
Nag_LowerTail	2.50	20.25	1.00	0.534	0

---