NAG Library Function Document

nag_deviates_normal_vector (g01tac)

1 Purpose

nag_deviates_normal_vector (g01tac) returns a number of deviates associated with given probabilities of the Normal distribution.

2 Specification

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

```
void nag_deviates_normal_vector (Integer ltail,
     const Nag_TailProbability tail[], Integer lp, const double p[],
     Integer lxmu, const double xmu[], Integer lxstd, const double xstd[],
     double x[], Integer ivalid[], NagError *fail)
```

3 Description

The deviate, x_{p_i} associated with the lower tail probability, p_i, for the Normal distribution is defined as the solution to

$$P\left(X_i \leq x_{p_i}\right) = p_i = \int_{-\infty}^{x_{p_i}} Z_i(X_i)\, dX_i,$$

where

$$Z_i(X_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-(X_i-\mu_i)^2/\left(2\sigma_i^2\right)}, \; -\infty < X_i < \infty.$$

The method used is an extension of that of Wichura (1988). p_i is first replaced by $q_i = p_i - 0.5$.

(a) If $|q_i| \leq 0.3$, z_i is computed by a rational Chebyshev approximation

$$z_i = s_i \frac{A_i(s_i^2)}{B_i(s_i^2)},$$

where $s_i = \sqrt{2\pi} q_i$ and A_i, B_i are polynomials of degree 7.

(b) If $0.3 < |q_i| \leq 0.42$, z_i is computed by a rational Chebyshev approximation

$$z_i = \operatorname{sign} q_i \left(\frac{C_i(t_i)}{D_i(t_i)}\right),$$

where $t_i = |q_i| - 0.3$ and C_i, D_i are polynomials of degree 5.

(c) If $|q_i| > 0.42$, z_i is computed as

$$z_i = \operatorname{sign} q_i \left[\left(\frac{E_i(u_i)}{F_i(u_i)}\right) + u_i\right],$$

where $u_i = \sqrt{-2 \times \log\left(\min(p_i, 1 - p_i)\right)}$ and E_i, F_i are polynomials of degree 6.

x_{p_i} is then calculated from z_i, using the relationsship $z_{p_i} = \frac{x_i - \mu_i}{\sigma_i}$.

For the upper tail probability $-x_{p_i}$ is returned, while for the two tail probabilities the value $x_{ip_i^*}$ is returned, where p_i^* is the required tail probability computed from the input value of p_i.

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

Wichura (1988) Algorithm AS 241: the percentage points of the Normal distribution *Appl. Statist.* **37** 477–484

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. Letting Z denote a variate from a standard Normal distribution, and $z_i = \frac{x_{p_i} - \mu_i}{\sigma_i}$, then for $j = (i-1) \bmod$ **ltail**, for $i = 1, 2, \ldots, \max($**ltail**, **lp**, **lxmu**, **lxstd**$)$:

tail$[j]$ = Nag_LowerTail
The lower tail probability, i.e., $p_i = P(Z \leq z_i)$.

tail$[j]$ = Nag_UpperTail
The upper tail probability, i.e., $p_i = P(Z \geq z_i)$.

tail$[j]$ = Nag_TwoTailConfid
The two tail (confidence interval) probability, i.e., $p_i = P(Z \leq |z_i|) - P(Z \leq -|z_i|)$.

tail$[j]$ = Nag_TwoTailSignif
The two tail (significance level) probability, i.e., $p_i = P(Z \geq |z_i|) + P(Z \leq -|z_i|)$.

Constraint: **tail**$[j-1]$ = Nag_LowerTail, Nag_UpperTail, Nag_TwoTailConfid or Nag_TwoTailSignif, 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 probabilities for the Normal distribution as defined by **tail** with $p_i =$ **p**$[j]$, $j = (i-1) \bmod$ **lp**.

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

5: **lxmu** – Integer *Input*

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

Constraint: **lxmu** > 0.

6: **xmu**[**lxmu**] – const double *Input*

On entry: μ_i, the means with $\mu_i =$ **xmu**$[j]$, $j = (i-1) \bmod$ **lxmu**.

7: **lxstd** – Integer *Input*

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

Constraint: **lxstd** > 0.

8: **xstd**[**lxstd**] – const double *Input*

On entry: σ_i, the standard deviations with $\sigma_i =$ **xstd**$[j]$, $j = (i - 1) \bmod$ **lxstd**.

Constraint: **xstd**$[j - 1] > 0.0$, for $j = 1, 2, \dots,$ **lxstd**.

9: **x**[*dim*] – double *Output*

Note: the dimension, *dim*, of the array **x** must be at least max(**ltail**, **lxmu**, **lxstd**, **lp**).

On exit: x_{p_i}, the deviates for the Normal distribution.

10: **ivalid**[*dim*] – Integer *Output*

Note: the dimension, *dim*, of the array **ivalid** must be at least max(**ltail**, **lxmu**, **lxstd**, **lp**).

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 x_{p_i}.

ivalid$[i - 1] = 2$
On entry, $p_i \leq 0.0$,
or $p_i \geq 1.0$.

ivalid$[i - 1] = 3$
On entry, $\sigma_i \leq 0.0$.

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: **lp** > 0.

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

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

On entry, array size = ⟨*value*⟩.
Constraint: **lxstd** > 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_IVALID

On entry, at least one value of **tail**, **xstd** or **p** was invalid.
Check **ivalid** for more information.

7 Accuracy

The accuracy is mainly limited by the ***machine precision***.

8 Parallelism and Performance

Not applicable.

9 Further Comments

None.

10 Example

This example reads vectors of values for μ_i, σ_i and p_i and prints the corresponding deviates.

10.1 Program Text

```
/* nag_deviates_normal_vector (g01tac) 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, lxmu, lxstd, 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, *xmu = 0, *xstd = 0, *x = 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_normal_vector (g01tac) 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] ", &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] ", &lxmu);
#else
  scanf("%"NAG_IFMT"%*[^\n] ", &lxmu);
#endif
  if (!(xmu = NAG_ALLOC(lxmu, double))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  for (i = 0; i < lxmu; i++)
#ifdef _WIN32
    scanf_s("%lf", &xmu[i]);
```

```
#else
    scanf("%lf", &xmu[i]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

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


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

  /* Calculate probability */
  nag_deviates_normal_vector(ltail, tail, lp, p, lxmu, xmu, lxstd, xstd,
                             x, ivalid, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_deviates_normal_vector (g01tac).\n%s\n",
           fail.message);
    exit_status = 1;
    if (fail.code != NW_IVALID) goto END;
  }

  /* Display title */
  printf("           tail           p        xmu      ");
  printf("xstd       x     ivalid\n");
  printf("---------------------------------------------");
  printf("--------------------------\n");

  /* Display results */
  for (i = 0; i < lout; i++)
    printf(" %17s    %6.3f    %6.2f    %6.2f   %7.3f    %3"NAG_IFMT"\n",
           nag_enum_value_to_name(tail[i%ltail]),
           p[i%lp], xmu[i%lxmu], xstd[i%lxstd], x[i], ivalid[i]);

 END:
  NAG_FREE(tail);
  NAG_FREE(p);
  NAG_FREE(xmu);
  NAG_FREE(xstd);
```

```
  NAG_FREE(x);
  NAG_FREE(ivalid);

  return(exit_status);
}
```

10.2 Program Data

```
nag_deviates_normal_vector (g01tac) Example Program Data
4                                                                   :: ltail
Nag_LowerTail  Nag_UpperTail  Nag_TwoTailConfid  Nag_TwoTailSignif  :: tail
4                                                                   :: lp
0.975 0.025 0.95 0.05                                               :: p
1                                                                   :: lxmu
0.0                                                                 :: xmu
1                                                                   :: lxstd
1.0                                                                 :: xstd
```

10.3 Program Results

```
nag_deviates_normal_vector (g01tac) Example Program Results

           tail            p         xmu       xstd        x     ivalid
---------------------------------------------------------------------
     Nag_LowerTail     0.975       0.00       1.00      1.960       0
     Nag_UpperTail     0.025       0.00       1.00      1.960       0
 Nag_TwoTailConfid     0.950       0.00       1.00      1.960       0
 Nag_TwoTailSignif     0.050       0.00       1.00      1.960       0
```