NAG Library Function Document

nag_prob_normal_vector (g01sac)

1 Purpose

nag_prob_normal_vector (g01sac) returns a number of one or two tail probabilities for the Normal distribution.

2 Specification

```
#include <nag.h>
#include <nagg01.h>
void nag_prob_normal_vector (Integer ltail,
     const Nag_TailProbability tail[], Integer lx, const double x[],
     Integer lxmu, const double xmu[], Integer lxstd, const double xstd[],
     double p[], Integer ivalid[], NagError *fail)
```

3 Description

The lower tail probability for the Normal distribution, $P(X_i \leq x_i)$ is defined by:

$$P(X_i \leq x_i) = \int_{-\infty}^{x_i} Z_i(X_i)\, dX_i,$$

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 relationship

$$P(X_i \leq x_i) = \tfrac{1}{2}\text{erfc}\left(\frac{-(x_i-\mu_i)}{\sqrt{2}\sigma_i}\right)$$

is used, where erfc is the complementary error function, and is computed using nag_erfc (s15adc).

When the two tail confidence probability is required the relationship

$$P(X_i \leq |x_i|) - P(X_i \leq -|x_i|) = \text{erf}\left(\frac{|x_i-\mu_i|}{\sqrt{2}\sigma_i}\right),$$

is used, where erf is the error function, and is computed using nag_erf (s15aec).

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 which tail the returned probabilities should represent. Letting Z denote a variate from a standard Normal distribution, and $z_i = \frac{x_i - \mu_i}{\sigma_i}$, then for $j = (i-1) \bmod \mathbf{ltail}$, for $i = 1, 2, \ldots, \max(\mathbf{lx}, \mathbf{ltail}, \mathbf{lxmu}, \mathbf{lxstd})$:

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

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

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

tail$[j]$ = Nag_TwoTailSignif
The two tail (significance level) probability is returned, 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: **lx** – Integer *Input*

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

Constraint: **lx** > 0.

4: **x**[**lx**] – const double *Input*

On entry: x_i, the Normal variate values with $x_i =$ **x**$[j]$, $j = (i-1) \bmod$ **lx**.

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 $j = 1, 2, \ldots,$ **lxstd**.

9: **p**$[dim]$ – double *Output*

Note: the dimension, *dim*, of the array **p** must be at least $\max(\mathbf{lx}, \mathbf{ltail}, \mathbf{lxmu}, \mathbf{lxstd})$.

On exit: p_i, the probabilities for the Normal distribution.

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

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

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, $\sigma_i \leq 0.0$.

11: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_ARRAY_SIZE

On entry, **ltail** $= \langle value \rangle$.
Constraint: **ltail** > 0.

On entry, **lx** $= \langle value \rangle$.
Constraint: **lx** > 0.

On entry, **lxmu** $= \langle value \rangle$.
Constraint: **lxmu** > 0.

On entry, **lxstd** $= \langle value \rangle$.
Constraint: **lxstd** > 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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NW_IVALID

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

7 Accuracy

Accuracy is limited by ***machine precision***. For detailed error analysis see nag_erfc (s15adc) and nag_erf (s15aec).

8 Parallelism and Performance

nag_prob_normal_vector (g01sac) is not threaded in any implementation.

9 Further Comments

None.

10 Example

Four values of **tail**, **x**, **xmu** and **xstd** are input and the probabilities calculated and printed.

10.1 Program Text

```
/* nag_prob_normal_vector (g01sac) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg01.h>

int main(void)
{
  /* Integer scalar and array declarations */
  Integer ltail, lx, 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 *x = 0, *xmu = 0, *xstd = 0, *p = 0;

  /* Character scalar and array declarations */
  char ctail[40];

  /* Initialize the error structure to print out any error messages */
  INIT_FAIL(fail);

  printf("nag_prob_normal_vector (g01sac) 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, (unsigned)_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] ", &lx);
#else
  scanf("%" NAG_IFMT "%*[^\n] ", &lx);
#endif
  if (!(x = NAG_ALLOC(lx, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  for (i = 0; i < lx; i++)
#ifdef _WIN32
    scanf_s("%lf", &x[i]);
#else
    scanf("%lf", &x[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(lx, MAX(lxmu, lxstd)));
  if (!(p = NAG_ALLOC(lout, double)) || !(ivalid = NAG_ALLOC(lout, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Calculate probability */
  nag_prob_normal_vector(ltail, tail, lx, x, lxmu, xmu, lxstd, xstd,
                         p, ivalid, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_prob_normal_vector (g01sac).\n%s\n", fail.message);
    exit_status = 1;
    if (fail.code != NW_IVALID)
      goto END;
  }

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

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

END:
  NAG_FREE(tail);
  NAG_FREE(x);
  NAG_FREE(xmu);
  NAG_FREE(xstd);
  NAG_FREE(p);
  NAG_FREE(ivalid);

  return (exit_status);
}
```

10.2 Program Data

```
nag_prob_normal_vector (g01sac) Example Program Data
4                                                                       :: ltail
Nag_LowerTail  Nag_UpperTail  Nag_TwoTailConfid  Nag_TwoTailSignif     :: tail
1                                                                       :: lx
1.96                                                                    :: x
1                                                                       :: lxmu
0.0                                                                     :: xmu
1                                                                       :: lxstd
1.0                                                                     :: xstd
```

10.3 Program Results

```
nag_prob_normal_vector (g01sac) Example Program Results

           tail           x        xmu       xstd          p     ivalid
------------------------------------------------------------------------
     Nag_LowerTail      1.96       0.00       1.00      0.975          0
     Nag_UpperTail      1.96       0.00       1.00      0.025          0
 Nag_TwoTailConfid      1.96       0.00       1.00      0.950          0
 Nag_TwoTailSignif      1.96       0.00       1.00      0.050          0
```