NAG Library Function Document

nag_bessel_i0_vector (s18asc)

1 Purpose

nag_bessel_i0_vector (s18asc) returns an array of values of the modified Bessel function $I_0(x)$.

2 Specification

```
#include <nag.h>
#include <nags.h>
void nag_bessel_i0_vector (Integer n, const double x[], double f[],
     Integer ivalid[], NagError *fail)
```

3 Description

nag_bessel_i0_vector (s18asc) evaluates an approximation to the modified Bessel function of the first kind $I_0(x_i)$ for an array of arguments x_i, for $i = 1, 2, \ldots, n$.

Note: $I_0(-x) = I_0(x)$, so the approximation need only consider $x \geq 0$.

The function is based on three Chebyshev expansions:

For $0 < x \leq 4$,

$$I_0(x) = e^x \sum_{r=0} a_r T_r(t), \qquad \text{where } t = 2\left(\frac{x}{4}\right) - 1.$$

For $4 < x \leq 12$,

$$I_0(x) = e^x \sum_{r=0} b_r T_r(t), \qquad \text{where } t = \frac{x - 8}{4}.$$

For $x > 12$,

$$I_0(x) = \frac{e^x}{\sqrt{x}} \sum_{r=0} c_r T_r(t), \qquad \text{where } t = 2\left(\frac{12}{x}\right) - 1.$$

For small x, $I_0(x) \simeq 1$. This approximation is used when x is sufficiently small for the result to be correct to ***machine precision***.

For large x, the function must fail because of the danger of overflow in calculating e^x.

4 References

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

5 Arguments

1: **n** – Integer *Input*

On entry: n, the number of points.

Constraint: $\mathbf{n} \geq 0$.

2: **x**[**n**] – const double *Input*

On entry: the argument x_i of the function, for $i = 1, 2, \ldots, \mathbf{n}$.

3: **f**[**n**] – double *Output*

On exit: $I_0(x_i)$, the function values.

4: **ivalid**[**n**] – Integer *Output*

On exit: **ivalid**$[i-1]$ contains the error code for x_i, for $i = 1, 2, \ldots,$ **n**.

ivalid$[i-1] = 0$
No error.

ivalid$[i-1] = 1$
x_i is too large. **f**$[i-1]$ contains the approximate value of $I_0(x_i)$ at the nearest valid argument. The threshold value is the same as for **fail.code** = NE_REAL_ARG_GT in nag_bessel_i0 (s18aec), as defined in the Users' Note for your implementation.

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

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

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.

NW_IVALID

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

7 Accuracy

Let δ and ϵ be the relative errors in the argument and result respectively.

If δ is somewhat larger than the ***machine precision*** (i.e., if δ is due to data errors etc.), then ϵ and δ are approximately related by:

$$\epsilon \simeq \left|\frac{xI_1(x)}{I_0(x)}\right|\delta.$$

Figure 1 shows the behaviour of the error amplification factor

$$\left|\frac{xI_1(x)}{I_0(x)}\right|.$$

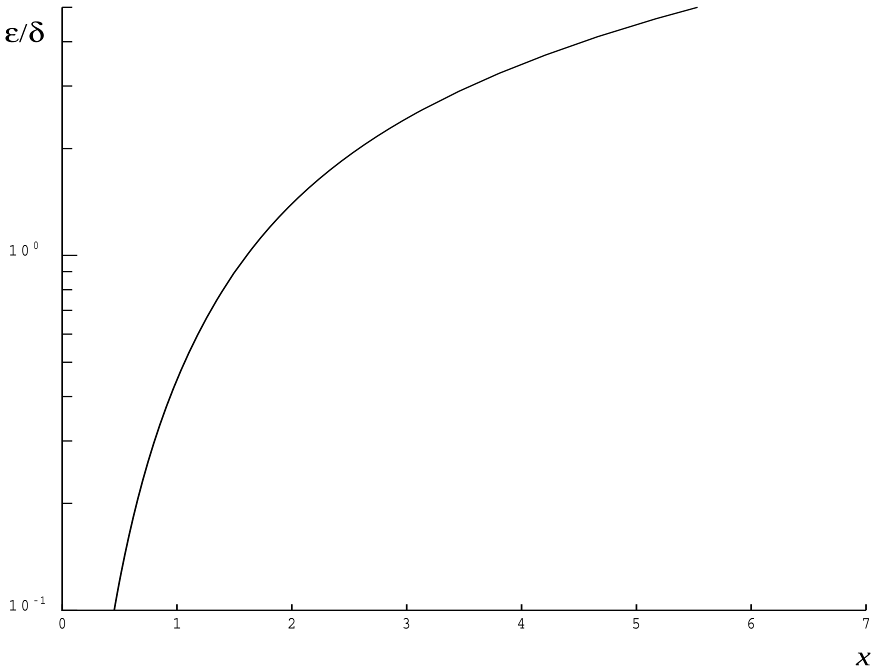

Figure 1

However if δ is of the same order as ***machine precision***, then rounding errors could make ϵ slightly larger than the above relation predicts.

For small x the amplification factor is approximately $\frac{x^2}{2}$, which implies strong attenuation of the error, but in general ϵ can never be less than the ***machine precision***.

For large x, $\epsilon \simeq x\delta$ and we have strong amplification of errors. However, for quite moderate values of x ($x > \hat{x}$, the threshold value), the function must fail because $I_0(x)$ would overflow; hence in practice the loss of accuracy for x close to $\hat{x}$ is not excessive and the errors will be dominated by those of the standard function exp.

8 Parallelism and Performance

Not applicable.

9 Further Comments

None.

10 Example

This example reads values of **x** from a file, evaluates the function at each value of x_i and prints the results.

10.1 Program Text

```
/* nag_bessel_i0_vector (s18asc) Example Program.
 *
 * Copyright 2011, Numerical Algorithms Group.
 *
 * Mark 23 2011.
 */
#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nags.h>
```

```
int main(void)
{
  Integer  exit_status = 0;
  Integer  i, n;
  double   *f = 0, *x = 0;
  Integer  *ivalid = 0;
  NagError fail;

  INIT_FAIL(fail);

  /* Skip heading in data file */
  scanf("%*[^\n]");

  printf("nag_bessel_i0_vector (s18asc) Example Program Results\n");
  printf("\n");
  printf("     x           f          ivalid\n");
  printf("\n");
  scanf("%ld", &n);
  scanf("%*[^\n]");

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

  for (i=0; i<n; i++)
    scanf("%lf", &x[i]);
  scanf("%*[^\n]");

  /* nag_bessel_i0_vector (s18asc).
   * modified Bessel Function I0(x)
   */
  nag_bessel_i0_vector(n, x, f, ivalid, &fail);
  if (fail.code!=NE_NOERROR && fail.code!=NW_IVALID)
    {
      printf("Error from nag_bessel_i0_vector (s18asc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

  for (i=0; i<n; i++)
    printf(" %11.3e %11.3e %4ld\n", x[i], f[i], ivalid[i]);

 END:
  NAG_FREE(f);
  NAG_FREE(x);
  NAG_FREE(ivalid);

  return exit_status;
}
```

10.2 Program Data

```
nag_bessel_i0_vector (s18asc) Example Program Data

10

0.0 0.5 1.0 3.0 6.0 8.0 10.0 15.0 20.0 -1.0
```

10.3 Program Results

```
nag_bessel_i0_vector (s18asc) Example Program Results

     x           f          ivalid

   0.000e+00   1.000e+00    0
   5.000e-01   1.063e+00    0
   1.000e+00   1.266e+00    0
   3.000e+00   4.881e+00    0
   6.000e+00   6.723e+01    0
   8.000e+00   4.276e+02    0
   1.000e+01   2.816e+03    0
   1.500e+01   3.396e+05    0
   2.000e+01   4.356e+07    0
  -1.000e+00   1.266e+00    0
```