NAG Library Function Document

nag_bessel_i1_vector (s18atc)

1 Purpose

nag_bessel_i1_vector (s18atc) returns an array of values for the modified Bessel function $I_1(x)$.

2 Specification

```c
#include <nag.h>
#include <nags.h>
void nag_bessel_i1_vector (Integer n, const double x[], double f[],
                          Integer invalid[], NagError *fail)
```

3 Description

nag_bessel_i1_vector (s18atc) evaluates an approximation to the modified Bessel function of the first kind $I_1(x_i)$ for an array of arguments $x_i$, for $i = 1, 2, \ldots, n$.

Note: $I_1(-x) = -I_1(x)$, so the approximation need only consider $x \geq 0$.

The function is based on three Chebyshev expansions:

For $0 < x \leq 4$,

$$I_1(x) = x \sum_{r=0} a_r T_r(t), \quad \text{where } t = 2 \left( \frac{x}{4} \right)^2 - 1;$$

For $4 < x \leq 12$,

$$I_1(x) = e^x \sum_{r=0} b_r T_r(t), \quad \text{where } t = \frac{x - 8}{4};$$

For $x > 12$,

$$I_1(x) = e^x \sqrt{x} \sum_{r=0} c_r T_r(t), \quad \text{where } t = 2 \left( \frac{12}{x} \right)^2 - 1.$$ 

For small $x$, $I_1(x) \approx x$. 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 $I_1(x)$ cannot be represented without overflow.

4 References


5 Arguments

1:  \text{n} – Integer  \hspace{2cm} \text{Input}

   \text{On entry: } n, \text{ the number of points.}

   \text{Constraint: } n \geq 0.

2:  \text{x[n]} – const double  \hspace{2cm} \text{Input}

   \text{On entry: } the \text{ argument } x_i \text{ of the function, for } i = 1, 2, \ldots, n.$
3:  \( f[n] \) – double

Output

On exit: \( I_1(x_i) \), the function values.

4:  \( \text{ivalid}[n] \) – Integer

Output

On exit: \( \text{ivalid}[i-1] \) contains the error code for \( x_i \), for \( i = 1, 2, \ldots, n \).

\( \text{ivalid}[i-1] = 0 \)

No error.

\( \text{ivalid}[i-1] = 1 \)

\( x_i \) is too large. \( f[i-1] \) contains the approximate value of \( I_1(x_i) \) at the nearest valid argument. The threshold value is the same as for fail.code = NE_REAL_ARG_GT in nag_bessel_i1 (s18afc), as defined in the Users’ Note for your implementation.

5:  \( \text{fail} \) – 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_BAD_PARAM

On entry, argument \( \langle \text{value} \rangle \) had an illegal value.

NE_INT

On entry, \( n = \langle \text{value} \rangle \).

Constraint: \( n \geq 0 \).

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.

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 \( x \) was invalid.

Check ivalid for more information.

7 Accuracy

Let \( \delta \) and \( \epsilon \) be the relative errors in the argument and result respectively.

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

\[
\epsilon \approx \left| \frac{xI_0(x) - I_1(x)}{I_1(x)} \right| \delta.
\]

Figure 1 shows the behaviour of the error amplification factor
However, if $\delta$ is of the same order as \textit{machine precision}, then rounding errors could make $\epsilon$ slightly larger than the above relation predicts.

For small $x$, $\epsilon \simeq \delta$ and there is no amplification of errors.

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 $I_1(x)$ 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

```c
/* nag_bessel_il_vector (s18atc) Example Program. */
/* Copyright 2014 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 */
    #ifdef _WIN32
        scanf_s("%*[\n]");
    #else
        scanf("%*[\n]");
    #endif
    printf("nag_bessel_i1_vector (s18atc) Example Program Results\n");
    printf("\n");
    printf(" x f ivalid\n");
    printf("\n");
    #ifdef _WIN32
        scanf_s("%"NAG_IFMT", &n);
    #else
        scanf("%"NAG_IFMT", &n);
    #endif
    #ifdef _WIN32
        scanf_s("%*[\n]");
    #else
        scanf("%*[\n]");
    #endif
    /* 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++)
    #ifdef _WIN32
        scanf_s("%lf", &x[i]);
    #else
        scanf("%lf", &x[i]);
    #endif
    #ifdef _WIN32
        scanf_s("%*[\n]");
    #else
        scanf("%*[\n]");
    #endif
    /* nag_bessel_i1_vector (s18atc).  *
        * modified Bessel Function I1(x)  */
    nag_bessel_i1_vector(n, x, f, ivalid, &fail);
    if (fail.code!=NE_NOERROR && fail.code!=NW_IVALID)
    {
        printf("Error from nag_bessel_i1_vector (s18atc).
fail.message\n");
        exit_status = 1;
        goto END;
    }

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

END:

s18atc.4 Mark 25
END:
NAG_FREE(f);
NAG_FREE(x);
NAG_FREE(ivalid);
return exit_status;
}

10.2 Program Data
nag_bessel_i1_vector (s18atc) 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_i1_vector (s18atc) Example Program Results

<table>
<thead>
<tr>
<th>x</th>
<th>f</th>
<th>invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0</td>
</tr>
<tr>
<td>5.000e-01</td>
<td>2.579e-01</td>
<td>0</td>
</tr>
<tr>
<td>1.000e+00</td>
<td>5.652e-01</td>
<td>0</td>
</tr>
<tr>
<td>3.000e+00</td>
<td>3.953e+00</td>
<td>0</td>
</tr>
<tr>
<td>6.000e+00</td>
<td>6.134e+01</td>
<td>0</td>
</tr>
<tr>
<td>8.000e+00</td>
<td>3.999e+02</td>
<td>0</td>
</tr>
<tr>
<td>1.000e+01</td>
<td>2.671e+03</td>
<td>0</td>
</tr>
<tr>
<td>1.500e+01</td>
<td>3.281e+05</td>
<td>0</td>
</tr>
<tr>
<td>2.000e+01</td>
<td>4.245e+07</td>
<td>0</td>
</tr>
<tr>
<td>-1.000e+00</td>
<td>-5.652e-01</td>
<td>0</td>
</tr>
</tbody>
</table>