NAG Library Function Document

nag_complex_bessel_i (s18dec)

1 Purpose
nag_complex_bessel_i (s18dec) returns a sequence of values for the modified Bessel functions $I_{\nu+n}(z)$ for complex $z$, non-negative $\nu$ and $n = 0, 1, \ldots, N - 1$, with an option for exponential scaling.

2 Specification
#include <nag.h>
#include <nags.h>

void nag_complex_bessel_i (double fnu, Complex z, Integer n, Nag_ScaleResType scal, Complex cy[], Integer *nz, NagError *fail)

3 Description
nag_complex_bessel_i (s18dec) evaluates a sequence of values for the modified Bessel function $I_{\nu}(z)$, where $z$ is complex, $-\pi < \arg z \leq \pi$, and $\nu$ is the real, non-negative order. The $N$-member sequence is generated for orders $\nu, \nu + 1, \ldots, \nu + N - 1$. Optionally, the sequence is scaled by the factor $e^{-|\Re(z)|}$.

The function is derived from the function CBESI in Amos (1986).

Note: although the function may not be called with $\nu$ less than zero, for negative orders the formula $I_{-\nu}(z) = I_{\nu}(z) + \frac{2}{\pi} \sin(\pi \nu) K_{\nu}(z)$ may be used (for the Bessel function $K_{\nu}(z)$, see nag_complex_bessel_k (s18dcc)).

When $N$ is greater than 1, extra values of $I_{\nu}(z)$ are computed using recurrence relations.

For very large $|z|$ or $(\nu + N - 1)$, argument reduction will cause total loss of accuracy, and so no computation is performed. For slightly smaller $|z|$ or $(\nu + N - 1)$, the computation is performed but results are accurate to less than half of machine precision. If $\Re(z)$ is too large and the unscaled function is required, there is a risk of overflow and so no computation is performed. In all the above cases, a warning is given by the function.

4 References

5 Arguments
1: fnu – double
   Input
   On entry: $\nu$, the order of the first member of the sequence of functions.
   Constraint: fnu $\geq 0.0$.

2: z – Complex
   Input
   On entry: the argument $z$ of the functions.
3: \( n \) – Integer

\( \text{Input} \)

\( On \ entry: \) \( N \), the number of members required in the sequence \( I_\nu(z), I_{\nu+1}(z), \ldots, I_{\nu+N-1}(z) \).

\( Constraint: \ n \geq 1. \)

4: \( \text{scal} \) – Nag_ScaleResType

\( \text{Input} \)

\( On \ entry: \) the scaling option.

\( \text{scal} = \text{Nag_UnscaleRes} \)

The results are returned unscaled.

\( \text{scal} = \text{Nag_ScaleRes} \)

The results are returned scaled by the factor \( e^{-|\text{Re}(z)|} \).

\( Constraint: \ \text{scal} = \text{Nag_UnscaleRes} \) or \( \text{Nag_ScaleRes} \).

5: \( \text{cy}[n] \) – Complex

\( \text{Output} \)

\( On \ exit: \) the \( N \) required function values: \( \text{cy}[i - 1] \) contains \( I_{\nu+i-1}(z) \), for \( i = 1, 2, \ldots, N \).

6: \( \text{nz} \) – Integer*

\( \text{Output} \)

\( On \ exit: \) the number of components of \( \text{cy} \) that are set to zero due to underflow.

If \( \text{nz} > 0 \), then elements \( \text{cy}[n - \text{nz}], \text{cy}[n - \text{nz} + 1], \ldots, \text{cy}[n - 1] \) are set to zero.

7: \( \text{fail} \) – NagError*

\( \text{Input/Output} \)

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: \( n \geq 1. \)

**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.

**NE_OVERFLOW_LIKELEY**

No computation because \( z.re = \langle \text{value} \rangle > \langle \text{value} \rangle \), \( \text{scal} = \text{Nag_UnscaleRes} \).
On entry, $\text{fnu} = \langle\text{value}\rangle$.  
Constraint: $\text{fnu} \geq 0.0$.

**NE_TERMINATION_FAILURE**

No computation – algorithm termination condition not met.

**NE_TOTAL_PRECISION_LOSS**

No computation because $|z| = \langle\text{value}\rangle > \langle\text{value}\rangle$.
No computation because $\text{fnu} + n - 1 = \langle\text{value}\rangle > \langle\text{value}\rangle$.

**NW_SOME_PRECISION_LOSS**

Results lack precision because $|z| = \langle\text{value}\rangle > \langle\text{value}\rangle$.
Results lack precision because $\text{fnu} + n - 1 = \langle\text{value}\rangle > \langle\text{value}\rangle$.

### 7 Accuracy

All constants in nag_complex_bessel_i (s18dec) are given to approximately 18 digits of precision. Calling the number of digits of precision in the floating-point arithmetic being used $t$, then clearly the maximum number of correct digits in the results obtained is limited by $p = \min(t, 18)$. Because of errors in argument reduction when computing elementary functions inside nag_complex_bessel_i (s18dec), the actual number of correct digits is limited, in general, by $p - s$, where $s \approx \max(1, |\log_{10}|z||, |\log_{10}\nu|)$ represents the number of digits lost due to the argument reduction. Thus the larger the values of $|z|$ and $\nu$, the less the precision in the result. If nag_complex_bessel_i (s18dec) is called with $n > 1$, then computation of function values via recurrence may lead to some further small loss of accuracy.

If function values which should nominally be identical are computed by calls to nag_complex_bessel_i (s18dec) with different base values of $\nu$ and different $n$, the computed values may not agree exactly. Empirical tests with modest values of $\nu$ and $z$ have shown that the discrepancy is limited to the least significant 3 – 4 digits of precision.

### 8 Parallelism and Performance

Not applicable.

### 9 Further Comments

The time taken for a call of nag_complex_bessel_i (s18dec) is approximately proportional to the value of $n$, plus a constant. In general it is much cheaper to call nag_complex_bessel_i (s18dec) with $n$ greater than 1, rather than to make $N$ separate calls to nag_complex_bessel_i (s18dec).

Paradoxically, for some values of $z$ and $\nu$, it is cheaper to call nag_complex_bessel_i (s18dec) with a larger value of $n$ than is required, and then discard the extra function values returned. However, it is not possible to state the precise circumstances in which this is likely to occur. It is due to the fact that the base value used to start recurrence may be calculated in different regions for different $n$, and the costs in each region may differ greatly.

Note that if the function required is $I_0(x)$ or $I_1(x)$, i.e., $\nu = 0.0$ or 1.0, where $x$ is real and positive, and only a single function value is required, then it may be much cheaper to call nag_bessel_i0 (s18aec), nag_bessel_i1 (s18afc), nag_bessel_i0_scaled (s18cec) or nag_bessel_i1_scaled (s18cfc), depending on whether a scaled result is required or not.
10 Example

This example prints a caption and then proceeds to read sets of data from the input data stream. The first datum is a value for the order \( \text{fnu} \), the second is a complex value for the argument, \( z \), and the third is a character value used as a flag to set the argument \( \text{scal} \). The program calls the function with \( n = 2 \) to evaluate the function for orders \( \text{fnu} \) and \( \text{fnu} + 1 \), and it prints the results. The process is repeated until the end of the input data stream is encountered.

10.1 Program Text

```c
/* nag_complex_bessel_i (s18dec) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 7, 2002. */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nags.h>

int main(void)
{
    Integer exit_status = 0;
    Complex z, cy[2];
    double fnu;
    const Integer n = 2;
    Integer nz;
    char nag_enum_arg[40];
    Nag_ScaleResType scal;
    NagError fail;

    INIT_FAIL(fail);

    /* Skip heading in data file */
    #ifdef _WIN32
        scanf_s("%*[\n]");
    #else
        scanf("%*[\n]");
    #endif
    printf("nag_complex_bessel_i (s18dec) Example Program Results\n");
    printf("Calling with n = \%"NAG_IFMT"\n", n);
    printf(" fnu z scal cy[0] cy[1] nz\n");
    #ifdef _WIN32
        while (scanf_s(" %lf (%lf,%lf) %39s%*[\n] ", &fnu, &z.re, &z.im,
                      nag_enum_arg, __countof(nag_enum_arg)) != EOF)
    #else
        while (scanf(" %lf (%lf,%lf) %39s%*[\n] ", &fnu, &z.re, &z.im,
                      nag_enum_arg) != EOF)
    #endif
    {
        /* nag_enum_name_to_value (x04nac). 
         * Converts NAG enum member name to value */
        scal = (Nag_ScaleResType) nag_enum_name_to_value(nag_enum_arg);

        /* nag_complex_bessel_i (s18dec). 
         * Modified Bessel functions I_\( (\text{nu}+a) \)(z), real a >= 0, 
         * complex z, nu = 0,1,2,... */
        nag_complex_bessel_i(fnu, z, n, scal, cy, &nz, &fail);
        if (fail.code != NE_NOERROR)
        {
            printf("Error from nag_complex_bessel_i (s18dec).\n\n", fail.message);
            exit_status = 1;
        }
    }

    return exit_status;
}
```

goto END;
}
}
printf("%7.4f (%7.3f,%7.3f) %-14s (%7.3f,%7.3f) (%7.3f,%7.3f) "
"%NAG_IFMT"
", fnu, z.re, z.im, nag_enum_arg, cy[0].re,
    cy[0].im, cy[1].re, cy[1].im, nz);
}

END:
return exit_status;
}

10.2 Program Data

nag_complex_bessel_i (s18dec) Example Program Data
0.00  ( 0.3, -0.4) Nag_UnscaleRes
2.30  ( 2.0,  0.0) Nag_UnscaleRes
2.12  (-1.0,  0.0) Nag_UnscaleRes
5.50  (-6.1,  9.8) Nag_UnscaleRes
5.50  (-6.1,  9.8) Nag_ScaleRes  - Values of fnu, z and scal

10.3 Program Results

nag_complex_bessel_i (s18dec) Example Program Results
Calling with n = 2

<table>
<thead>
<tr>
<th>fnu</th>
<th>z</th>
<th>scal</th>
<th>cy[0]</th>
<th>cy[1]</th>
<th>nz</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>( 0.300, -0.400) Nag_UnscaleRes</td>
<td>( 0.982, -0.059)</td>
<td>( 0.143, -0.203)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2.3000</td>
<td>( 2.000,  0.000) Nag_UnscaleRes</td>
<td>( 0.500,  0.000)</td>
<td>( 0.142,  0.000)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2.1200</td>
<td>(-1.000,  0.000) Nag_UnscaleRes</td>
<td>( 0.103,  0.041)</td>
<td>(-0.016, -0.006)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5.5000</td>
<td>(-6.100,  9.800) Nag_UnscaleRes</td>
<td>( 22.534, 13.710)</td>
<td>(-19.635, -1.660)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5.5000</td>
<td>(-6.100,  9.800) Nag_ScaleRes</td>
<td>( 0.051,  0.031)</td>
<td>(-0.044, -0.004)</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>