NAG Library Function Document

nag_complex_bessel_j (s17dec)

1 Purpose

nag_complex_bessel_j (s17dec) returns a sequence of values for the Bessel functions $J_{\nu+n}(z)$ for complex z, non-negative ν 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_j (double fnu, Complex z, Integer n,
     Nag_ScaleResType scal, Complex cy[], Integer *nz, NagError *fail)
```

3 Description

nag_complex_bessel_j (s17dec) evaluates a sequence of values for the Bessel function $J_\nu(z)$, where z is complex, $-\pi < \arg z \leq \pi$, and ν is the real, non-negative order. The N-member sequence is generated for orders ν, $\nu + 1, \ldots, \nu + N - 1$. Optionally, the sequence is scaled by the factor $e^{-|\mathrm{Im}(z)|}$.

Note: although the function may not be called with ν less than zero, for negative orders the formula $J_{-\nu}(z) = J_\nu(z)\cos(\pi\nu) - Y_\nu(z)\sin(\pi\nu)$ may be used (for the Bessel function $Y_\nu(z)$, see nag_complex_bessel_y (s17dcc)).

The function is derived from the function CBESJ in Amos (1986). It is based on the relations $J_\nu(z) = e^{\nu\pi i/2} I_\nu(-iz)$, $\mathrm{Im}(z) \geq 0.0$, and $J_\nu(z) = e^{-\nu\pi i/2} I_\nu(iz)$, $\mathrm{Im}(z) < 0.0$.

The Bessel function $I_\nu(z)$ is computed using a variety of techniques depending on the region under consideration.

When N is greater than 1, extra values of $J_\nu(z)$ 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 $\mathrm{Im}(z)$ is large, 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

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

Amos D E (1986) Algorithm 644: A portable package for Bessel functions of a complex argument and non-negative order *ACM Trans. Math. Software* **12** 265–273

5 Arguments

1: **fnu** – double *Input*

On entry: ν, the order of the first member of the sequence of functions.

Constraint: **fnu** ≥ 0.0.

2: **z** – Complex *Input*

On entry: the argument z of the functions.

3: **n** – Integer *Input*

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

Constraint: $\mathbf{n} \geq 1$.

4: **scal** – Nag_ScaleResType *Input*

On entry: the scaling option.

scal = Nag_UnscaleRes
The results are returned unscaled.

scal = Nag_ScaleRes
The results are returned scaled by the factor $e^{-|\mathrm{Im}(z)|}$.

Constraint: **scal** = Nag_UnscaleRes or Nag_ScaleRes.

5: **cy**[**n**] – Complex *Output*

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

6: **nz** – Integer * *Output*

On exit: the number of components of **cy** that are set to zero due to underflow. If $\mathbf{nz} > 0$, then elements **cy**[**n** − **nz**], **cy**[**n** − **nz** + 1], ..., **cy**[**n** − 1] are set to zero.

7: **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_BAD_PARAM

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

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: $\mathbf{n} \geq 1$.

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 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.

NE_OVERFLOW_LIKELY

No computation because **z**.im = ⟨*value*⟩ > ⟨*value*⟩, **scal** = Nag_UnscaleRes.

NE_REAL

On entry, **fnu** $= \langle value \rangle$.
Constraint: **fnu** ≥ 0.0.

NE_TERMINATION_FAILURE

No computation – algorithm termination condition not met.

NE_TOTAL_PRECISION_LOSS

No computation because $|\mathbf{z}| = \langle value \rangle > \langle value \rangle$.

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

NW_SOME_PRECISION_LOSS

Results lack precision because $|\mathbf{z}| = \langle value \rangle > \langle value \rangle$.

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

7 Accuracy

All constants in nag_complex_bessel_j (s17dec) 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_j (s17dec), 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 ν, the less the precision in the result. If nag_complex_bessel_j (s17dec) is called with $\mathbf{n} > 1$, 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_j (s17dec) with different base values of ν and different **n**, the computed values may not agree exactly. Empirical tests with modest values of ν and z have shown that the discrepancy is limited to the least significant 3 – 4 digits of precision.

8 Parallelism and Performance

nag_complex_bessel_j (s17dec) is not threaded in any implementation.

9 Further Comments

The time taken for a call of nag_complex_bessel_j (s17dec) is approximately proportional to the value of **n**, plus a constant. In general it is much cheaper to call nag_complex_bessel_j (s17dec) with **n** greater than 1, rather than to make N separate calls to nag_complex_bessel_j (s17dec).

Paradoxically, for some values of z and ν, it is cheaper to call nag_complex_bessel_j (s17dec) 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 $J_0(x)$ or $J_1(x)$, i.e., $\nu = 0.0$ or 1.0, where x is real and positive, and only a single unscaled function value is required, then it may be much cheaper to call nag_bessel_j0 (s17aec) or nag_bessel_j1 (s17afc) respectively.

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 **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 **scal**. The program calls the function with $\mathbf{n} = 2$ to evaluate the function for orders **fnu** and $\mathbf{fnu} + 1$, and it prints the results. The process is repeated until the end of the input data stream is encountered.

10.1 Program Text

```
/* nag_complex_bessel_j (s17dec) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#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_j (s17dec) 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,
                 (unsigned)_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_j (s17dec).
     * Bessel functions J_(nu+a)(z), real a >= 0, complex z,
     * nu = 0,1,2,...
     */
    nag_complex_bessel_j(fnu, z, n, scal, cy, &nz, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_complex_bessel_j (s17dec).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
```

```
    }
    printf("%7.4f (%7.3f,%7.3f) %-14s (%7.3f,%7.3f) (%7.3f,%7.3f) "
           "%" NAG_IFMT "\n", 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_j (s17dec) 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
 1.58    (-2.3,  5.6)    Nag_UnscaleRes
 1.58    (-2.3,  5.6)    Nag_ScaleRes
```

10.3 Program Results

```
nag_complex_bessel_j (s17dec) Example Program Results
Calling with n = 2
   fnu           z               scal            cy[0]                cy[1]     nz
 0.0000 (  0.300,  0.400) Nag_UnscaleRes (  1.017, -0.061) (  0.157,  0.197) 0
 2.3000 (  2.000,  0.000) Nag_UnscaleRes (  0.272, -0.000) (  0.089, -0.000) 0
 2.1200 ( -1.000,  0.000) Nag_UnscaleRes (  0.088,  0.035) ( -0.014, -0.006) 0
 1.5800 ( -2.300,  5.600) Nag_UnscaleRes ( -1.551,-36.476) ( 25.910,  2.677) 0
 1.5800 ( -2.300,  5.600) Nag_ScaleRes   ( -0.006, -0.135) (  0.096,  0.010) 0
```