NAG Library Function Document

nag_specfun_1f1_real_scaled (s22bbc)

1 Purpose

nag_specfun_1f1_real_scaled (s22bbc) returns a value for the confluent hypergeometric function \( {}_1F_1(a; b; x) \), with real parameters \( a \) and \( b \) and real argument \( x \). The solution is returned in the scaled form \( {}_1F_1(a; b; x) = \frac{m_f}{C_2^2} \frac{m_s}{x} \), where \( m_f \) is the real scaled component and \( m_s \) is the integer power of two scaling. This function is sometimes also known as Kummer’s function \( M(a, b; x) \).

2 Specification

```c
#include <nag.h>
#include <nags.h>

void nag_specfun_1f1_real_scaled (double ani, double adr, double bni,
                                 double bdr, double x, double *frm, Integer *scm, NagError *fail)
```

3 Description

nag_specfun_1f1_real_scaled (s22bbc) returns a value for the confluent hypergeometric function \( {}_1F_1(a; b; x) \), with real parameters \( a \) and \( b \) and real argument \( x \), in the scaled form \( {}_1F_1(a; b; x) = \frac{m_f}{C_2^2} \frac{m_s}{x} \), where \( m_f \) is the real scaled component and \( m_s \) is the integer power of two scaling. This function is unbounded or not uniquely defined for \( b \) equal to zero or a negative integer.

The confluent hypergeometric function is defined by the confluent series,

\[
{}_1F_1(a; b; x) = M(a, b, x) = \sum_{s=0}^{\infty} \frac{(a)_s x^s}{(b)_s s!} = 1 + \frac{a}{b} x + \frac{a(a+1)}{b(b+1)2!} x^2 + \cdots
\]

where \((a)_s = 1(a)(a+1)(a+2)\ldots(a+s-1)\) is the rising factorial of \( a \). \( M(a, b, x) \) is a solution to the second order ODE (Kummer’s Equation):

\[
\frac{d^2M}{dx^2} + (b - x) \frac{dM}{dx} - aM = 0.
\]  

Given the parameters and argument \((a, b, x)\), this function determines a set of safe values \([a_i, b_i, \zeta_i] | i \leq 2\) and selects an appropriate algorithm to accurately evaluate the functions \( M_i(a_i, b_i, \zeta_i) \). The result is then used to construct the solution to the original problem \( M(a, b, x) \) using, where necessary, recurrence relations and/or continuation.

For improved precision in the final result, this function accepts \( a \) and \( b \) split into an integral and a decimal fractional component. Specifically \( a = a_i + a_r \), where \( |a_r| \leq 0.5 \) and \( a_i = a - a_r \) is integral. \( b \) is similarly deconstructed.

Additionally, an artificial bound, \( \text{arbd} \) is placed on the magnitudes of \( a_i, b_i \) and \( x \) to minimize the occurrence of overflow in internal calculations. \( \text{arbd} = 0.0001 \times I_{\text{max}} \), where \( I_{\text{max}} = X02BBC \). It should, however, not be assumed that this function will produce an accurate result for all values of \( a_i, b_i \) \( \text{and} x \) satisfying this criterion.

Please consult the NIST Digital Library of Mathematical Functions or the companion (2010) for a detailed discussion of the confluent hypergeometric function including special cases, transformations, relations and asymptotic approximations.
4 References


5 Arguments

1: ani – double
   
   On entry: ai, the nearest integer to a, satisfying ai = a − ar.
   
   Constraints:
   
   ani = |ani|;
   |ani| ≤ arbd.

2: adr – double
   
   On entry: ar, the signed decimal remainder satisfying ar = a − ai and |ar| ≤ 0.5.
   
   Constraint: |adr| ≤ 0.5.
   
   Note: if |adr| < 100.0e, ar = 0.0 will be used, where e is the machine precision as returned by nag_machine_precision (X02AJC).

3: bni – double
   
   On entry: bi, the nearest integer to b, satisfying bi = b − br.
   
   Constraints:
   
   bni = |bni|;
   |bni| ≤ arbd;
   if bdr = 0.0, bni > 0.

4: bdr – double
   
   On entry: br, the signed decimal remainder satisfying br = b − bi and |br| ≤ 0.5.
   
   Constraint: |bdr| ≤ 0.5.
   
   Note: if |bdr − adr| < 100.0e, ar = b, will be used, where e is the machine precision as returned by nag_machine_precision (X02AJC).

5: x – double
   
   On entry: the argument x of the function.
   
   Constraint: |x| ≤ arbd.

6: frm – double *
   
   On exit: mf, the scaled real component of the solution satisfying mf = M(a, b, x) × 2−ms.
   
   Note: if overflow occurs upon completion, as indicated by fail.code = NW_OVERFLOW_WARN, the value of mf returned may still be correct. If overflow occurs in a subcalculation, as indicated by fail.code = NE_OVERFLOW, this should not be assumed.

7: scm – Integer *
   
   On exit: ms, the scaling power of two, satisfying ms = log2 \( \frac{M(a, b, x)}{mf} \).
Note: if overflow occurs upon completion, as indicated by
fail. code = NW_OVERFLOW_WARN, then \( m_s \geq I_{\text{max}} \), where \( I_{\text{max}} \) is the largest representable integer (see nag_max_integer (X02BBC)). If overflow occurs during a subcalculation, as indicated by
fail. code = NE_OVERFLOW, \( m_s \) may or may not be greater than \( I_{\text{max}} \). In either case, 
scm = nag_max_integer will have been returned.

8: fail – NagError *

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_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**

Overflow occurred in a subcalculation of \( M(a,b,x) \).
The answer may be completely incorrect.

**NE_REAL**

On entry, \( \text{adr} = \langle \text{value} \rangle \).
Constraint: \( |\text{adr}| \leq 0.5 \).

On entry, \( \text{bdr} = \langle \text{value} \rangle \).
Constraint: \( |\text{bdr}| \leq 0.5 \).

**NE_REAL_2**

On entry, \( b = \text{bni} + \text{bdr} = \langle \text{value} \rangle \).
\( M(a,b,x) \) is undefined when \( b \) is zero or a negative integer.

**NE_REAL_ARG_NON_INTEGRAL**

\( \text{ani} \) is non-integral.
On entry, \( \text{ani} = \langle \text{value} \rangle \).
Constraint: \( \text{ani} = \lfloor \text{ani} \rfloor \).

\( \text{bni} \) is non-integral.
On entry, \( \text{bni} = \langle \text{value} \rangle \).
Constraint: \( \text{bni} = \lfloor \text{bni} \rfloor \).

Mark 25

s22bbc.3
NE_REAL_RANGECONS
On entry, \( an_i = \langle value \rangle \).
Constraint: \( |an_i| \leq \text{arbnd} = \langle value \rangle \).

On entry, \( bn_i = \langle value \rangle \).
Constraint: \( |bn_i| \leq \text{arbnd} = \langle value \rangle \).

On entry, \( x = \langle value \rangle \).
Constraint: \( |x| \leq \text{arbnd} = \langle value \rangle \).

NE_TOTAL_PRECISIONLOSS
All approximations have completed, and the final residual estimate indicates no accuracy can be guaranteed.
Relative residual = \( \langle value \rangle \).

NW_OVERFLOW_WARN
On completion, overflow occurred in the evaluation of \( M(a, b, x) \).

NW_SOME_PRECISIONLOSS
All approximations have completed, and the final residual estimate indicates some precision may have been lost.
Relative residual = \( \langle value \rangle \).

NW_UNDERFLOW_WARN
Underflow occurred during the evaluation of \( M(a, b, x) \).
The returned value may be inaccurate.

7 Accuracy
In general, if \( \text{fail.code} = \text{NE_NOERROR} \), the value of \( M \) may be assumed accurate, with the possible loss of one or two decimal places. Assuming the result does not under or overflow, an error estimate \( res \) is made internally using equation (1). If the magnitude of \( res \) is sufficiently large a different \( \text{fail.code} \) will be returned. Specifically,

\[
\begin{align*}
\text{fail.code} &= \text{NE_NOERROR} & \text{res} &\leq 1000e \\
\text{fail.code} &= \text{NW_SOME_PRECISION_LOSS} & 1000e &< \text{res} \leq 0.1 \\
\text{fail.code} &= \text{NE_TOTAL_PRECISION_LOSS} & \text{res} &> 0.1
\end{align*}
\]

A further estimate of the residual can be constructed using equation (1), and the differential identity,

\[
\begin{align*}
\frac{dM(a, b, x)}{dx} &= \frac{a}{b} M(a + 1, b + 1, x), \\
\frac{d^2M(a, b, x)}{dx^2} &= \frac{a(a + 1)}{b(b + 1)} M(a + 2, b + 2, x).
\end{align*}
\]

This estimate is however dependent upon the error involved in approximating \( M(a + 1, b + 1, x) \) and \( M(a + 2, b + 2, x) \).

8 Parallelism and Performance
\text{nag_specfun_1f1_real_scaled (s22bbc)} is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.
\text{nag_specfun_1f1_real_scaled (s22bbc)} makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.
Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users’ Note for your implementation for any additional implementation-specific information.

9 Further Comments

The values of \( m_f \) and \( m_s \) are implementation dependent. In most cases, if \( \, _1F_1(a; b; x) = 0 \), \( m_f = 0 \) and \( m_s = 0 \) will be returned, and if \( \, _1F_1(a; b; x) \) is finite, the fractional component will be bound by \( 0.5 \leq |m_f| < 1 \), with \( m_s \) chosen accordingly.

The values returned in \( \text{frm} (m_f) \) and \( \text{scm} (m_s) \) may be used to explicitly evaluate \( M(a, b, x) \), and may also be used to evaluate products and ratios of multiple values of \( M \) as follows,

\[
M(a, b, x) = m_f \times 2^{m_s},
\]

\[
M(a_1, b_1, x_1) \times M(a_2, b_2, x_2) = (m_{f1} \times m_{f2}) \times 2^{(m_{s1}+m_{s2})},
\]

\[
\frac{M(a_1, b_1, x_1)}{M(a_2, b_2, x_2)} = \frac{m_{f1}}{m_{f2}} \times 2^{(m_{s1}-m_{s2})},
\]

\[
\ln|M(a, b, x)| = \ln|m_f| + m_s \times \ln(2).
\]

10 Example

This example evaluates the confluent hypergeometric function at two points in scaled form using \nag_specfun_1f1_real_scaled (s22bbc), and subsequently calculates their product and ratio without having to explicitly construct \( M \).

10.1 Program Text

```c
/* nag_specfun_1f1_real_scaled (s22bbc) Example Program. */
* Copyright 2014 Numerical Algorithms Group.
* Mark 24, 2013.
*/
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nags.h>
#include <nagx02.h>

int main(void)
{
    /* Scalars */
    Integer exit_status = 0;
    Integer k, maxexponent, scm;
    double ani, adr, bni, bdr, delta, frm, x;
    /* Arrays */
    double frmv[2];
    Integer scmv[2];
    /* Nag Types */
    NagError fail;

    maxexponent = X02BLC;
    printf("nag_specfun_1f1_real_scaled (s22bbc) Example Program Results\n\n");

    ani = -10.0;
    bni = 30.0;
    delta = 1.0E-4;
    adr = delta;
    bdr = -delta;
    x = 25.0;
```
for (k = 0; k < 2; k++)
{
    INIT_FAIL(fail);
    /* Compute the real confluent hypergeometric function M(a,b,x) in scaled
     * form using nag_specfun_1f1_real_scaled (s22bbc).
     */
    nag_specfun_1f1_real_scaled(ani, adr, bni, bdr, x, &frm, &scm, &fail);
    switch (fail.code) {
      case NE_NOERROR:
      case NW_UNDERFLOW_WARN:
      case NW_SOME_PRECISION_LOSS:
      {
        if (scm < maxexponent)
          printf(" %9.4f %9.4f %9.4f %13.4e %5"NAG_IFMT" %13.4e
", 
                 ani+adr, bni+bdr, x, frm, scm, frm*pow(2.0, scm));
        else
          printf(" %9.4f %9.4f %9.4f %13.4e %5"NAG_IFMT" %17s
", 
                 ani+adr, bni+bdr, x, frm, scm, "Not Representable");
        frmv[k] = frm;
        scmv[k] = scm;
        break;
      }
      default:
      {
        /* Either the result has overflowed, no accuracy may be assumed,
         * or an input error has been detected.
         */
        printf(" %9.4f %9.4f %9.4f %17s
", ani+adr, bni+bdr, x, "FAILED");
        exit_status = 1;
        goto END;
      }
    }
    adr = -adr;
    bdr = -bdr;
}

/* Calculate the product M1*M2*/
frm = frmv[0] * frmv[1];
scm = scmv[0] + scmv[1];
printf("\n");
if (scm < maxexponent)
  printf("%30s%12.4e%6"NAG_IFMT"%12.4e\n",
         "Solution product", frm, scm, frm*pow(2.0, scm));
else
  printf("%30s%12.4e%6"NAG_IFMT"%17s",
         "Solution product", frm, scm, "Not Representable");

/* Calculate the ratio M1/M2*/
if (frmv[1] != 0.0)
{
  frm = frmv[0]/frmv[1];
  scm = scmv[0] - scmv[1];
  printf("\n");
  if (scm < maxexponent)
    printf("%30s%12.4e%6"NAG_IFMT"%12.4e\n",
           "Solution ratio", frm, scm, frm*pow(2.0, scm));
  else
    printf("%30s%12.4e%6"NAG_IFMT"%17s",
           "Solution ratio", frm, scm, "Not Representable");
}

END:
  return exit_status;
}

10.2 Program Data

None.
10.3 Program Results

nag_specfun_1f1_real_scaled (s22bbc) Example Program Results

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>x</th>
<th>frm</th>
<th>scm</th>
<th>M(a,b,x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10.0001</td>
<td>30.0001</td>
<td>25.0000</td>
<td>-7.7318e-01</td>
<td>-15</td>
<td>-2.3596e-05</td>
</tr>
</tbody>
</table>

Solution product 5.9789e-01 -30 5.5683e-10
Solution ratio 1.0001e+00 0 1.0001e+00