NAG Library Function Document

nag_specfun_2f1_real (s22bec)

1 Purpose

nag_specfun_2f1_real (s22bec) returns a value for the Gauss hypergeometric function ${}_2F_1(a,b;c;x)$ for real parameters a,b and c, and real argument x.

2 Specification

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

```
double nag_specfun_2f1_real (double a, double b, double c, double x,
       NagError *fail)
```

3 Description

nag_specfun_2f1_real (s22bec) returns a value for the Gauss hypergeometric function ${}_2F_1(a,b;c;x)$ for real parameters a, b and c, and for real argument x.

The associated function nag_specfun_2f1_real_scaled (s22bfc) performs the same operations, but returns ${}_2F_1(a,b;c;x)$ in the scaled form ${}_2F_1(a,b;c;x) = f_{\mathrm{fr}} \times 2^{f_{\mathrm{sc}}}$ to allow calculations to be performed when ${}_2F_1(a,b;c;x)$ is not representable as a single working precision number. It also accepts the parameters a, b and c as summations of an integer and a decimal fraction, giving higher accuracy when any are close to an integer.

The Gauss hypergeometric function is a solution to the hypergeometric differential equation,

$$x(1-x)\frac{d^2f}{dx^2} + (c-(a+b+1)x)\frac{df}{dx} - abf = 0. \tag{1}$$

For $|x| < 1$, it may be defined by the Gauss series,

$${}_2F_1(a,b;c;x) = \sum_{s=0}^{\infty}\frac{(a)_s(b)_s}{(c)_s s!}x^s = 1 + \frac{ab}{c}x + \frac{a(a+1)b(b+1)}{c(c+1)2!}x^2 + \cdots, \tag{2}$$

where $(a)_s = 1(a)(a+1)(a+2)\dots(a+s-1)$ is the rising factorial of a. ${}_2F_1(a,b;c;x)$ is undefined for $c = 0$ or c a negative integer.

For $|x| < 1$, the series is absolutely convergent and ${}_2F_1(a,b;c;x)$ is finite.

For $x < 1$, linear transformations of the form,

$${}_2F_1(a,b;c;x) = C_1(a_1,b_1,c_1,x_1)\,{}_2F_1(a_1,b_1;c_1;x_1) + C_2(a_2,b_2,c_2,x_2)\,{}_2F_1(a_2,b_2;c_2;x_2) \tag{3}$$

exist, where $x_1,\ x_2 \in (0,1]$. C_1 and C_2 are real valued functions of the parameters and argument, typically involving products of gamma functions. When these are degenerate, finite limiting cases exist. Hence for $x < 0$, ${}_2F_1(a,b;c;x)$ is defined by analytic continuation, and for $x < 1$, ${}_2F_1(a,b;c;x)$ is real and finite.

For $x = 1$, the following apply:

If $c > a+b$, ${}_2F_1(a,b;c;1) = \dfrac{\Gamma(c)\Gamma(c-a-b)}{\Gamma(c-a)\Gamma(c-b)}$, and hence is finite. Solutions also exist for the degenerate cases where $c-a$ or $c-b$ are negative integers or zero.

If $c \le a+b$, ${}_2F_1(a,b;c;1)$ is infinite, and the sign of ${}_2F_1(a,b;c;1)$ is determinable as x approaches 1 from below.

In the complex plane, the principal branch of ${}_2F_1(a,b;c;z)$ is taken along the real axis from $x = 1.0$ increasing. ${}_2F_1(a,b;c;z)$ is multivalued along this branch, and for real parameters a, b and c is typically not real valued. As such, this function will not compute a solution when $x > 1$.

The solution strategy used by this function is primarily dependent upon the value of the argument x. Once trivial cases and the case $x = 1.0$ are eliminated, this proceeds as follows.

For $0 < x \leq 0.5$, sets of safe parameters $\{\alpha_{i,j}; \beta_{i,j}; \zeta_{i,j}; \chi_j | 1 \leq j \leq 2|; 1 \leq i \leq 4\}$ are determined, such that the values of ${}_2F_1(a_j, b_j; c_j; x_j)$ required for an appropriate transformation of the type (3) may be calculated either directly or using recurrence relations from the solutions of ${}_2F_1(\alpha_{i,j}, \beta_{i,j}; \zeta_{i,j}; \chi_j)$. If c is positive, then only transformations with $C_2 = 0.0$ will be used, implying only ${}_2F_1(a_1, b_1; c_1; x_1)$ will be required, with the transformed argument $x_1 = x$. If c is negative, in some cases a transformation with $C_2 \neq 0.0$ will be used, with the argument $x_2 = 1.0 - x$. The function then cycles through these sets until acceptable solutions are generated. If no computation produces an accurate answer, the least inaccurate answer is selected to complete the computation. See Section 7.

For $0.5 < x < 1.0$, an identical approach is first used with the argument x. Should this fail, a linear transformation resulting in both transformed arguments satisfying $x_j = 1.0 - x$ is employed, and the above strategy for $0 < x \leq 0.5$ is utilized on both components. Further transformations in these sub-computations are however limited to single terms with no argument transformation.

For $x < 0$, a linear transformation mapping the argument x to the interval $(0, 0.5]$ is first employed. The strategy for $0 < x \leq 0.5$ is then used on each component, including possible further two term transforms. To avoid some degenerate cases, a transform mapping the argument x to $[0.5, 1)$ may also be used.

In addition to the above restrictions on c and x, an artificial bound, *arbnd*, is placed on the magnitudes of a, b, c and x to minimize the occurrence of overflow in internal calculations, particularly those involving real to integer conversions. $arbnd = 0.0001 \times I_{\max}$, where $I_{\max}$ is the largest machine integer (see nag_max_integer (X02BBC)). It should however not be assumed that this function will produce accurate answers for all values of a, b, c and x satisfying this criterion.

This function also tests for non-finite values of the parameters and argument on entry, and assigns non-finite values upon completion if appropriate. See Section 9 and Chapter x07.

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

4 References

NIST Handbook of Mathematical Functions (2010) (eds F W J Olver, D W Lozier, R F Boisvert, C W Clark) Cambridge University Press

Pearson J (2009) Computation of hypergeometric functions *MSc Dissertation, Mathematical Institute, University of Oxford*

5 Arguments

1: **a** – double *Input*

On entry: the parameter a.

Constraint: $|\mathbf{a}| \leq arbnd$.

2: **b** – double *Input*

On entry: the parameter b.

Constraint: $|\mathbf{b}| \leq arbnd$.

3: **c** – double *Input*

On entry: the parameter c.

Constraints:

$|\mathbf{c}| \leq arbnd$;
$\mathbf{c} \neq 0, -1, -2, \ldots$.

4: **x** – double *Input*

On entry: the argument x.

Constraint: $-arbnd < \mathbf{x} \leq 1$.

5: **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_CANNOT_CALCULATE

An internal calculation has resulted in an undefined result.

NE_COMPLEX

On entry, $\mathbf{x} = \langle value \rangle$.
In general, ${}_2F_1(a, b; c; x)$ is not real valued when $x > 1$.

NE_INFINITE

On entry, $\mathbf{x} = \langle value \rangle$, $c = \langle value \rangle$, $a + b = \langle value \rangle$.
${}_2F_1(a, b; c; 1)$ is infinite in the case $c \leq a + b$.

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 ${}_2F_1(a, b; c; x)$. The result may or may not be infinite.

NE_REAL

On entry, $\mathbf{c} = \langle value \rangle$.
${}_2F_1(a, b; c; x)$ is undefined when c is zero or a negative integer.

NE_REAL_RANGE_CONS

On entry, **a** does not satisfy $|\mathbf{a}| \leq arbnd = \langle value \rangle$.

On entry, **b** does not satisfy $|\mathbf{b}| \leq arbnd = \langle value \rangle$.

On entry, **c** does not satisfy $|\mathbf{c}| \leq arbnd = \langle value \rangle$.

On entry, **x** does not satisfy $|\mathbf{x}| \leq arbnd = \langle value \rangle$.

NE_TOTAL_PRECISION_LOSS

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 ${}_2F_1(a,b;c;x)$.

NW_SOME_PRECISION_LOSS

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 ${}_2F_1(a,b;c;x)$. The returned value may be inaccurate.

7 Accuracy

In general, if **fail.code** = NE_NOERROR, the value of ${}_2F_1(a,b;c;x)$ 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 **fail.code** will be returned. Specifically,

fail.code = NE_NOERROR or NW_UNDERFLOW_WARN	$res \leq 1000\epsilon$
fail.code = NW_SOME_PRECISION_LOSS	$1000\epsilon < res \leq 0.1$
fail.code = NE_TOTAL_PRECISION_LOSS	$res > 0.1$

where ϵ is the ***machine precision*** as returned by nag_machine_precision (X02AJC).

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

$$\begin{aligned} \frac{d\left({}_2F_1(a,b;c;x)\right)}{dx} &= \frac{ab}{c}{}_2F_1(a+1,b+1;c+1;x) \\ \frac{d^2\left({}_2F_1(a,b;c;x)\right)}{dx^2} &= \frac{a(a+1)b(b+1)}{c(c+1)}{}_2F_1(a+2,b+2;c+2;x) \end{aligned} \tag{4}$$

This estimate is however dependent upon the error involved in approximating ${}_2F_1(a+1,b+1;c+1;x)$ and ${}_2F_1(a+2,b+2;c+2;x)$.

Furthermore, the accuracy of the solution, and the error estimate, can be dependent upon the accuracy of the decimal fraction of the input parameters a and b. For example, if $c = c_i + c_r = 100 + 1.0\text{e}{-6}$, then on a machine with 16 decimal digits of precision, the internal calculation of c_r will only be accurate to 8 decimal places. This can subsequently pollute the final solution by several decimal places without affecting the residual estimate as greatly. Should you require higher accuracy in such regions, then you should use nag_specfun_2f1_real_scaled (s22bfc), which requires you to supply the correct decimal fraction.

8 Parallelism and Performance

Not applicable.

9 Further Comments

nag_specfun_2f1_real (s22bec) returns non-finite values when appropriate. See Chapter x07 for more information on the definitions of non-finite values.

Should a non-finite value be returned, this will be indicated in the value of **fail**, as detailed in the following cases.

If **fail.code** = NE_NOERROR, or **fail.code** = NE_TOTAL_PRECISION_LOSS, NW_SOME_PRECISION_LOSS or NW_UNDERFLOW_WARN, a finite value will have been returned with an approximate accuracy as detailed in Section 7.

If **fail.code** = NE_INFINITE then ${}_2F_1(a,b;c;x)$ is infinite, and a signed infinity will have been returned. The sign of the infinity should be correct when taking the limit as x approaches 1 from below.

If **fail.code** = NW_OVERFLOW_WARN then upon completion, $\left|{}_2F_1(a,b;c;x)\right| > R_{\mathrm{max}}$, where R_{max} is the largest machine number given by nag_real_largest_number (X02ALC), and hence is too large to be representable. The result will be returned as a signed infinity. The sign should be correct.

If **fail.code** = NE_OVERFLOW then overflow occurred during a subcalculation of ${}_2F_1(a,b;c;x)$. A signed infinity will have been returned, however there is no guarantee that this is representative of either the magnitude or the sign of ${}_2F_1(a,b;c;x)$.

For all other error exits, nag_specfun_2f1_real (s22bec) will return a signalling NaN (see nag_create_nan (x07bbc)).

If **fail.code** = NE_CANNOT_CALCULATE then an internal computation produced an undefined result. This may occur when two terms overflow with opposite signs, and the result is dependent upon their summation for example.

If **fail.code** = NE_REAL then c is too close to a negative integer or zero on entry, and ${}_2F_1(a,b;c;x)$ is considered undefined. Note, this will also be the case when c is a negative integer, and a (possibly trivial) linear transformation of the form (3) would result in either:

(i) all c_j not being negative integers,

(ii) for any c_j which remain as negative integers, one of the corresponding parameters a_j or b_j is a negative integer of magnitude less than c_j.

In the first case, the transformation coefficients $C_j(a_j,b_j,c_j,x_j)$ are typically either infinite or undefined, preventing a solution being constructed. In the second case, the series (2) will terminate before the degenerate term, resulting in a polynomial of fixed degree, and hence potentially a finite solution.

If **fail.code** = NE_REAL_RANGE_CONS then no computation will have been performed. The actual solution may however be finite.

fail.code = NE_COMPLEX indicates $x > 1$. Hence the requested solution is on the boundary of the principal branch of ${}_2F_1(a,b;c;x)$, and hence is multivalued, typically with a non-zero imaginary component. It is however strictly finite.

10 Example

This example evaluates ${}_2F_1(a,b;c;x)$ at a fixed set of parameters a, b and c, and for several values for the argument x.

10.1 Program Text

```
/* nag_specfun_2f1_real (s22bec) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 24, 2013.
 */

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

void construct_table(double a, double b, double c);
```

```
int main(void)
{
  /* Scalars */
  Integer  exit_status = 0;
  Integer  kx;
  double   a, b, c, f, x;
  /* Nag Types */
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_specfun_2f1_real (s22bec) Example Program Results\n\n");

  a = 1.2;
  b = -2.6;
  c = 3.5;

  construct_table(a, b, c);

  for (kx = 1; kx < 22; kx++)
    {
      x = -4.0 + ((double) (kx-1))*0.25;
      /* Evaluate Real Gauss hypergeometric function 2F1(a,b;c;x) using
       * nag_specfun_2f1_real (s22bec).
       */
      f = nag_specfun_2f1_real(a, b, c, x, &fail);
      switch (fail.code) {
      case NE_NOERROR:
      case NW_UNDERFLOW_WARN:
      case NW_SOME_PRECISION_LOSS:
      case NE_TOTAL_PRECISION_LOSS:
        printf("    %10.2f     %10.4f\n", x, f);
        break;
      case NE_INFINITE:
      case NW_OVERFLOW_WARN:
      case NE_OVERFLOW:
        if(f>=0.0)
          printf("    %10.2f   %13s\n", x, "+Infinity");
        else
          printf("    %10.2f   %13s\n", x, "-Infinity");
        break;
      case NE_CANNOT_CALCULATE:
        printf("    %10.2f   %13s\n", x, "NaN");
        break;
      default:
        printf(" Illegal parameter.");
        exit_status = 1;
        goto END;
        break;
      }
    }
 END:
  return exit_status;
}

void construct_table(double a, double b, double c)
{
  printf("            a              b              c\n");
  printf("+--------------+--------------+--------------+\n");
  printf("    %10.2f     %10.2f     %10.2f\n\n", a, b, c);
  printf("            x    2F1(a,b;c;x)\n");
  printf("+--------------+--------------+\n");
  return;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_specfun_2f1_real (s22bec) Example Program Results

            a             b             c
+--------------+--------------+--------------+
         1.20         -2.60          3.50

            x    2F1(a,b;c;x)
+--------------+--------------+
        -4.00       12.3289
        -3.75       11.0602
        -3.50        9.8783
        -3.25        8.7806
        -3.00        7.7649
        -2.75        6.8286
        -2.50        5.9692
        -2.25        5.1841
        -2.00        4.4707
        -1.75        3.8263
        -1.50        3.2480
        -1.25        2.7330
        -1.00        2.2784
        -0.75        1.8811
        -0.50        1.5378
        -0.25        1.2453
         0.00        1.0000
         0.25        0.7983
         0.50        0.6362
         0.75        0.5094
         1.00        0.3659
```