For \( x < 1 \), the Gauss hypergeometric function is a solution to the hypergeometric differential equation,

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

For \(|x| < 1\), it may be defined by the Gauss series,

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

where \((a)_s = 1(a)(a+1)(a+2)\cdots(a+s-1)\) 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,

\[
\,_2F_1(a,b;c;x) = C_1(a_1,b_1,c_1,x_1)\,_2F_1(a_1,b_1;c_1;x_1) + C_2(a_2,b_2,c_2,x_2)\,_2F_1(a_2,b_2;c_2;x_2)
\]

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

If \(c \leq a + b\), \(_2F_1(a,b;c;1)\) 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 \( \{\alpha_{ij}; \beta_{ij}; \gamma_{ij}; \delta_{ij}; \epsilon_{ij}; \zeta_{ij}; \chi_{ij} \mid 1 \leq j \leq 2; 1 \leq i \leq 4\} \) are determined, such that the values of \( _2F_1(a_j, b_j; c_j; x) \) required for an appropriate transformation of the type (3) may be calculated either directly or by using recurrence relations from the solutions of \( _2F_1(a_{ij}, b_{ij}; c_{ij}; x_{ij}) \). 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. \( arbnd = 0.0001 \times I_{\text{max}} \), where \( I_{\text{max}} \) 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


### 5 Arguments

1. \( a \) – double \hspace{5cm} Input
   
   On entry: the parameter \( a \).
   
   Constraint: \( |a| \leq arbnd \).

2. \( b \) – double \hspace{5cm} Input
   
   On entry: the parameter \( b \).
   
   Constraint: \( |b| \leq arbnd \).

3. \( c \) – double \hspace{5cm} Input
   
   On entry: the parameter \( c \).
Constraints:
\[ |c| \leq \text{arbnd}; \]
\[ c \neq 0, -1, -2, \ldots \]

4: \textbf{x} – double
\textit{Input}

\textit{On entry:} the argument \textit{x}.
\textit{Constraint:} \( -\text{arbnd} < x \leq 1 \).

5: \textbf{fail} – NagError *
\textit{Input/Output}

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

\textbf{NE_ALLOC_FAIL}
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE_CANNOT_CALCULATE}
An internal calculation has resulted in an undefined result.

\textbf{NE_COMPLEX}

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

\textbf{NE_INFINITE}

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

\textbf{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.

\textbf{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.

\textbf{NE_OVERFLOW}

Overflow occurred in a subcalculation of \( _2F_1(a, b; c; x) \). The result may or may not be infinite.

\textbf{NE_REAL}

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

\textbf{NE_REAL_RANGE_CONS}

\textit{On entry,} \( a \) does not satisfy \( |a| \leq \text{arbnd} = \langle \text{value} \rangle \).
\textit{On entry,} \( b \) does not satisfy \( |b| \leq \text{arbnd} = \langle \text{value} \rangle \).
\textit{On entry,} \( c \) does not satisfy \( |c| \leq \text{arbnd} = \langle \text{value} \rangle \).
On entry, $x$ does not satisfy $|x| \leq \text{arbnd} = \langle\text{value}\rangle$.

**NE_TOTAL_PRECISION_LOSS**

All approximations have completed, and the final residual estimate indicates no accuracy can be guaranteed.

Relative residual $= \langle\text{value}\rangle$.

**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 $= \langle\text{value}\rangle$.

**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 $\text{fail code} = \text{NE_NOERROR}$, 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 $\text{res}$ is made internally using equation (1). If the magnitude of $\text{res}$ is sufficiently large, a different $\text{fail code}$ will be returned. Specifically,

\[
\text{fail code} = \text{NE_NOERROR} \text{ or } \text{NW_UNDERFLOW_WARN} \quad \text{res} \leq 1000\epsilon
\]

\[
\text{fail code} = \text{NW_SOME_PRECISION_LOSS} \quad 1000\epsilon < \text{res} \leq 0.1
\]

\[
\text{fail code} = \text{NE_TOTAL_PRECISION_LOSS} \quad \text{res} > 0.1
\]

where $\epsilon$ 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,

\[
\frac{d\left(_2F_1(a, b; c; x)\right)}{dx} = \frac{ab}{c} _2F_1(a+1, b+1; c+1; x)
\]

\[
\frac{d^2\left(_2F_1(a, b; c; x)\right)}{dx^2} = \frac{a(a+1)b(b+1)}{c(c+1)} _2F_1(a+2, b+2; c+2; x)
\]

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 $c = c_i + c_r = 100 + 1.0e-6$, 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 \texttt{fail}, as detailed in the following cases.

If \texttt{fail.code} = \texttt{NE\_NO\_ERROR}, or \texttt{fail.code} = \texttt{NE\_TOTAL\_PRECISION\_LOSS}, \texttt{NW\_SOME\_PRECISION\_LOSS} or \texttt{NW\_UNDERFLOW\_WARN}, a finite value will have been returned with an approximate accuracy as detailed in Section 7.

If \texttt{fail.code} = \texttt{NE\_INFINITE} then $\mathbb{F}_1(a, b; c; x)$ 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 \texttt{fail.code} = \texttt{NW\_OVERFLOW\_WARN} then upon completion, $|\mathbb{F}_1(a, b; c; x)| > R_{\text{max}}$, where $R_{\text{max}}$ is the largest machine number given by \texttt{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 \texttt{fail.code} = \texttt{NE\_OVERFLOW} then overflow occurred during a subcalculation of $\mathbb{F}_1(a, b; c; x)$. A signed infinity will have been returned, however there is no guarantee that this is representative of either the magnitude or the sign of $\mathbb{F}_1(a, b; c; x)$.

For all other error exits, \texttt{nag\_specfun\_2f1\_real (s22bec)} will return a signalling NaN (see \texttt{nag\_create\_nan (x07bbc)}).

If \texttt{fail.code} = \texttt{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 \texttt{fail.code} = \texttt{NE\_REAL} then $c$ is too close to a negative integer or zero on entry, and $\mathbb{F}_1(a, b; c; x)$ 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 \texttt{fail.code} = \texttt{NE\_REAL\_RANGE\_CONS} then no computation will have been performed. The actual solution may however be finite.

\texttt{fail.code} = \texttt{NE\_COMPLEX} indicates $x > 1$. Hence the requested solution is on the boundary of the principal branch of $\mathbb{F}_1(a, b; c; x)$, and hence is multivalued, typically with a non-zero imaginary component. It is however strictly finite.

10 Example

This example evaluates $\mathbb{F}_1(a, b; c; x)$ at a fixed set of parameters $a$, $b$ and $c$, and for several values for the argument $x$.

10.1 Program Text

/* \texttt{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.\n");
                exit_status = 1;
                goto END;
                break;
        }
    }

    END:
    return exit_status;
}

void construct_table(double a, double b, double c)
{
    printf("+-----------------+-----------------+-----------\n");
    printf(" %10.2f %10.2f %10.2f\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**

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