NAG Library Routine Document

S22BEF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

S22BEF 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

```
FUNCTION S22BEF (A, B, C, X, IFAIL)
REAL (KIND=nag_wp) S22BEF
INTEGER            IFAIL
REAL (KIND=nag_wp) A, B, C, X
```

3 Description

S22BEF 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 routine S22BFF 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 $(a)_s = 1(a)(a+1)(a+2)\ldots(a+s-1)$ 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 $c \leq a+b$, ${}_2F_1(a,b;c;1)$ 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 routine will not compute a solution when $x > 1$.

The solution strategy used by this routine 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 $\left\{\alpha_{i,j};\beta_{i,j};\zeta_{i,j};\chi_j | 1 \leq j \leq 2 | ; 1 \leq i \leq 4\right\}$ are determined, such that the values of ${}_2F_1\left(a_j,b_j;c_j;x_j\right)$ required for an appropriate transformation of the type (3) may be calculated either directly or using recurrence relations from the solutions of ${}_2F_1\left(\alpha_{i,j},\beta_{i,j};\zeta_{i,j};\chi_j\right)$. 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 routine 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 X02BBF). It should however not be assumed that this routine will produce accurate answers for all values of a, b, c and x satisfying this criterion.

This routine 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 – REAL (KIND=nag_wp) *Input*

On entry: the parameter a.

Constraint: $|\mathrm{A}| \leq arbnd$.

2: B – REAL (KIND=nag_wp) *Input*

On entry: the parameter b.

Constraint: $|\mathrm{B}| \leq arbnd$.

3: C – REAL (KIND=nag_wp) *Input*

On entry: the parameter c.

Constraints:

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

4: X – REAL (KIND=nag_wp) *Input*

On entry: the argument x.

Constraint: $-arbnd < \mathrm{X} \leq 1$.

5: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, if you are not familiar with this argument, the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: $\mathrm{IFAIL} = 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry $\mathrm{IFAIL} = 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

$\mathrm{IFAIL} = 1$

Underflow occurred during the evaluation of ${}_2F_1(a, b; c; x)$. The returned value may be inaccurate.

$\mathrm{IFAIL} = 2$

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

$\mathrm{IFAIL} = 3$

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

$\mathrm{IFAIL} = 4$

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

$\mathrm{IFAIL} = 5$

On completion, overflow occurred in the evaluation of ${}_2F_1(a, b; c; x)$.

$\mathrm{IFAIL} = 6$

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

$\mathrm{IFAIL} = 9$

An internal calculation has resulted in an undefined result.

IFAIL $= 11$

On entry, A does not satisfy $|\mathrm{A}| \leq arbnd = \langle value \rangle$.

IFAIL $= 21$

On entry, B does not satisfy $|\mathrm{B}| \leq arbnd = \langle value \rangle$.

IFAIL $= 31$

On entry, C does not satisfy $|\mathrm{C}| \leq arbnd = \langle value \rangle$.

IFAIL $= 32$

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

IFAIL $= 41$

On entry, X does not satisfy $|\mathrm{X}| \leq arbnd = \langle value \rangle$.

IFAIL $= 42$

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

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

In general, if IFAIL $= 0$, 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 nonzero IFAIL will be returned. Specifically,

IFAIL $= 0$ or 1	$res \leq 1000\epsilon$
IFAIL $= 2$	$1000\epsilon < res \leq 0.1$
IFAIL $= 3$	$res > 0.1$

where ϵ is the ***machine precision*** as returned by X02AJF.

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 S22BFF, which requires you to supply the correct decimal fraction.

8 Parallelism and Performance

S22BEF is not threaded in any implementation.

9 Further Comments

S22BEF 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 IFAIL, as detailed in the following cases.

If IFAIL $= 0$, or IFAIL $= 1$, 2 or 3, a finite value will have been returned with an approximate accuracy as detailed in Section 7.

If IFAIL $= 4$ 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 IFAIL $= 5$ then upon completion, $\left|{}_2F_1(a, b; c; x)\right| > R_{\text{max}}$, where R_{max} is the largest machine number given by X02ALF, and hence is too large to be representable. The result will be returned as a signed infinity. The sign should be correct.

If IFAIL $= 6$ 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, S22BEF will return a signalling NaN (see X07BBF).

If IFAIL $= 9$ 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 IFAIL $= 32$ 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 IFAIL $= 11$, 21, 31 or 41 then no computation will have been performed. The actual solution may however be finite.

IFAIL $= 42$ 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 nonzero 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

```
    Program s22befe

!     S22BEF Example Program Text

!     Mark 26 Release. NAG Copyright 2016.

!     .. Use Statements ..
      Use nag_library, Only: nag_wp, s22bef
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter               :: nout = 6
!     .. Local Scalars ..
      Real (Kind=nag_wp)               :: a, b, c, f, x
      Integer                          :: ifail, kx
!     .. Intrinsic Procedures ..
      Intrinsic                        :: real
!     .. Executable Statements ..
      Write (nout,*) 'S22BEF Example Program Results'

      a = 1.2E0_nag_wp
      b = -2.6E0_nag_wp
      c = 3.5E0_nag_wp

      Write (nout,*)
      Write (nout,99999)
      Write (nout,99998)
      Write (nout,99997) a, b, c
      Write (nout,*)
      Write (nout,99996)
      Write (nout,99995)

      Do kx = 1, 21
        x = -4.0E0_nag_wp + real(kx-1,kind=nag_wp)*0.25E0_nag_wp
        ifail = 1
        f = s22bef(a,b,c,x,ifail)
        Select Case (ifail)
        Case (0,1,2,3)
          Write (nout,99994) x, f
        Case (4,5,6)
          If (f>=0.0E0_nag_wp) Then
            Write (nout,99993) x, '+Infinity'
          Else
            Write (nout,99993) x, '-Infinity'
          End If
        Case (9)
          Write (nout,99993) x, 'NaN'
        Case Default
          Write (nout,*) 'Illegal parameter: ifail = ', ifail
          Go To 100
        End Select
      End Do

100   Continue

99999 Format (12X,'a',14X,'b',14X,'c')
99998 Format (3('+--------------'),'+')
99997 Format (3(4X,F10.2,1X))
99996 Format (12X,'x',4X,'2F1(a,b;c;x)')
99995 Format (2('+--------------'),'+')
99994 Format (4X,F10.2,5X,F10.4)
99993 Format (4X,F10.2,3X,A13)
    End Program s22befe
```

10.2 Program Data

None.

10.3 Program Results

```
 S22BEF 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
```