NAG Library Routine Document

S22BFF

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

S22BFF 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. The result is returned in the scaled form ${}_2F_1(a,b;c;x) = f_{\mathrm{fr}} \times 2^{f_{\mathrm{sc}}}$.

2 Specification

```
SUBROUTINE S22BFF (ANI, ADR, BNI, BDR, CNI, CDR, X, FRF, SCF, IFAIL)
INTEGER            SCF, IFAIL
REAL (KIND=nag_wp) ANI, ADR, BNI, BDR, CNI, CDR, X, FRF
```

3 Description

S22BFF 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 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 $\{\alpha_{i,j}, \beta_{i,j}, \zeta_{i,j}, \chi_j | 1 \leq j \leq 2|, 1 \leq i \leq 4\}$ 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 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.

For improved precision in the final result, this routine accepts a, b and c 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. The other parameters b and c are similarly deconstructed.

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: ANI – REAL (KIND=nag_wp) *Input*

On entry: a_i, the nearest integer to a, satisfying $a_i = a - a_r$.

Constraints:

ANI $= \lfloor$ANI$\rfloor$;
$|$ANI$| \leq arbnd$.

2: ADR – REAL (KIND=nag_wp) *Input*

On entry: a_r, the signed decimal remainder satisfying $a_r = a - a_i$ and $|a_r| \leq 0.5$.

Constraint: $|$ADR$| \leq 0.5$.

3: BNI – REAL (KIND=nag_wp) *Input*

On entry: b_i, the nearest integer to b, satisfying $b_i = b - b_r$.

Constraints:

BNI $= \lfloor$BNI$\rfloor$;
$|$BNI$| \leq arbnd$.

4: BDR – REAL (KIND=nag_wp) *Input*

On entry: b_r, the signed decimal remainder satisfying $b_r = b - b_i$ and $|b_r| \leq 0.5$.

Constraint: $|$BDR$| \leq 0.5$.

5: CNI – REAL (KIND=nag_wp) *Input*

On entry: c_i, the nearest integer to c, satisfying $c_i = c - c_r$.

Constraints:

CNI $= \lfloor$CNI$\rfloor$;
$|$CNI$| \leq arbnd$;
if $|$CDR$| < 16.0\epsilon$, CNI ≥ 1.0.

6: CDR – REAL (KIND=nag_wp) *Input*

On entry: c_r, the signed decimal remainder satisfying $c_r = c - c_i$ and $|c_r| \leq 0.5$.

Constraint: $|$CDR$| \leq 0.5$.

7: X – REAL (KIND=nag_wp) *Input*

On entry: the argument x.

Constraint: $-arbnd <$ X ≤ 1.

8: FRF – REAL (KIND=nag_wp) *Output*

On exit: f_{fr}, the scaled real component of the solution satisfying $f_{\mathrm{fr}} = {}_2F_1(a,b;c;x) \times 2^{-f_{\mathrm{sc}}}$, i.e., ${}_2F_1(a,b;c;x) = f_{\mathrm{fr}} \times 2^{f_{\mathrm{sc}}}$. See Section 9 for the behaviour of f_{fr} when a finite or non-finite answer is returned.

9: SCF – INTEGER *Output*

On exit: f_{sc}, the scaling power of two, satisfying $f_{\mathrm{sc}} = \log_2\left(\dfrac{{}_2F_1(a,b;c;x)}{f_{\mathrm{fr}}}\right)$, i.e., ${}_2F_1(a,b;c;x) = f_{\mathrm{fr}} \times 2^{f_{\mathrm{sc}}}$. See Section 9 for the behaviour of f_{sc} when a non-finite answer is returned.

10: 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: 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 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:

IFAIL $= 1$

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

IFAIL $= 2$

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

IFAIL $= 3$

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

IFAIL $= 4$

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

IFAIL $= 5$

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

IFAIL $= 6$

Overflow occurred in a subcalculation of ${}_2F_1(a, b; c; x)$. The answer may be completely incorrect.

IFAIL $= 9$

An internal calculation has resulted in an undefined result.

IFAIL $= 11$

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

IFAIL $= 13$

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

IFAIL $= 21$

On entry, ADR does not satisfy $|\text{ADR}| \leq 0.5$.

IFAIL $= 31$

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

IFAIL $= 33$

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

IFAIL $= 41$

On entry, BDR does not satisfy $|\text{BDR}| \leq 0.5$.

IFAIL $= 51$

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

IFAIL $= 52$

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

IFAIL $= 53$

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

IFAIL $= 61$

On entry, CDR does not satisfy $|\text{CDR}| \leq 0.5$.

IFAIL $= 71$

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

IFAIL $= 72$

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 overflow, an error estimate res is made internally using equation (1). If the magnitude of this residual 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. Note that underflow may also have occurred if IFAIL $= 2$ or 3.

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 ${}_2F_1(a+2,b+2;c+2;x)$.

8 Parallelism and Performance

S22BFF is not threaded in any implementation.

9 Further Comments

S22BFF 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 approximate accuracy as detailed in Section 7.

The values of f_{fr} and f_{sc} are implementation dependent. In most cases, if ${}_2F_1(a,b;c;x) = 0$, $f_{fr} = 0$ and $f_{sc} = 0$ will be returned, and if ${}_2F_1(a,b;c;x)$ is finite, the fractional component will be bound by $0.5 \leq \left|f_{fr}\right| < 1$, with f_{sc} chosen accordingly.

The values returned in FRF (f_{fr}) and SCF (f_{sc}) may be used to explicitly evaluate ${}_2F_1(a,b;c;x)$, and may also be used to evaluate products and ratios of multiple values of ${}_2F_1$ as follows,

$$\begin{aligned}{}_2F_1(a,b;c;x) &= f_{\mathrm{fr}} \times 2^{f_{\mathrm{sc}}}\\ {}_2F_1(a_1,b_1;c_1;x_1) \times {}_2F_1(a_2,b_2;c_2;x_2) &= (f_{\mathrm{fr1}} \times f_{\mathrm{fr2}}) \times 2^{(f_{\mathrm{sc1}}+f_{\mathrm{sc2}})}\\ \frac{{}_2F_1(a_1,b_1;c_1;x_1)}{{}_2F_1(a_2,b_2;c_2;x_2)} &= \frac{f_{\mathrm{fr1}}}{f_{\mathrm{fr2}}} \times 2^{(f_{\mathrm{sc1}}-f_{\mathrm{sc2}})}\\ \ln\left|{}_2F_1(a,b;c;x)\right| &= \ln|f_{\mathrm{fr}}| + f_{\mathrm{sc}} \times \ln(2).\end{aligned}$$

If IFAIL $= 4$ then ${}_2F_1(a,b;c;x)$ is infinite. A signed infinity will have been returned for FRF, and SCF $= 0$. The sign of FRF should be correct when taking the limit as x approaches 1 from below.

If IFAIL $= 5$ then upon completion, $\left|{}_2F_1(a,b;c;x)\right| > 2^{I_{\max}}$, where $I_{\max}$ is given by X02BBF, and hence is too large to be representable even in the scaled form. The scaled real component returned in FRF may still be correct, whilst SCF $= I_{\max}$ will have been returned.

If IFAIL $= 6$ then overflow occurred during a subcalculation of ${}_2F_1(a,b;c;x)$. The same result as for IFAIL $= 5$ will have been returned, however there is no guarantee that this is representative of either the magnitude of the scaling power f_{sc}, or the scaled component f_{fr} of ${}_2F_1(a,b;c;x)$.

For all other error exits, SCF $= 0$ will be returned and FRF will be returned as a signalling NaN (see X07BBF).

If IFAIL $= 9$ 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 $= 52$ then c is too close to a negative integer or zero on entry, and ${}_2F_1(a,b;c;x)$ is 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 $= 11$, 31, 51 or 71 then no computation will have been performed due to the risk of integer overflow. The actual solution may however be finite.

IFAIL $= 72$ indicates $x > 1$, and hence the requested solution is on the boundary of the principal branch of ${}_2F_1(a, b; c; x)$. Hence it is multivalued, typically with a nonzero imaginary component. It is however strictly finite.

10 Example

This example evaluates the Gauss hypergeometric function at two points in scaled form using S22BFF, and subsequently calculates their product and ratio implicitly.

10.1 Program Text

```
    Program s22bffe

!     S22BFF Example Program Text

!     Mark 26 Release. NAG Copyright 2016.

!     .. Use Statements ..
      Use nag_library, Only: nag_wp, s22bff, x02bhf, x02blf, x07caf, x07cbf
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter                :: nout = 6
!     .. Local Scalars ..
      Real (Kind=nag_wp)                :: adr, ani, bdr, bni, cdr, cni, delta, &
                                           frf, scale, x
      Integer                           :: ifail, k, scf
      Logical                           :: finite_solutions
!     .. Local Arrays ..
      Real (Kind=nag_wp)                :: frfv(2)
      Integer                           :: exmode(3), scfv(2)
!     .. Intrinsic Procedures ..
      Intrinsic                         :: real
!     .. Executable Statements ..
      Write (nout,*) 'S22BFF Example Program Results'

!     Get current exception mode
      Call x07caf(exmode)
!     Disable exceptions
      Call x07cbf((/0,0,0/))

      finite_solutions = .True.

      ani = -10.0_nag_wp
      bni = 2.0_nag_wp
      cni = -5.0E0_nag_wp
      delta = 1.0E-4_nag_wp
      adr = delta
      bdr = -delta
      cdr = delta
      x = 0.45_nag_wp

      Write (nout,99999) 'a', 'b', 'c', 'x', 'frf', 'scf', ' 2F1(a,b;c;x)'

      Do k = 1, 2

        ifail = 1
        Call s22bff(ani,adr,bni,bdr,cni,cdr,x,frf,scf,ifail)
        Select Case (ifail)
        Case (0,1,2,3)
!         A finite result has been returned.
```

```
            If (scf<x02blf()) Then
              scale = frf*2.0E0_nag_wp**scf
              Write (nout,99998) ani + adr, bni + bdr, cni + cdr, x, frf, scf,   &
                scale
            Else
              Write (nout,99997) ani + adr, bni + bdr, cni + cdr, x, frf, scf,   &
                'Not representable'
            End If
          Case (4)
!           The result is analytically infinite.
            finite_solutions = .False.
            If (frf>=0.0E0_nag_wp) Then
              Write (nout,99993) ani + adr, bni + bdr, cni + cdr, x, 'Inf', scf, &
                'Inf'
            Else
              Write (nout,99993) ani + adr, bni + bdr, cni + cdr, x, '-Inf',     &
                scf, '-Inf'
            End If
          Case (5,6)
!           The final result has overflowed.
            finite_solutions = .False.
            If (frf>=0.0E0_nag_wp) Then
              Write (nout,99992) ani + adr, bni + bdr, cni + cdr, x, frf,        &
                'IMAX', '>2**IMAX'
            Else
              Write (nout,99992) ani + adr, bni + bdr, cni + cdr, x, frf,        &
                'IMAX', '<-2**IMAX'
            End If
          Case (9)
!           An internal calculation resulted in a non-finite, non-infinite
!           result.
            finite_solutions = .False.
            Write (nout,99993) ani + adr, bni + bdr, cni + cdr, x, 'NaN', scf,   &
              'NaN'
          Case Default
!           An input error has been detected.
            Write (nout,99996) ani + adr, bni + bdr, cni + cdr, x, 'FAILED'
            Go To 100
          End Select

          frfv(k) = frf
          scfv(k) = scf

          adr = -adr
          bdr = -bdr
          cdr = -cdr

        End Do

        If (finite_solutions) Then
!         Calculate the product M1*M2
          frf = frfv(1)*frfv(2)
          scf = scfv(1) + scfv(2)
          Write (nout,*)
          If (scf<x02blf()) Then
            scale = frf*real(x02bhf(),kind=nag_wp)**scf
            Write (nout,99995) 'Solution product', frf, scf, scale
          Else
            Write (nout,99994) 'Solution product', frf, scf, 'Not representable'
          End If

!         Calculate the ratio M1/M2
          If (frfv(2)/=0.0_nag_wp) Then
            frf = frfv(1)/frfv(2)
            scf = scfv(1) - scfv(2)
            Write (nout,*)
            If (scf<x02blf()) Then
              scale = frf*real(x02bhf(),kind=nag_wp)**scf
              Write (nout,99995) 'Solution ratio  ', frf, scf, scale
            Else
              Write (nout,99994) 'Solution ratio  ', frf, scf,                   &
```

```
              'Not representable'
          End If
        End If
      End If

100   Continue
!     Restore exception mode.
      Call x07cbf(exmode)

99999 Format (/,1X,4(A10,1X),A13,1X,A6,1X,A13)
99998 Format (1X,4(F10.4,1X),Es13.5,1X,I6,1X,Es13.5)
99997 Format (1X,4(F10.4,1X),Es13.5,1X,I6,1X,A17)
99996 Format (1X,4(F10.4,1X),20X,A17)
99995 Format (1X,A16,17X,Es13.5,1X,I6,1X,Es13.5)
99994 Format (1X,A16,17X,Es13.5,1X,I6,1X,A17)
99993 Format (1X,4(F10.4,1X),A13,1X,I6,1X,A13)
99992 Format (1X,4(F10.4,1X),Es13.5,1X,A6,1X,A13)

    End Program s22bffe
```

10.2 Program Data

None.

10.3 Program Results

```
 S22BFF Example Program Results

          a          b          c          x           frf    scf  2F1(a,b;c;x)
    -9.9999     1.9999    -4.9999     0.4500  -5.44477E-01     16  -3.56828E+04
   -10.0001     2.0001    -5.0001     0.4500   5.44547E-01     16   3.56875E+04

 Solution product                  -2.96494E-01     32  -1.27343E+09

 Solution ratio                    -9.99871E-01      0  -9.99871E-01
```