NAG Library Routine Document

C02AFF

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

C02AFF finds all the roots of a complex polynomial equation, using a variant of Laguerre's method.

2 Specification

```
SUBROUTINE C02AFF (A, N, SCAL, Z, W, IFAIL)
INTEGER            N, IFAIL
REAL (KIND=nag_wp) A(2,N+1), Z(2,N), W(4*(N+1))
LOGICAL            SCAL
```

3 Description

C02AFF attempts to find all the roots of the nth degree complex polynomial equation

$$P(z) = a_0 z^n + a_1 z^{n-1} + a_2 z^{n-2} + \cdots + a_{n-1} z + a_n = 0.$$

The roots are located using a modified form of Laguerre's method, originally proposed by Smith (1967).

The method of Laguerre (see Wilkinson (1965)) can be described by the iterative scheme

$$L(z_k) = z_{k+1} - z_k = \frac{-nP(z_k)}{P'(z_k) \pm \sqrt{H(z_k)}},$$

where $H(z_k) = (n-1)\Big[(n-1)(P'(z_k))^2 - nP(z_k)P''(z_k)\Big]$ and z_0 is specified.

The sign in the denominator is chosen so that the modulus of the Laguerre step at z_k, viz. $|L(z_k)|$, is as small as possible. The method can be shown to be cubically convergent for isolated roots (real or complex) and linearly convergent for multiple roots.

The routine generates a sequence of iterates $z_1, z_2, z_3, \ldots$, such that $|P(z_{k+1})| < |P(z_k)|$ and ensures that $z_{k+1} + L(z_{k+1})$ 'roughly' lies inside a circular region of radius $|F|$ about z_k known to contain a zero of $P(z)$; that is, $|L(z_{k+1})| \leq |F|$, where F denotes the Fejér bound (see Marden (1966)) at the point z_k. Following Smith (1967), F is taken to be $\min(B, 1.445nR)$, where B is an upper bound for the magnitude of the smallest zero given by

$$B = 1.0001 \times \min\Big(\sqrt{n}L(z_k), |r_1|, |a_n/a_0|^{1/n}\Big),$$

r_1 is the zero X of smaller magnitude of the quadratic equation

$$\frac{P''(z_k)}{2n(n-1)}X^2 + \frac{P'(z_k)}{n}X + \tfrac{1}{2}P(z_k) = 0$$

and the Cauchy lower bound R for the smallest zero is computed (using Newton's Method) as the positive root of the polynomial equation

$$|a_0|z^n + |a_1|z^{n-1} + |a_2|z^{n-2} + \cdots + |a_{n-1}|z - |a_n| = 0.$$

Starting from the origin, successive iterates are generated according to the rule $z_{k+1} = z_k + L(z_k)$, for $k = 1, 2, 3, \ldots$, and $L(z_k)$ is 'adjusted' so that $|P(z_{k+1})| < |P(z_k)|$ and $|L(z_{k+1})| \leq |F|$. The iterative procedure terminates if $P(z_{k+1})$ is smaller in absolute value than the bound on the rounding error in $P(z_{k+1})$ and the current iterate $z_p = z_{k+1}$ is taken to be a zero of $P(z)$. The deflated polynomial $\tilde{P}(z) = P(z)/\big(z - z_p\big)$ of degree $n-1$ is then formed, and the above procedure is repeated on the

deflated polynomial until $n < 3$, whereupon the remaining roots are obtained via the 'standard' closed formulae for a linear ($n = 1$) or quadratic ($n = 2$) equation.

Note that C02AHF, C02AMF and C02ANF can be used to obtain the roots of a quadratic, cubic ($n = 3$) and quartic ($n = 4$) polynomial, respectively.

4 References

Marden M (1966) Geometry of polynomials *Mathematical Surveys* **3** American Mathematical Society, Providence, RI

Smith B T (1967) ZERPOL: a zero finding algorithm for polynomials using Laguerre's method *Technical Report* Department of Computer Science, University of Toronto, Canada

Thompson K W (1991) Error analysis for polynomial solvers *Fortran Journal (Volume 3)* **3** 10–13

Wilkinson J H (1965) *The Algebraic Eigenvalue Problem* Oxford University Press, Oxford

5 Arguments

1: A(2, N + 1) – REAL (KIND=nag_wp) array *Input*

On entry: if A is declared with bounds (2, 0 : N), then A(1, i) and A(2, i) must contain the real and imaginary parts of a_i (i.e., the coefficient of z^{n-i}), for $i = 0, 1, \ldots, n$.

Constraint: A(1, 0) $\neq$ 0.0 or A(2, 0) $\neq$ 0.0.

2: N – INTEGER *Input*

On entry: n, the degree of the polynomial.

Constraint: N $\geq$ 1.

3: SCAL – LOGICAL *Input*

On entry: indicates whether or not the polynomial is to be scaled. See Section 9 for advice on when it may be preferable to set SCAL = .FALSE. and for a description of the scaling strategy.

Suggested value: SCAL = .TRUE..

4: Z(2, N) – REAL (KIND=nag_wp) array *Output*

On exit: the real and imaginary parts of the roots are stored in Z(1, i) and Z(2, i) respectively, for $i = 1, 2, \ldots, n$.

5: W(4 × (N + 1)) – REAL (KIND=nag_wp) array *Workspace*

6: 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 $= 1$

On entry, A$(1,0) = 0.0$ and A$(2,0) = 0.0$,
or N < 1.

IFAIL $= 2$

The iterative procedure has failed to converge. This error is very unlikely to occur. If it does, please contact NAG, as some basic assumption for the arithmetic has been violated. See also Section 9.

IFAIL $= 3$

Either overflow or underflow prevents the evaluation of $P(z)$ near some of its zeros. This error is very unlikely to occur. If it does, please contact NAG. See also Section 9.

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

All roots are evaluated as accurately as possible, but because of the inherent nature of the problem complete accuracy cannot be guaranteed. See also Section 10.

8 Parallelism and Performance

C02AFF is not threaded in any implementation.

9 Further Comments

If SCAL = .TRUE., then a scaling factor for the coefficients is chosen as a power of the base b of the machine so that the largest coefficient in magnitude approaches $thresh = b^{e_{\max}-p}$. You should note that no scaling is performed if the largest coefficient in magnitude exceeds $thresh$, even if SCAL = .TRUE.. (b, $e_{\max}$ and p are defined in Chapter X02.)

However, with SCAL = .TRUE., overflow may be encountered when the input coefficients $a_0, a_1, a_2, \ldots, a_n$ vary widely in magnitude, particularly on those machines for which $b^{(4p)}$ overflows. In such cases, SCAL should be set to .FALSE. and the coefficients scaled so that the largest coefficient in magnitude does not exceed $b^{(e_{\max}-2p)}$.

Even so, the scaling strategy used by C02AFF is sometimes insufficient to avoid overflow and/or underflow conditions. In such cases, you are recommended to scale the independent variable (z) so that

the disparity between the largest and smallest coefficient in magnitude is reduced. That is, use the routine to locate the zeros of the polynomial $dP(cz)$ for some suitable values of c and d. For example, if the original polynomial was $P(z) = 2^{-100}i + 2^{100}z^{20}$, then choosing $c = 2^{-10}$ and $d = 2^{100}$, for instance, would yield the scaled polynomial $i + z^{20}$, which is well-behaved relative to overflow and underflow and has zeros which are 2^{10} times those of $P(z)$.

If the routine fails with IFAIL $= 2$ or 3, then the real and imaginary parts of any roots obtained before the failure occurred are stored in Z in the reverse order in which they were found. Let n_R denote the number of roots found before the failure occurred. Then $\mathrm{Z}(1, n)$ and $\mathrm{Z}(2, n)$ contain the real and imaginary parts of the first root found, $\mathrm{Z}(1, n-1)$ and $\mathrm{Z}(2, n-1)$ contain the real and imaginary parts of the second root found, $\ldots, \mathrm{Z}(1, n - n_R + 1)$ and $\mathrm{Z}(2, n - n_R + 1)$ contain the real and imaginary parts of the n_Rth root found. After the failure has occurred, the remaining $2 \times (n - n_R)$ elements of Z contain a large negative number (equal to $-1/\big(\mathrm{X02AMF}() \times \sqrt{2}\big)$).

10 Example

For this routine two examples are presented. There is a single example program for C02AFF, with a main program and the code to solve the two example problems given in the subroutines EX1 and EX2.

Example 1 (EX1)

This example finds the roots of the polynomial

$$a_0 z^5 + a_1 z^4 + a_2 z^3 + a_3 z^2 + a_4 z + a_5 = 0,$$

where $a_0 = (5.0 + 6.0i)$, $a_1 = (30.0 + 20.0i)$, $a_2 = -(0.2 + 6.0i)$, $a_3 = (50.0 + 100000.0i)$, $a_4 = -(2.0 - 40.0i)$ and $a_5 = (10.0 + 1.0i)$.

Example 2 (EX2)

This example solves the same problem as subroutine EX1, but in addition attempts to estimate the accuracy of the computed roots using a perturbation analysis. Further details can be found in Thompson (1991).

10.1 Program Text

```
!   C02AFF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module c02affe_mod

!     C02AFF Example Program Module:
!            Parameters

!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
!     .. Parameters ..
      Integer, Parameter, Public       :: nin = 5, nout = 6
      Logical, Parameter, Public       :: scal = .True.
    End Module c02affe_mod
    Program c02affe

!     C02AFF Example Main Program

!     .. Use Statements ..
      Use c02affe_mod, Only: nout
!     .. Implicit None Statement ..
      Implicit None
!     .. Executable Statements ..
      Write (nout,*) 'C02AFF Example Program Results'

      Call ex1

      Call ex2
```

```
    Contains
      Subroutine ex1

!       .. Use Statements ..
        Use nag_library, Only: c02aff, nag_wp
        Use c02affe_mod, Only: nin, scal
!       .. Local Scalars ..
        Integer                       :: i, ifail, n
!       .. Local Arrays ..
        Real (Kind=nag_wp), Allocatable :: a(:,:), w(:), z(:,:)
!       .. Executable Statements ..
        Write (nout,*)
        Write (nout,*)
        Write (nout,*) 'Example 1'

!       Skip heading in data file
        Read (nin,*)
        Read (nin,*)
        Read (nin,*)

        Read (nin,*) n
        Allocate (a(2,0:n),w(4*(n+1)),z(2,n))

        Read (nin,*)(a(1,i),a(2,i),i=0,n)

        ifail = 0
        Call c02aff(a,n,scal,z,w,ifail)

        Write (nout,*)
        Write (nout,99999) 'Degree of polynomial = ', n
        Write (nout,*)
        Write (nout,*) 'Computed roots of polynomial'
        Write (nout,*)

        Do i = 1, n
          Write (nout,99998) 'z = ', z(1,i), z(2,i), '*i'
        End Do

99999   Format (1X,A,I4)
99998   Format (1X,A,1P,E12.4,Sp,E12.4,A)
      End Subroutine ex1
      Subroutine ex2

!       .. Use Statements ..
        Use nag_library, Only: a02abf, c02aff, nag_wp, x02ajf, x02alf
        Use c02affe_mod, Only: nin, scal
!       .. Local Scalars ..
        Real (Kind=nag_wp)            :: deltac, deltai, di, eps, epsbar, f,  &
                                         r1, r2, r3, rmax
        Integer                       :: i, ifail, j, jmin, n
!       .. Local Arrays ..
        Real (Kind=nag_wp), Allocatable :: a(:,:), abar(:,:), r(:), w(:),     &
                                         z(:,:), zbar(:,:)
        Integer, Allocatable          :: m(:)
!       .. Intrinsic Procedures ..
        Intrinsic                     :: abs, max, min
!       .. Executable Statements ..
        Write (nout,*)
        Write (nout,*)
        Write (nout,*) 'Example 2'

!       Skip heading in data file
        Read (nin,*)
        Read (nin,*)

        Read (nin,*) n
        Allocate (a(2,0:n),abar(2,0:n),r(n),w(4*(n+1)),z(2,n),zbar(2,n),m(n))

!       Read in the coefficients of the original polynomial.
```

```
        Read (nin,*)(a(1,i),a(2,i),i=0,n)

!       Compute the roots of the original polynomial.

        ifail = 0
        Call c02aff(a,n,scal,z,w,ifail)

!       Form the coefficients of the perturbed polynomial.

        eps = x02ajf()
        epsbar = 3.0E0_nag_wp*eps

        Do i = 0, n

          If (a(1,i)/=0.0E0_nag_wp) Then
            f = 1.0E0_nag_wp + epsbar
            epsbar = -epsbar
            abar(1,i) = f*a(1,i)

            If (a(2,i)/=0.0E0_nag_wp) Then
              abar(2,i) = f*a(2,i)
            Else
              abar(2,i) = 0.0E0_nag_wp
            End If

          Else
            abar(1,i) = 0.0E0_nag_wp

            If (a(2,i)/=0.0E0_nag_wp) Then
              f = 1.0E0_nag_wp + epsbar
              epsbar = -epsbar
              abar(2,i) = f*a(2,i)
            Else
              abar(2,i) = 0.0E0_nag_wp
            End If
          End If

        End Do

!       Compute the roots of the perturbed polynomial.

        ifail = 0
        Call c02aff(abar,n,scal,zbar,w,ifail)

!       Perform error analysis.

!       Initialize markers to 0 (unmarked).

        m(1:n) = 0

        rmax = x02alf()

!       Loop over all unperturbed roots (stored in Z).

        Do i = 1, n
          deltai = rmax
          r1 = a02abf(z(1,i),z(2,i))

!         Loop over all perturbed roots (stored in ZBAR).

          Do j = 1, n

!           Compare the current unperturbed root to all unmarked
!           perturbed roots.

            If (m(j)==0) Then
              r2 = a02abf(zbar(1,j),zbar(2,j))
              deltac = abs(r1-r2)

              If (deltac<deltai) Then
                deltai = deltac
```

```
                jmin = j
              End If

            End If

          End Do

!         Mark the selected perturbed root.

          m(jmin) = 1

!         Compute the relative error.

          If (r1/=0.0E0_nag_wp) Then
            r3 = a02abf(zbar(1,jmin),zbar(2,jmin))
            di = min(r1,r3)
            r(i) = max(deltai/max(di,deltai/rmax),eps)
          Else
            r(i) = 0.0E0_nag_wp
          End If

        End Do

        Write (nout,*)
        Write (nout,99999) 'Degree of polynomial = ', n
        Write (nout,*)
        Write (nout,*) 'Computed roots of polynomial   ', '  Error estimates'
        Write (nout,*) '                               ',                              &
          '   (machine-dependent)'
        Write (nout,*)

        Do i = 1, n
          Write (nout,99998) 'z = ', z(1,i), z(2,i), '*i', r(i)
        End Do

99999   Format (1X,A,I4)
99998   Format (1X,A,1P,E12.4,Sp,E12.4,A,5X,Ss,E9.1)
      End Subroutine ex2
    End Program c02affe
```

10.2 Program Data

```
C02AFF Example Program Data

Example 1
 5
    5.0       6.0
   30.0      20.0
   -0.2      -6.0
   50.0  100000.0
   -2.0      40.0
   10.0       1.0

Example 2
 5
    5.0       6.0
   30.0      20.0
   -0.2      -6.0
   50.0  100000.0
   -2.0      40.0
   10.0       1.0
```

10.3 Program Results

```
 C02AFF Example Program Results


 Example 1

 Degree of polynomial =    5
```

```
 Computed roots of polynomial

 z =  -2.4328E+01 -4.8555E+00*i
 z =   5.2487E+00 +2.2736E+01*i
 z =   1.4653E+01 -1.6569E+01*i
 z =  -6.9264E-03 -7.4434E-03*i
 z =   6.5264E-03 +7.4232E-03*i


 Example 2

 Degree of polynomial =    5

 Computed roots of polynomial     Error estimates
                                 (machine-dependent)

 z =  -2.4328E+01 -4.8555E+00*i       1.1E-16
 z =   5.2487E+00 +2.2736E+01*i       3.0E-16
 z =   1.4653E+01 -1.6569E+01*i       3.2E-16
 z =  -6.9264E-03 -7.4434E-03*i       1.7E-16
 z =   6.5264E-03 +7.4232E-03*i       1.1E-16
```