NAG Fortran Library Routine Document

C02AGF

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

C02AGF finds all the roots of a real polynomial equation, using a variant of Laguerre's Method.

2 Specification

```
SUBROUTINE C02AGF(A, N, SCALE, Z, W, IFAIL)
INTEGER            N, IFAIL
real               A(N+1), Z(2,N), W(2*(N+1))
LOGICAL            SCALE
```

3 Description

The routine attempts to find all the roots of the nth degree real polynomial equation

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

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 $H(z_k) = (n-1)[(n-1)(P'(z_k))^2 - nP(z_k)P''(z_k)]$ 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 Féjer 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

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

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

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

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

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

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 $P(z)$ (as is its conjugate $\bar{z}_p$ if z_p is complex). The deflated polynomial $\tilde{P}(z) = P(z)/(z - z_p)$ of degree $n-1$ if z_p is real ($\tilde{P}(z) = P(z)/((z - z_p)(z - \bar{z}_p))$ of degree $n-2$ if z_p is complex) 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 C02AJF, C02AKF and C02ALF 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 Parameters

1: A(N+1) – ***real*** array *Input*

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

Constraint: $\mathrm{A}(0) \neq 0.0$.

2: N – INTEGER *Input*

On entry: the degree of the polynomial, n.

Constraint: $\mathrm{N} \geq 1$.

3: SCALE – LOGICAL *Input*

On entry: indicates whether or not the polynomial is to be scaled. See Section 8 for advice on when it may be preferable to set SCALE = .FALSE. and for a description of the scaling strategy.

Suggested value: SCALE = .TRUE..

4: Z(2,N) – ***real*** array *Output*

On exit: the real and imaginary parts of the roots are stored in $\mathrm{Z}(1,i)$ and $\mathrm{Z}(2,i)$ respectively, for $i = 1, 2, \ldots, n$. Complex conjugate pairs of roots are stored in consecutive pairs of elements of Z; that is, $\mathrm{Z}(1,i+1) = \mathrm{Z}(1,i)$ and $\mathrm{Z}(2,i+1) = -\mathrm{Z}(2,i)$.

5: W(2∗(N+1)) – ***real*** array *Workspace*

6: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. Users who are unfamiliar with this parameter should refer to Chapter P01 for details.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

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, for users not familiar with this parameter the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

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(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 immediately, as some basic assumption for the arithmetic has been violated. See also Section 8.

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 immediately. See also Section 8.

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 9.2.

8 Further Comments

If SCALE = .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^{EMAX-P}. Users should note that no scaling is performed if the largest coefficient in magnitude exceeds THRESH, even if SCALE = .TRUE.. (B, $EMAX$ and P are defined in Chapter X02.)

However, with SCALE = .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, SCALE should be set to .FALSE. and the coefficients scaled so that the largest coefficient in magnitude does not exceed $B^{(EMAX-2P)}$.

Even so, the scaling strategy used by C02AGF is sometimes insufficient to avoid overflow and/or underflow conditions. In such cases, the user is 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 $P(z) = 2^{-100} + 2^{100}z^{20}$, then choosing $c = 2^{-10}$ and $d = 2^{100}$, for instance, would yield the scaled polynomial $1 + z^{20}$, 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 Z$(1, n)$ and Z$(2, n)$ contain the real and imaginary parts of the first root found, Z$(1, n-1)$ and Z$(2, n-1)$ contain the real and imaginary parts of the second root found,$\ldots$, Z$(1, n_R)$ and Z$(2, n_R)$ 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 $-1/(\mathrm{X02AMF}() \times \sqrt{2})$).

9 Example

For this routine two examples are presented in Section 9.1 of the documents for C02AGF and C02AGF. In the example program distributed to sites, there is a single example program for C02AGF, with a main program:

```
*     C02AGF Example Program Text
*     Mark 20 Revised. NAG Copyright 2001.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. External Subroutines ..
```

```
      EXTERNAL         EX1, EX2
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C02AGF Example Program Results'
      CALL EX1
      CALL EX2
      STOP
      END
```

The code to solve the two example problems is given in the subroutines EX1 and EX2, in Section 9.1.1 of the documents for C02AGF and C02AGF respectively.

9.1 Example 1

To find the roots of the 5th degree polynomial

$$z^5 + 2z^4 + 3z^3 + 4z^2 + 5z + 6 = 0.$$

9.1.1 Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
      SUBROUTINE EX1
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      real             ZERO
      PARAMETER        (ZERO=0.0e0)
      INTEGER          MAXDEG
      PARAMETER        (MAXDEG=100)
      LOGICAL          SCALE
      PARAMETER        (SCALE=.TRUE.)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, N, NROOT
*     .. Local Arrays ..
      real             A(0:MAXDEG), W(2*MAXDEG+2), Z(2,MAXDEG)
*     .. External Subroutines ..
      EXTERNAL         C02AGF
*     .. Intrinsic Functions ..
      INTRINSIC        ABS
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Example 1'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*)
      READ (NIN,*)
      READ (NIN,*) N
      IF (N.GT.0 .AND. N.LE.MAXDEG) THEN
         READ (NIN,*) (A(I),I=0,N)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Degree of polynomial = ', N
*
         IFAIL = 0
*
         CALL C02AGF(A,N,SCALE,Z,W,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Computed roots of polynomial'
         WRITE (NOUT,*)
         NROOT = 1
   20    IF (NROOT.LE.N) THEN
            IF (Z(2,NROOT).EQ.ZERO) THEN
               WRITE (NOUT,99998) 'z = ', Z(1,NROOT)
               NROOT = NROOT + 1
            ELSE
               WRITE (NOUT,99998) 'z = ', Z(1,NROOT), ' +/- ',
     +           ABS(Z(2,NROOT)), '*i'
```

```
               NROOT = NROOT + 2
            END IF
            GO TO 20
         END IF
      ELSE
         WRITE (NOUT,*) 'N is out of range'
      END IF
*
99999 FORMAT (1X,A,I4)
99998 FORMAT (1X,A,1P,E12.4,A,E12.4,A)
      END
```

9.1.2 Program Data

```
C02AGF Example Program Data

Example 1
 5
     1.0    2.0    3.0    4.0    5.0    6.0
```

9.1.3 Program Results

```
 C02AGF Example Program Results


 Example 1

 Degree of polynomial =    5

 Computed roots of polynomial

 z =  -1.4918E+00
 z =   5.5169E-01 +/-   1.2533E+00*i
 z =  -8.0579E-01 +/-   1.2229E+00*i
```

9.2 Example 2

This example solves the same problem as Section 9.1, but in addition attempts to estimate the accuracy of the computed roots using a perturbation analysis. Further details can be found in Thompson (1991).

9.2.1 Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
      SUBROUTINE EX2
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      real             ZERO, ONE, THREE
      PARAMETER        (ZERO=0.0e0,ONE=1.0e0,THREE=3.0e0)
      INTEGER          MAXDEG
      PARAMETER        (MAXDEG=100)
      LOGICAL          SCALE
      PARAMETER        (SCALE=.TRUE.)
*     .. Local Scalars ..
      real             DELTAC, DELTAI, DI, EPS, EPSBAR, F, R1, R2, R3,
     +                 RMAX
      INTEGER          I, IFAIL, J, JMIN, N
*     .. Local Arrays ..
      real             A(0:MAXDEG), ABAR(0:MAXDEG), R(MAXDEG),
     +                 W(2*MAXDEG+2), Z(2,MAXDEG), ZBAR(2,MAXDEG)
      INTEGER          M(MAXDEG)
*     .. External Functions ..
      real             A02ABF, X02AJF, X02ALF
      EXTERNAL         A02ABF, X02AJF, X02ALF
*     .. External Subroutines ..
      EXTERNAL         C02AGF
```

```
*     .. Intrinsic Functions ..
      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
      IF (N.GT.0 .AND. N.LE.MAXDEG) THEN
*
*        Read in the coefficients of the original polynomial.
*
         READ (NIN,*) (A(I),I=0,N)
*
*        Compute the roots of the original polynomial.
*
         IFAIL = 0
*
         CALL C02AGF(A,N,SCALE,Z,W,IFAIL)
*
*        Form the coefficients of the perturbed polynomial.
*
         EPS = X02AJF()
         EPSBAR = THREE*EPS
         DO 20 I = 0, N
            IF (A(I).NE.ZERO) THEN
               F = ONE + EPSBAR
               EPSBAR = -EPSBAR
               ABAR(I) = F*A(I)
            ELSE
               ABAR(I) = ZERO
            END IF
   20    CONTINUE
*
*        Compute the roots of the perturbed polynomial.
*
         IFAIL = 0
*
         CALL C02AGF(ABAR,N,SCALE,ZBAR,W,IFAIL)
*
*        Perform error analysis.
*
         DO 40 I = 1, N
*           Initialize markers to 0 (unmarked).
            M(I) = 0
   40    CONTINUE
         RMAX = X02ALF()
*        Loop over all unperturbed roots (stored in Z).
         DO 80 I = 1, N
            DELTAI = RMAX
            R1 = A02ABF(Z(1,I),Z(2,I))
*           Loop over all perturbed roots (stored in ZBAR).
            DO 60 J = 1, N
*              Compare the current unperturbed root to all unmarked
*              perturbed roots.
               IF (M(J).EQ.0) THEN
                  R2 = A02ABF(ZBAR(1,J),ZBAR(2,J))
                  DELTAC = ABS(R1-R2)
                  IF (DELTAC.LT.DELTAI) THEN
                     DELTAI = DELTAC
                     JMIN = J
                  END IF
               END IF
   60       CONTINUE
*           Mark the selected perturbed root.
            M(JMIN) = 1
*           Compute the relative error.
            IF (R1.NE.ZERO) 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) = ZERO
            END IF
   80    CONTINUE
*
         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 100 I = 1, N
            WRITE (NOUT,99998) 'z = ', Z(1,I), Z(2,I), '*i', R(I)
  100    CONTINUE
      ELSE
         WRITE (NOUT,*) 'N is out of range'
      END IF
*
99999 FORMAT (1X,A,I4)
99998 FORMAT (1X,A,1P,E12.4,SP,E12.4,A,5X,SS,E9.1)
      END
```

9.2.2 Program Data

```
C02AGF Example Program Data

Example 2
 5
     1.0    2.0    3.0    4.0    5.0    6.0
```

9.2.3 Program Results

```
 C02AGF Example Program Results


 Example 2

 Degree of polynomial =    5

 Computed roots of polynomial     Error estimates
                                (machine-dependent)

 z =  -1.4918E+00 +0.0000E+00*i       1.2E-15
 z =   5.5169E-01 +1.2533E+00*i       1.1E-16
 z =   5.5169E-01 -1.2533E+00*i       1.1E-16
 z =  -8.0579E-01 +1.2229E+00*i       1.1E-16
 z =  -8.0579E-01 -1.2229E+00*i       1.1E-16
```