NAG Toolbox for MATLAB

c02ag

1 Purpose

c02ag finds all the roots of a real polynomial equation, using a variant of Laguerre's Method.

2 Syntax

```
[z, ifail] = c02ag(a, n, 'scal', scal)
```

3 Description

c02ag attempts to find all the roots of the nth degree real polynomial equation

$$P(z) = a_0z^n + a_1z^{n-1} + a_2z^{n-2} + \cdots + a_{n-1}z + a_n = 0.$$

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 $H(z_k) = (n-1)\left[(n-1)\left(P'(z_k)\right)^2 - nP(z_k)P''(z_k)\right]$ 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 function generates a sequence of iterates $z_1, z_2, z_3, \cdots$, 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

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

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 $P(z)$ (as is its conjugate $\bar{z}_p$ if z_p is complex). The deflated polynomial $\tilde{P}(z) = P(z)/\left(z - z_p\right)$ of degree $n-1$ if z_p is real ($\tilde{P}(z) = P(z)/\left(\left(z - z_p\right)\left(z - \bar{z}_p\right)\right)$ 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 c02aj, c02ak and c02al 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

5.1 Compulsory Input Parameters

1: **a(n + 1) – double array**

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: **a**$(0) \neq 0.0$.

2: **n – int32 scalar**

n, the degree of the polynomial.

Constraint: **n** ≥ 1.

5.2 Optional Input Parameters

1: **scal – logical scalar**

Indicates whether or not the polynomial is to be scaled. See Section 8 for advice on when it may be preferable to set **scal** = **false** and for a description of the scaling strategy.

Suggested value: **scal** = **true**.

Default: **true**

5.3 Input Parameters Omitted from the MATLAB Interface

w

5.4 Output Parameters

1: **z(2,n) – double array**

The real and imaginary parts of the roots are stored in **z**$(1, i)$ and **z**$(2, i)$ respectively, for $i = 1, 2, \ldots, n$. Complex conjugate pairs of roots are stored in consecutive pairs of elements of **z**; that is, **z**$(1, i + 1) =$ **z**$(1, i)$ and **z**$(2, i + 1) = -$**z**$(2, i)$.

2: **ifail – int32 scalar**

0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

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.

8 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 c02ag 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 function 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 function 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 $\mathbf{z}(1, n)$ and $\mathbf{z}(2, n)$ contain the real and imaginary parts of the first root found, $\mathbf{z}(1, n-1)$ and $\mathbf{z}(2, n-1)$ contain the real and imaginary parts of the second root found, $\ldots, \mathbf{z}(1, n - n_R + 1)$ and $\mathbf{z}(2, n - n_R + 1)$ 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 $-1/(\text{x02am}() \times \sqrt{2})$).

9 Example

```
a = [1;
     2;
     3;
     4;
     5;
     6];
n = int32(5);
[z, ifail] = c02ag(a, n)
```

```
z =
   -1.4918    0.5517    0.5517   -0.8058   -0.8058
         0    1.2533   -1.2533    1.2229   -1.2229
ifail =
           0
```