NAG Library Function Document

nag_zeros_real_poly (c02agc)

1 Purpose

nag_zeros_real_poly (c02agc) finds all the roots of a real polynomial equation, using a variant of Laguerre's method.

2 Specification

```
#include <nag.h>
#include <nagc02.h>
```

```
void nag_zeros_real_poly (Integer n, const double a[], Nag_Boolean scale,
     Complex z[], NagError *fail)
```

3 Description

nag_zeros_real_poly (c02agc) attempts to find all the roots of the nth degree real polynomial equation

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

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 function 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

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

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 $\tilde{P}(z) = P(z)/(z - z_p)$ of degree $n-1$ if z_p is real $(\tilde{P}(z) = P(z)/\big((z - z_p)(z - \bar{z}_p)\big)$ 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.

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

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

5 Arguments

1: **n** – Integer *Input*

On entry: n, the degree of the polynomial.

Constraint: $\mathbf{n} \geq 1$.

2: $\mathbf{a}[\mathbf{n}+\mathbf{1}]$ – const double *Input*

On entry: $\mathbf{a}[i]$ must contain a_i (i.e., the coefficient of z^{n-i}), for $i = 0, 1, \ldots, n$.

Constraint: $\mathbf{a}[0] \neq 0.0$.

3: **scale** – Nag_Boolean *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 **scale** = Nag_FALSE and for a description of the scaling strategy.

Suggested value: **scale** = Nag_TRUE.

4: $\mathbf{z}[\mathbf{n}]$ – Complex *Output*

On exit: the real and imaginary parts of the roots are stored in $\mathbf{z}[i].re$ and $\mathbf{z}[i].im$ respectively, for $i = 0, 1, \ldots, n-1$. Complex conjugate pairs of roots are stored in consecutive pairs of **z**; that is, $\mathbf{z}[i+1].re = \mathbf{z}[i].re$ and $\mathbf{z}[i+1].im = -\mathbf{z}[i].im$

5: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument $\langle$*value*$\rangle$ had an illegal value.

NE_INT_ARG_LT

On entry, **n** = $\langle$*value*$\rangle$.
Constraint: $\mathbf{n} \geq 1$.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

NE_POLY_NOT_CONV

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.

NE_POLY_OVFLOW

The function cannot evaluate $P(z)$ near some of its zeros without overflow. Please contact NAG immediately.

NE_POLY_UNFLOW

The function cannot evaluate $P(z)$ near some of its zeros without underflow. Please contact NAG immediately.

NE_REAL_ARG_EQ

On entry, $\mathbf{a}[0] = \langle\mathit{value}\rangle$.
Constraint: $\mathbf{a}[0] \neq 0.0$.

7 Accuracy

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

8 Parallelism and Performance

Not applicable.

9 Further Comments

If **scale** $=$ Nag_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 **scale** $=$ Nag_TRUE. (b, $e_{\max}$ and p are defined in Chapter x02.)

However, with **scale** $=$ Nag_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 Nag_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 in nag_zeros_real_poly (c02agc) 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 $d \times P(cz)$ 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 NE_POLY_NOT_CONV, NE_POLY_UNFLOW or NE_POLY_OVFLOW, 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. More precisely, $\mathbf{z}[\mathbf{n} - 1].re$ and $\mathbf{z}[\mathbf{n} - 1].im$ contain the real and imaginary parts of the 1st root found, $\mathbf{z}[\mathbf{n} - 2].re$ and $\mathbf{z}[\mathbf{n} - 2].im$ contain the real and imaginary parts of the 2nd root found, and so on. The real and imaginary parts of any roots not found will be set to a large negative number, specifically $-1.0/\left(\sqrt{2.0} \times \text{nag\_real\_safe\_small\_number}\right)$.

10 Example

To find the roots of the 5th degree polynomial $z^5 + 2z^4 + 3z^3 + 4z^2 + 5z + 6 = 0$.

10.1 Program Text

```
/* nag_zeros_real_poly (c02agc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 *
 * Mark 8 revised, 2004.
 */

#include <nag.h>
#include <stdio.h>
#include <math.h>
#include <nag_stdlib.h>
#include <nagc02.h>

int main(void)
{

  Nag_Boolean scale;
  Complex     *z = 0;
  Integer     exit_status = 0, i, n, nroot;
  NagError    fail;
  double      *a = 0;

  INIT_FAIL(fail);

  printf("nag_zeros_real_poly (c02agc) Example Program Results\n");
  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"", &n);
#else
  scanf("%"NAG_IFMT"", &n);
#endif
  if (n > 0)
    {
      scale = Nag_TRUE;
      if (!(a = NAG_ALLOC(n+1, double)) ||
          !(z = NAG_ALLOC(n, Complex)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
    }
  else
    {
      printf("Invalid n.\n");
      exit_status = 1;
      return exit_status;
    }

  for (i = 0; i <= n; i++)
#ifdef _WIN32
    scanf_s("%lf", &a[i]);
#else
    scanf("%lf", &a[i]);
#endif
  printf("\nDegree of polynomial = %4"NAG_IFMT"\n\n", n);

  /* nag_zeros_real_poly (c02agc).
   * Zeros of a polynomial with real coefficients
   */
  nag_zeros_real_poly(n, a, scale, z, &fail);
  if (fail.code != NE_NOERROR)
```

```
    {
      printf("Error from nag_zeros_real_poly (c02agc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  printf("Roots of polynomial\n\n");
  nroot = 1;
  while (nroot <= n)
    {
      if (z[nroot-1].im == 0.0)
        {
          printf("z = %13.4e\n", z[nroot-1].re);
          nroot += 1;
        }
      else
        {
          printf("z = %13.4e   +/- %14.4e\n", z[nroot-1].re,
                  fabs(z[nroot-1].im));
          nroot += 2;
        }
    }
 END:
  NAG_FREE(a);
  NAG_FREE(z);
  return exit_status;
}
```

10.2 Program Data

```
nag_zeros_real_poly (c02agc) Example Program Data
 5
     1.0    2.0    3.0    4.0    5.0    6.0
```

10.3 Program Results

```
nag_zeros_real_poly (c02agc) Example Program Results

Degree of polynomial =    5

Roots of polynomial

z =  -1.4918e+00
z =   5.5169e-01   +/-     1.2533e+00
z =  -8.0579e-01   +/-     1.2229e+00
```