7.1 Roots of a Polynomial

A. Purpose

Given the coefficients a_i of a polynomial of degree n = NDEG > 0,

$$a_1 z^n + a_2 z^{n-1} + ... + a_n z + a_{n+1}$$

with $a_1 \neq 0$, this subroutine computes the NDEG roots of the polynomial. Since the roots may be complex, they will always be returned as complex numbers, even when they are real.

Subroutines are provided for four cases:

SPOLZ	for single precision real coefficients.
DPOLZ	for double precision real coefficients.
CPOLZ	for single precision complex coefficients.
ZPOLZ	for double precision complex coefficients.

See Section E for remarks on the method of storing double precision complex numbers.

B. Usage

B.1 Program Prototype, Single Precision Real Coefficients

INTEGER NDEG, IERR

REAL A($\geq$ NDEG + 1), **H**($\geq$ NDEG**2)

COMPLEX Z($\geq$ NDEG)

Assign values to A(), and NDEG.

CALL SPOLZ(A, NDEG, Z, H, IERR)

Computed quantities are returned in Z() and IERR.

B.2 Program Prototype, Double Precision Real Coefficients

INTEGER NDEG, IERR

DOUBLE PRECISION A($\geq$ NDEG + 1), **H**($\geq$NDEG**2)

DOUBLE PRECISION Z(2, $\geq$ NDEG) or

COMPLEX*16 Z($\geq$ NDEG)

Assign values to A(), and NDEG.

CALL DPOLZ(A, NDEG, Z, H, IERR)

Computed quantities are returned in Z() and IERR.

B.3 Program Prototype, Single Precision Complex Coefficients

INTEGER NDEG, IERR

COMPLEX A($\geq$NDEG + 1)

REAL H($\geq$ 2 * NDEG**2)

COMPLEX Z($\geq$ NDEG)

Assign values to A(), and NDEG.

CALL CPOLZ(A, NDEG, Z, H, IERR)

Computed quantities are returned in Z() and IERR.

B.4 Program Prototype, Double Precision Complex Coefficients

INTEGER NDEG, IERR

DOUBLE PRECISION A(2, $\geq$ NDEG + 1) or

COMPLEX*16 A($\geq$ NDEG + 1)

DOUBLE PRECISION H ($\geq$2 * NDEG**2)

DOUBLE PRECISION Z(2, $\geq$ NDEG) or

COMPLEX*16 Z($\geq$ NDEG)

Assign values to A(), and NDEG.

CALL ZPOLZ(A, NDEG, Z, H, IERR)

Computed quantities are returned in Z() and IERR.

B.5 Argument Definitions

A() [in] Contains the coefficients of a polynomial, high order coefficient first, with A(1) $\neq$ 0. The contents of this array will not be modified by the subroutine.

In the case of ZPOLZ, if the declaration DOUBLE PRECISION A(2, NDEG+1) is used, the user must store the real part of a_i in A(1, i) and the imaginary part in A(2, i).

NDEG [in] Degree of the polynomial.

H() [scratch] Work space for the subroutine.

Z() [out] Contains the polynomial roots stored as complex numbers. In the cases of DPOLZ and ZPOLZ, if the declaration DOUBLE PRECISION Z(2, NDEG) is used, the real part of z_i will be returned in Z$(1, i)$ and the imaginary part in Z$(2, i)$.

IERR [out] Error flag. Set by the subroutine to 0 on normal termination. Set to -1 if A(1) = 0. Set to -2 if NDEG < 1. Set to J > 0 if the iteration count limit has been exceeded and roots 1 through J have not been determined.

C. Examples and Remarks

The program, DRSPOLZ, uses SPOLZ to compute the roots of the two polynomials

$$x^3 - 4x^2 + x - 4, \quad \text{and}$$
$$x^5 - 15x^4 + 85x^3 - 225x^2 + 274x - 120$$

The output is shown in ODSPOLZ.

The Fortran 77 standard does not support a double precision complex data type, although such a type is supported in many compilers using the declaration, COMPLEX*16. The subroutines described here conform to the standard and thus do not use COMPLEX*16. For cases in which double precision complex quantities are communicated using the arrays A() or Z(), these subroutines store a complex number as an adjacent pair of double precision numbers, representing respectively the real and imaginary parts of the complex number. This is compatible with the Fortran 90 storage convention for double precision complex. If the user has the COMPLEX*16 declaration available and wishes to use it, it will generally be compatible with this storage convention.

D. Functional Description

Method

The degree, NDEG, and coefficients, a_1, ..., a_{NDEG+1}, are given. An error condition is reported if NDEG < 1 or if $a_1 = 0$.

Let $k+1$ be the index of the last nonzero coefficient. If $k <$ NDEG, the polynomial has a root at zero repeated NDEG $- k$ times and these roots are recorded. The remaining roots will be roots of the polynomial of degree k whose coefficients are the first $k+1$ given coefficients. If $k = 1$, the root $-a_2/a_1$ is recorded.

If $k > 1$, the subroutine forms the $k \times k$ companion matrix H for this k^{th} degree polynomial. The matrix H is zero except for the first row,

$$h_{1,j} = -a_{j+1}/a_1, \quad j = 1, ..., k$$

and the elements just below the diagonal,

$$h_{i,i-1} = 1, \quad i = 2, ..., k$$

A scaling algorithm is applied to H to balance the sizes of the nonzero elements. Testing showed that accuracy could be very unsatisfactory if no scaling were done. The scaling method used is a modification of the subroutine BALANC from the EISPACK collection of matrix eigensystem subroutines [1], [2]. The modification avoids treating the elements that are known to be zero due to the form of the companion matrix. The search for useful row or column permutations done in BALANC is also deleted since, with a_{k+1} known to be nonzero, none of these permutations are possible.

The eigenvalues of the balanced H matrix are then computed using appropriate code for the QR algorithm from EISPACK. For SPOLZ and DPOLZ the code for the EISPACK subroutine, HQR, has been incorporated in-line. For CPOLZ or ZPOLZ, a call is made to SCOMQR or DCOMQR, respectively. These latter two subroutines are, respectively, single precision and double precision versions of the EISPACK subroutine, COMQR.

Reference [3] reports that compared with other widely used methods, the conversion of the root finding problem to an eigenvalue problem is superior in reliability, generally superior in accuracy, and comparable in speed.

Accuracy tests

A root of a polynomial is called ill-conditioned if a small relative change in the coefficients makes a significantly larger relative change in the root. Roots that are bunched closely together relative to their magnitude, or relative to their distance from other roots, tend to be ill-conditioned. An extreme case of ill-conditioning occurs with multiple roots. If the coefficients of a polynomial are only known to some relative precision, say p, then a double root will typically have a relative uncertainty of about $p^{1/2}$, and a triple root a relative uncertainty of about $p^{1/3}$.

The accuracy of a computed root is limited by the inherent conditioning of the root. Rather than directly testing the accuracy of the computed roots, we have chosen to test the accuracy with which polynomial coefficients could be reconstituted from the computed roots.

To test these subroutines we selected eight sets of roots to test SPOLZ and DPOLZ and a different group of eight sets of roots to test CPOLZ and ZPOLZ. These sets included double and triple roots, roots that formed a small cluster relative to other roots, roots that differed by a factor of 10^8, and both real and complex roots. The number of roots in each set varied from 1 to 7.

From each set of roots we created 8 test sets by multiplying all roots in the set by one of the factors, 10^{-3}, 10^{-2}, ..., 10^3, or 10^4. This produced a total of 64 test sets for each subroutine.

To execute the test for one subroutine, say xPOLZ, and one set of roots, say r_1, ..., r_n, we did the following:

1. Compute, in double precision, the coefficients, say a_i, of the monic polynomial having the roots, r_i. Subroutine ZCOEF of Chapter 15.3 is used for this.
2. Use subroutine xPOLZ to compute the roots, say s_i, of this polynomial.

3. Compute, in double precision, the coefficients, say b_i, of the monic polynomial having the roots, s_i.
4. Compute δ = the maximum of the relative differences between corresponding coefficients, a_i and b_i.
5. Compute $\varepsilon = \delta / \rho$, where ρ is the relative precision of the arithmetic being used by xPOLZ. The value for ρ was obtained as R1MACH(4) for single precision and D1MACH(4) for double precision. (See Chapter 19.1.)

Ideally, Step 3 should be done using significantly higher precision than is used by the subroutine xPOLZ being tested, so the error measures, δ and ε, will be attributable only to xPOLZ with no inflation from Step 3. Since we used double precision in Step 3 for all cases, we conclude that the reported values of ε are solely attributable to SPOLZ and CPOLZ in the tests of those subroutines, while the values of ε reported for DPOLZ and ZPOLZ are somewhat inflated due to errors from Step 3.

These tests were run on an IBM PC/AT equipped with the Intel 80287 math coprocessor. The precision was $\rho \approx 1.2 \times 10^{-7}$ for single precision and 2.2×10^{-16} for double precision. The results may be summarized as follows:

Subroutine	Max. ε over 64 test cases	Number of Cases with $\epsilon > 10$
SPOLZ	35	19
DPOLZ	57	21
CPOLZ	8	0
ZPOLZ	43	8

References

1. B. T. Smith, J. M. Boyle, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler, **Matrix Eigensystem Routines — EISPACK Guide**, *Lecture Notes in Computer Science 6*, Springer Verlag, Berlin (1974) 387 pages.
2. B. S. Garbow, J. M. Boyle, J. J. Dongarra, and C. B. Moler, **Matrix Eigensystem Routines — EISPACK Guide Extension**, *Lecture Notes in Computer Science 51*, Springer Verlag, Berlin (1977) 343 pages.
3. S. Goedecker, *Remark on algorithms to find roots of polynomials*, **SIAM J. on Scientific Computing 15**, 5 (Sept. 1994) 1058–1063.

E. Error Procedures and Restrictions

The error indicator, IERR, will be set as follows:

= 0 when no errors are detected.

= −1 if A(1) is zero, or A(1, 1) and A(2, 1) are both zero.

= −2 if NDEG < 1.

= J > 0 if the iteration count limit has been exceeded and roots 1 through J have not been determined. The matrix eigenvalue codes allow a maximum of 30 iterations for each root. This limit is ample.

When IERR is set nonzero, the subroutine will also issue an error message using ERMSG, of Chapter 19.2, with an error level of 0.

F. Supporting Information

Entry	Required Files
CPOLZ	AMACH, CPOLZ, ERFIN, ERMSG, SCOMQR
DPOLZ	AMACH, DPOLZ, ERFIN, ERMSG
SPOLZ	AMACH, ERFIN, ERMSG, SPOLZ
ZPOLZ	AMACH, DCOMQR, DZABS, ERFIN, ERMSG, ZPOLZ, ZQUO, ZSQRT

Designed by C. L. Lawson, JPL, May 1986. Programmed by C. L. Lawson and S. Y. Chiu, JPL, May 1986, Feb. 1987.

DRSPOLZ

```
c      Program DRSPOLZ
c>> 2009-10-28 DRZPOLZ Krogh Mods to get comples used in single prec.
c>> 1996-07-09 DRZPOLZ Krogh Set for deriving single precision C vers.
c>> 1994-08-09 DRSPOLZ WVS Remove '0' in format
c>> 1992-03-06 DRSPOLZ CLL
c>> 1987-12-09 DRSPOLZ  Lawson  Initial Code.
c Conversion should only be done from "D" to "S" for processing to C.
c--S replaces "?": DR?POLZ, ?POLZ
c      Demonstration driver for SPOLZ.
c     ------------------------------------------------------------------
      real             A1(4), A2(6), H(25)

c++ CODE for .D. | .C. is inactive
C       real              Z1(2,3), Z2(2,5)
c++ CODE for .S. & ~.C. is active
      complex          Z1(3), Z2(5)
c++ END
      integer N1, N2, IERR
c++ CODE for ~.C. is active
      integer I, k
c++ CODE for .C. is inactive
c%%    long int i, k;
c++ END
c
      data A1 / 1.E0, -4.E0, 1.E0, -4.E0 /
      data A2 / 1.E0, -15.E0, 85.E0, -225.E0, 274.E0, -120.E0 /
c     ------------------------------------------------------------------
      N1 = 3
      N2 = 5
c
c++ CODE for ~.C. is active
  100 format(' ','Degree =',I2/' ', 'Coefficients =',(T20,4(F10.4,1X)))
  200 format(' ','Roots ='/(2(1X,'(',1X,F8.5,',',1X,F8.5,2X,')':2X)))
  300 format(//' ')
      print 100, N1, (A1(I),I=1,4)
c++ CODE for .C. is inactive
c%%   printf( " Degree =%2ld\n Coefficients =      ", n1 );
c%%   for (i = 0; i < 4; i++) printf( "%10.4f", a1[i] );
c++ End
      call SPOLZ(A1,N1,Z1,H,IERR)
c++ CODE for .D. & ~.C. is inactive
C       print 200, ((Z1(K,I),K=1,2),I=1,3)
c++ CODE for .S. & ~.C. is active
      print 200, (Z1(I),I=1,3)
c++ CODE for ~.C. is active
      print 300
      print 100, N2, (A2(I),I=1,6)
c++ CODE for .C. is inactive
c%%   printf( "\n Roots =\n" );
c%%   for (i = 0; i < 3; i+=2) {
c%%      printf( " ( %8.5f, %8.5f   )", z1[i][0],z1[i][1] );
c%%      if (i<2) printf( " ( %8.5f, %8.5f   )", z1[i+1][0],z1[i+1][1] );
c%%      printf( "\n" );}
c%%   printf( "\n\n Degree =%2ld\n Coefficients =      ", n2 );
c%%   for (i = 0; i < 6; i+=4) {
c%%      for (k = i ; k < (i < 2 ? i + 4 : 6); k++)
```

```
c%%           printf( "%10.4f", a2[k] );
c%%           if (i < 4) printf( "\n                          " );}
c++ End
      call SPOLZ(A2,N2,Z2,H,IERR)
c++ CODE for .D. & ~.C. is inactive
C       print 200,((Z2(K,I),K=1,2),I=1,5)
c++ CODE for .S. & ~.C. is active
      print 200,(Z2(I),I=1,5)
c++ CODE for .C. is inactive
c%%    printf( "\n Roots =\n" );
c%%    for (i = 0; i < 5; i+=2) {
c%%         printf( " ( %8.5f, %8.5f  )", z2[i][0],z2[i][1] );
c%%         if (i<4) printf(" ( %8.5f, %8.5f  )", z2[i+1][0],z2[i+1][1]);
c%%         printf( "\n" );}
c%%    printf( "\n" );
c++ END
      end
```

ODSPOLZ

```
Degree = 3
Coefficients =          1.0000     -4.0000      1.0000     -4.0000
Roots =
(   4.00000,   0.00000  )    (   0.00000,   1.00000  )
(   0.00000,  -1.00000  )



Degree = 5
Coefficients =          1.0000    -15.0000     85.0000   -225.0000
                      274.0000   -120.0000
Roots =
(   4.99991,   0.00000  )    (   4.00018,   0.00000  )
(   2.99988,   0.00000  )    (   2.00003,   0.00000  )
(   1.00000,   0.00000  )
```