

MULTROOT – A Matlab package computing polynomial roots and multiplicities

Zhonggang Zeng*

Abstract

MULTROOT is a collection of Matlab modules for accurate computation of polynomial roots, especially roots with non-trivial multiplicities. As a blackbox-type software, MULTROOT requires the polynomial coefficients as the only input, and outputs the computed roots, multiplicities, backward error, estimated forward error, as well as the pejorative condition number. The most significant features of MULTROOT are the multiplicity identification capability and the remarkable accuracy on multiple roots without using the multiprecision arithmetic, even if the polynomial coefficients are inexact. A comprehensive test suit of polynomials that are collected from the literature is included for numerical experiments and performance comparison.

Categories and Subject Descriptors: G.1.5 [Mathematics of Computing]: Roots of Nonlinear Equations – Iterative methods; methods for polynomials; G.4 [Mathematics of Computing]: Mathematical Software – Algorithm design and analysis; Certification and testing.

Additional key Words and Phrases: Polynomial, root-finding, multiple roots, multiplicity identification

1 Overview

Polynomial root-finding is a fundamental mathematical problem with a long history. It remains a challenge today. One of the most difficult issues in root-finding has been computing multiple roots and identifying the multiplicities.

Several root-finders have been established and implemented as standard softwares, such as Laguerre’s method implemented by Numerical Algorithms Group (FORTRAN code C02AFF) and Numerical Recipes (FORTRAN code ZROOTS), Jenkins-Traub method implemented by IMSL (FORTRAN code DZPOCC) and Mathematica (NSOLVE), as well as QR algorithm on the companion matrix implemented in Matlab (ROOTS). There is, however, a common limitation for all those methods. Namely they are subject to a barrier of *attainable accuracy* [Igarashi et al.1995] in computing multiple roots. Using the standard double precision of 16 decimal digits, a root of multiplicity 5 can only be calculated to an accuracy of 3 correct digits, assuming that the polynomial coefficients have at least 15 digits of accuracy. If the

*Department of Mathematics, Northeastern Illinois University, Chicago, IL 60625, email: zzeng@neiu.edu.

coefficients are inexact, those standard methods can not calculate multiple roots accurately even if multiprecision is used.

In this paper, we introduce a root-finding package MULTROOT that accurately calculates polynomial roots, especially those involve non-trivial multiplicities and inexact coefficients, as well as the corresponding multiplicities, using the standard machine precision only.

The package is an implementation of two novel algorithms GCDROOT and PEJROOT and their combination MULTROOT [Zeng 2003]. For a given polynomial

$$p(x) = p_0x^n + p_1x^{n-1} + \cdots + p_{n-1}x + p_n, \quad (1)$$

GCDROOT calculates its multiplicity structure $\ell = [\ell_1, \cdots, \ell_m]$ that reveals the factorization

$$p(x) = p_0(x - x_1)^{\ell_1} \cdots (x - x_m)^{\ell_m} \quad (2)$$

with distinct unknown roots x_1, \cdots, x_m . While identifying the multiplicity structure, GCDROOT also computes an initial approximation to the distinct roots x_j 's. Given the multiplicity structure and the initial root approximation, PEJROOT refines the roots to an optimal accuracy permissible by the pejorative condition number.

The analysis and description of the algorithms is reported in a paper [Zeng 2003] written by the author.

2 The call sequence and the interpretation of output

MULTROOT is easy to use. Using the Matlab representation of the polynomial (1) as a coefficient vector $\mathbf{p} = (p_0, p_1, \cdots, p_n)$, execution of MULTROOT a simple Matlab call

```
>> z = multroot(p);
```

For example, to calculate the roots of

$$p(x) = x^{10} - 17x^9 + 127x^8 - 549x^7 + 1521x^6 - 2823x^5 + 3557x^4 - 3007x^3 + 1634x^2 - 516x + 72,$$

only two Matlab commands are needed. One line defines the coefficient vector, and the other line calls MULTROOT:

```
>> p = [1 -17 127 -549 1521 -2823 3557 -3007 1634 -516 72];
>> z = multroot(p);
```

```
THE PEJORATIVE CONDITION NUMBER:          20.1463
THE BACKWARD ERROR:                      3.22e-016
THE ESTIMATED FORWARD ERROR:             1.30e-014
```

```
      computed roots      multiplicities
```

2.999999999999997	2
2.000000000000001	3
1.000000000000000	5

The result shows an accurate factorization of the original polynomial

$$p(x) = (x - 1)^5(x - 2)^3(x - 3)^2. \quad (3)$$

In addition to this screen print-out, the output **z** of MULTROOT is the two-column matrix

$$\mathbf{z} = \begin{pmatrix} 2.999999999999997 & 2 \\ 2.000000000000001 & 3 \\ 1.000000000000000 & 5 \end{pmatrix} \quad (4)$$

The first column consists of those distinct computed roots, while the entries in second column are the corresponding multiplicities. In contrast, the standard Matlab built-in root-finder **ROOTS** not only fails to recognize the presence of multiple roots, but also produces results that are not nearly as accurate. It outputs all “simple” roots with an “attainable accuracy” of 2–6 correct digits:

```

3.00000454105053
2.99999545886518

2.00045075042739
1.99977462490777 + 0.00039040060697i
1.99977462490777 - 0.00039040060697i

1.00241744569987 + 0.00177521346397i
1.00241744569987 - 0.00177521346397i
0.99905908264182 + 0.00281750296534i
0.99905908264182 - 0.00281750296534i
0.99704694315800

```

When multiplicities increases, the “attainable accuracy” of standard methods deteriorates even further. This phenomenon can be demonstrated by applying root-finders on the powers of $p(x)$, such as

$$q(x) = [p(x)]^6 = (x - 1)^{30}(x - 2)^{18}(x - 3)^{12}$$

Our MULTROOT maintains its high accuracy:

THE PEJORATIVE CONDITION NUMBER:	2.06248
THE BACKWARD ERROR:	1.55e-015
THE ESTIMATED FORWARD ERROR:	6.41e-015

computed roots	multiplicities
3.000000000000001	12
1.999999999999999	18
1.000000000000000	30

while the Matlab built-in root-finder `ROOTS` produces scattered results with huge forward error, as shown in Fig. 1. All standard softwares output similar inaccurate results, due to the “attainable accuracy” barrier they are subject to.

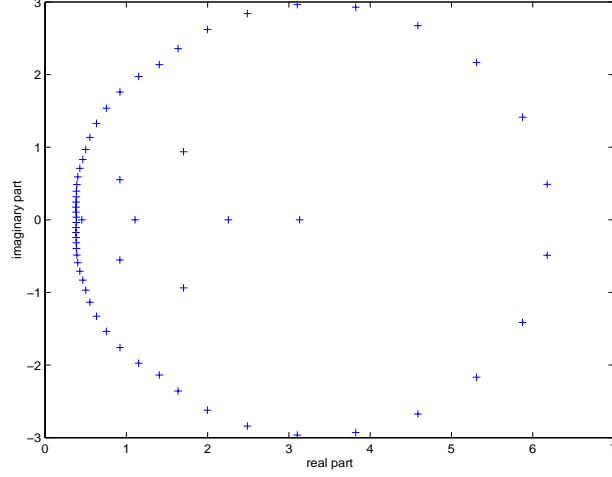


Figure 1: Roots of $q(x) = (x-1)^{30}(x-2)^{18}(x-3)^{12}$ computed by Matlab built-in root-finder `ROOTS`

By design, `MULTROOT` calculates a complete factorization (2) of the polynomial $p(x)$ in (1). Let z_1, \dots, z_m be computed roots with multiplicities ℓ_1, \dots, ℓ_m respectively. The results of `MULTROOT` are *exact* roots/multiplicities of *another* polynomial $\tilde{p}(x)$ near $p(x)$

$$\begin{aligned} \tilde{p}(x) &= \tilde{p}_0 x^n + \tilde{p}_1 x^{n-1} + \dots + \tilde{p}_n = \tilde{p}_0 (x - z_1)^{\ell_1} \dots (x - z_m)^{\ell_m} \\ &\approx p_0 x^n + p_1 x^{n-1} + \dots + p_n = p_0 (x - x_1)^{\ell_1} \dots (x - x_m)^{\ell_m} = p(x) \end{aligned}$$

with coefficient vectors

$$\tilde{\mathbf{p}} = (\tilde{p}_0, \dots, \tilde{p}_n) \approx (p_0, \dots, p_n) = \mathbf{p}.$$

The backward error is a weighted distance between \mathbf{p} and $\tilde{\mathbf{p}}$

$$\|\mathbf{p} - \tilde{\mathbf{p}}\|_W = \sqrt{\sum_{j=1}^n w_j \left(\frac{p_j}{p_0} - \frac{\tilde{p}_j}{\tilde{p}_0} \right)^2}$$

The forward error is the 2-norm difference between the root vector $\mathbf{z} = (z_1, \dots, z_m)$ of $\tilde{p}(x)$ and the root vector $\mathbf{x} = (x_1, \dots, x_m)$ of $p(x)$, assuming their components are properly ordered to match roots. The pejorative condition number κ measures the sensitivity of the roots with respect to *multiplicity preserving* perturbation [Zeng 2003]. If $p(x)$ and $\tilde{p}(x)$ have the same multiplicity structure $\ell = [\ell_1, \dots, \ell_m]$, then

$$\|\mathbf{z} - \mathbf{x}\|_2 \leq \kappa \|\mathbf{p} - \tilde{\mathbf{p}}\|_W,$$

ignoring some higher order terms [Zeng 2003].

If the given polynomial $p(x)$ is an *inexact* representation of the intended polynomial $\hat{p}(x)$, with the same multiplicity structure of the computed polynomial $\tilde{p}(x)$ and the exact roots $\mathbf{y} = (y_1, \dots, y_m)$, then

$$\|\mathbf{z} - \mathbf{y}\|_2 \leq 2\kappa \|\tilde{\mathbf{p}} - \mathbf{p}\|_W$$

as established in [Zeng 2003]. Since the intended polynomial $\hat{p}(x)$ is represented by $p(x)$, we estimate the forward error as

$$2\kappa \left\| \mathbf{p} - \tilde{\mathbf{p}} \right\|_w$$

If the multiplicity structure of the given polynomial can not be identified by GCDROOT, then MULTROOT automatically switches to the Matlab built-in **roots** command. In other words, the results of MULTROOT are at least as trustworthy as the Matlab standard results of **roots**.

3 Description of modules

Matlab modules are in the form of so called M-files with the generic name *module.m*, where *module* is the name of the module. The usage description of each module can be accessed by the Matlab command “help *module*”. The package MULTROOT consists of the following modules.

multroot	Given the coefficient vector of a polynomial, multroot computes the distinct roots, corresponding multiplicities, backward error, estimated forward root error, and the pejorative condition number.
gcdroot	Given the coefficient vector of a polynomial, gcdroot calculates the multiplicity structure of the polynomial and the initial root approximation.
pejroot	Given the coefficient vector of a polynomial, its multiplicity structure and initial root approximation, pejroot refines the distinct roots, calculates the pejorative condition number and the backward error.
backsub	Sub-module. Backward substitution for an upper-triangular linear system.
cauchymt	Sub-module. It generates the Cauchy matrix of a polynomial.
forsub	Sub-module. Forward substitution for a lower-triangular linear system.
hessqr	Sub-module. Hessenberg QR decomposition.
hqrt	Sub-module. The orthogonal transformation in Hessenberg QR decomposition.
scalsq	Sub-module. Scaled linear least squares solver.
sylnmat	Sub-module. It generates the Sylvester resultant matrix of three polynomials.
sylves	Sub-module. It generates the Sylvester discriminant matrix of a polynomial.
gcdgn	Sub-module. The Gauss-Newton iteration in GCD computation.
minsv	Sub-module. Computing the smallest singular value and the associated right singular vector of a matrix using an implicit inverse iteration.

The modules GCDROOT and PEJROOT can be used independently for expert users. GCDROOT calculates the multiplicity structure and the initial root approximations, without reaching the optimal root accuracy. PEJROOT is more than an accuracy refinement module. It is capable of calculating the (roots of the) polynomial with a predetermined multiplicity structure that is nearest to the given polynomial. This is particularly useful with polynomials that is near different structures [Zeng 2003].

4 The test suit

In the very first issue of *ACM Transaction on Mathematical Software* in 1975, M. A. Jenkins and J. F. Traub published *Principles for testing polynomial zerofinding programs*. Its conclusion worths repeating here [Jenkins et al. 1975]:

We feel that the polynomial zerofinding area has reached a level of maturity where good mathematical software does exist and criteria for discriminating between good and poor programs are reasonably well known. It seems to us a waste of effort for someone to publish a new polynomial zerofinding program unless it is at least competitive with the best of those available and/or it has important and relevant features that published programs do not have. A requirement for publication of a new algorithm should be a thorough evaluation of the reliability and efficiency of the program carried out by someone other than its author.

Since then, hundreds of more papers on root-finding [McNamee 1993] have been published. The testing principles proposed by Jenkins and Traub are not always followed. Apparently, benchmarks for testing polynomial root-finding programs are long overdue. In light of Jenkins-Traub Principles, we searched the literature for test polynomials and made a comprehensive test suit. Those polynomials have been used to test the robustness, stability, accuracy and efficiency of root-finders by many researchers, such as in [Jenkins et al. 1975, Loizou 1983, Brugnano et al. 1995, Dvorčuk 1969, Farmer et al. 1975, Goedecker 1994, Igarashi et al. 1995, Iliev 2000, Miyakoda 1989, Petkovic 1989, Stolan 1995, Toh et al. 1994, Uhlig 1999, Zeng 2003, Zhang 2001].

Because polynomial roots of high multiplicities have been considered extremely difficult in root finding, test polynomials we found in literature usually have very low multiplicities. Most of the test polynomials in the literature possess root multiplicities no more than five. In contrast, our MULTROOT program can easily calculate roots of multiplicities over 20. The module PEJROOT can even handle multiplicities in hundreds. Therefore, we modified many of the polynomials to set higher standard in testing our root-finder. A typical modification is to increase the multiplicities via repeatedly multiplying an existing test polynomial to itself. For example, from the polynomial

$$(x - 1)^4(x - 2)^3(x - 3)^2(x - 4)$$

used by Dvorčuk [Dvorčuk 1969] and Farmer-Loizou [Farmer et al. 1975, Loizou 1983], we generate a series of polynomials

$$p_k(x) = (x - 1)^{4k}(x - 2)^{3k}(x - 3)^{2k}(x - 4)^k, \quad k = 1, 2, \dots, 7.$$

The same technique can be applied to the polynomials in our test suit to generate more test problems.

The test suit consists of Matlab modules that generate test polynomials and their accurate roots. The generic file name, like our root finding modules, is *module.m*. A call

```
>> [p, z] = module
```

returns the polynomial coefficient vector p and root/multiplicity matrix z in the format of MULTROOT output, as in (4).

We list the test polynomials in Appendix A.

On all those test polynomials, especially those with multiple roots, our MULTROOT package successfully outputs accurate root/multiplicity results near machine precision, and far beyond the so-called “attainable accuracy”. To the best of our knowledge, there is no other method that can accurately calculate multiple roots of those polynomials with high multiplicities, with nearby multiple roots, or with inexact coefficients, such as fl03–07, twin02–04, miyak08, triple02–04, inex03–04 and large04–05. In addition to its remarkable accuracy on multiple roots, MULTROOT shows that it is at least as reliable, robust and stable as the standard softwares on all the test polynomials in the suit.

A Appendix: List of the test polynomials

The following are modules that generates polynomials in the test suit. The module names are in boldface, with $i = \sqrt{-1}$.

1. The Jenkins-Traub test polynomials [Jenkins et al. 1975]

- jt01a** $a = 10^{10}$, $p(x) = (x - A)(x + A)(x - 1)$
- jt01b** $a = 10^{-10}$, $p(x) = (x - A)(x + A)(x - 1)$
- jt02** $p(x) = (x - 1)(x - 2) \dots (x - 17)$
- jt03** $p(x) = (x - 0.1)(x - 0.001) \dots (x - 0.00000001)$
- jt04** $p(x) = (x - .1)^3(x - .5)(x - .6)(x - .7)$
- jt05** $p(x) = (x - .1)^4(x - .2)^3(x - .3)^2(x - .4)$
- jt06** $p(x) = (x - .1)(x - 1.001)(x - .998)(x - 1.00002)(x - .99999)$
- jt07a** $a = 10^{-10}$, $p(x) = (x - .001)(x - .01)(x - .1)(x - .1 + ai)(x - .1 - ai)(x - 1)(x - 10)$
- jt07b** $a = 10^{-9}$, $p(x) = (x - .001)(x - .01)(x - .1)(x - .1 + ai)(x - .1 - ai)(x - 1)(x - 10)$
- jt07d** $a = 10^{-7}$, $p(x) = (x - .001)(x - .01)(x - .1)(x - .1 + ai)(x - .1 - ai)(x - 1)(x - 10)$
- jt08** $p(x) = (x + 1)^5$
- jt09** $p(x) = (x^{10} - 10^{-20})(x^{10} + 10^{20})$
- jt10a** $a = 10^3$, $p(x) = (x - a)(x - a) \left(x - \frac{1}{a}\right)$
- jt10b** $a = 10^6$, $p(x) = (x - a)(x - a) \left(x - \frac{1}{a}\right)$
- jt10c** $a = 10^9$, $p(x) = (x - a)(x - a) \left(x - \frac{1}{a}\right)$
- jt11a** $m = 15$, roots: $e^{\frac{k\pi}{2m}i}$, $k = 1 - m, \dots, m - 1$; $0.9e^{\frac{k\pi}{2m}i}$, $k = m, \dots, 3m$
- jt11a** $m = 20$, roots: $e^{\frac{k\pi}{2m}i}$, $k = 1 - m, \dots, m - 1$; $0.9e^{\frac{k\pi}{2m}i}$, $k = m, \dots, 3m$
- jt11a** $m = 25$, roots: $e^{\frac{k\pi}{2m}i}$, $k = 1 - m, \dots, m - 1$; $0.9e^{\frac{k\pi}{2m}i}$, $k = m, \dots, 3m$

2. The Uhlig test polynomials [Uhlig 1999]

- uhlig01** $a = 0.01$, $p(x) = (x - a^4)(x - a)^4$
- uhlig02** $a = 0.001$, $p(x) = (x - a^4)(x - a)^4$

$$\begin{aligned}\text{uhlig03} \quad p(x) &= \left(x - \frac{3}{11}\right)^{12} \left(x - \frac{11}{3}\right)^2 \left(x - \frac{2}{7}i\right)^4 \left(x - 2.5 - \frac{i}{4}\right)^2 \left(x - \frac{1}{4}\right) \\ \text{uhlig04} \quad p(x) &= \left(x - \frac{3}{11}\right)^{12} \left(x - \frac{11}{3}\right)^2 \left(x - \frac{2}{7}i\right)^4 \left(x - 2.5 - \frac{i}{4}\right)^2 \left(x - \frac{1}{8}\right) \\ \text{uhlig05} \quad p(x) &= (x+1)^6 (x+2)^2\end{aligned}$$

3. The Goedecker's test polynomials [Goedecker 1994]

(a) Fibonacci polynomials $f_n(x) = x^n - x^{n-1} - \dots - x - 1$

- fib05** Fibonacci polynomial, $n = 5$
- fib10** Fibonacci polynomial, $n = 10$
- fib15** Fibonacci polynomial, $n = 15$
- fib20** Fibonacci polynomial, $n = 20$
- fib30** Fibonacci polynomial, $n = 30$
- fib50** Fibonacci polynomial, $n = 10$
- fib100** Fibonacci polynomial, $n = 100$
- fib150** Fibonacci polynomial, $n = 150$

(b) Squared Fibonacci polynomials $[f_n(x)]^2$

- fibsq04** squared Fibonacci polynomial, $n = 4$
- fibsq08** squared Fibonacci polynomial, $n = 8$
- fibsq16** squared Fibonacci polynomial, $n = 16$
- fibsq24** squared Fibonacci polynomial, $n = 24$
- fibsq32** squared Fibonacci polynomial, $n = 32$
- fibsq48** squared Fibonacci polynomial, $n = 48$

(c) The Legendre polynomials

- lgd05** Legendre polynomial, $n = 5$
- lgd10** Legendre polynomial, $n = 10$
- lgd15** Legendre polynomial, $n = 15$
- lgd20** Legendre polynomial, $n = 20$
- lgd24** Legendre polynomial, $n = 24$
- lgd50** Legendre polynomial, $n = 50$
- lgd100** Legendre polynomial, $n = 100$

4. Modifications of the Farmer-Loizou test polynomials. [Farmer et al. 1975, Loizou 1983, Zeng 2003]

$$\begin{aligned}\text{fl01} \quad k = 1, \quad p(x) &= (x-1)^{4k} (x-2)^{3k} (x-3)^{2k} (x-4)^k \\ \text{fl02} \quad k = 2, \quad p(x) &= (x-1)^{4k} (x-2)^{3k} (x-3)^{2k} (x-4)^k \\ \text{fl03} \quad k = 3, \quad p(x) &= (x-1)^{4k} (x-2)^{3k} (x-3)^{2k} (x-4)^k \\ \text{fl04} \quad k = 4, \quad p(x) &= (x-1)^{4k} (x-2)^{3k} (x-3)^{2k} (x-4)^k \\ \text{fl05} \quad k = 5, \quad p(x) &= (x-1)^{4k} (x-2)^{3k} (x-3)^{2k} (x-4)^k \\ \text{fl06} \quad k = 6, \quad p(x) &= (x-1)^{4k} (x-2)^{3k} (x-3)^{2k} (x-4)^k \\ \text{farloi01} \quad p(x) &= [(x^2 + x + 2)(x^2 + x + 3)]^4\end{aligned}$$

5. The Miyakoda test polynomial [Miyakoda 1989] and its modifications.

miyak00 $p(x) = (x - 1.1 - 1.1i)^4(x - 3.2 - 2.3i)^2(x - 2.1 - 1.5i)$

miyak02 square of miyak00

miyak04 square of miyak02

miyak08 square of miyak04

6. The Petkovic test polynomials [Petkovic 1989].

petk01 $p(x) = (x + 1)^2(x - 3)^3(x - i)^4(x^2 - 2x + 5)^2$

petk02 $p(x) = (x - 1)^2(x + i)^3(x - 5i)^2(x + 5i)^2$

petk03 $p(x) = 70(x^2 - 2x + 3)^2 \left(x - 1 - \frac{99}{70}i\right) (x + 1)$

petk04 $p(x) = (x - 3)(x + 1)^3(x - 2i)^3(x^2 + 4x + 5)^2(x^2 - 4x + 5)^2$

petk05 $p(x) = (x - 3)^2(x^2 + 2x + 5)^2(x + 1)^3$

petk06 $p(x) = (x - 3)^2(x^2 + 2x + 5)^2(x + 1)^4(x + i)^2$

petk07 $p(x) = (x - 1)^3(x^2 - 4x + 5)(x^2 + 25)^2$

henrici $p(x) = (x+4.1)(x+3.8)(x+2.05)(x+1.85)(x-1.95)(x-2.15)(x-3.9)(x-4.05)$

7. The Brugnano-Trigiante test polynomials [Brugnano et al. 1995]

bt01 $p(x) = (x - 1)^6(x + 1)^2(x + i)^3(x - i)^3(x - 2)$

bt02 $p(x) = (x - 1)^{10}(x - 2)^2(x + i)(x - i)$

bt03 $p(x) = (x^2 + 1)^5(x - 0.5i)^4(x + 0.5i)^4(x - 0.75i)(x + 0.75i)$

bt04 $p(x) = (x - 1)^3(x + 1)^4(x - .5 - i)^3(x - .5 + i)^3(x - .5 - .5i)^2(x - .5 + .5i)^2$

8. The Iliev test polynomial [Iliev 2000] and its modifications.

iliev00 $p(x) = (x - 1)(x + 2)^2(x - 3)^3$

iliev01 $p(x) = (x - 1)^2(x - 2)^4(x - 3)^6$

iliev02 $p(x) = (x - 1)^4(x - 2)^8(x - 3)^{12}$

iliev03 $p(x) = (x - 1)^8(x - 2)^{16}(x - 3)^{24}$

9. The Igarashi-Ypma test polynomials [Igarashi et al.1995].

igyp00 $p(x) = (x - 2.35)(x - 2.37)(x - 2.39)$

igyp01 $p(x) = (x - 2.35)^3(x - 2.56)$

igyp02a $m = 8, \quad p(x) = (x - 10 - 10i)^m(x + 1)^{10-m}$

igyp0ab $m = 7, \quad p(x) = (x - 10 - 10i)^m(x + 1)^{10-m}$

igyp0ab $m = 3, \quad p(x) = (x - 10 - 10i)^m(x + 1)^{10-m}$

10. The Toh-Trefethen test polynomials and modifications [Toh et al. 1994, Zhang 2001]

toh01 monic polynomial with roots $\frac{2(k+0.5)}{19} + i \sin \frac{2\pi(k+0.5)}{19}, k = -10, -9, \dots, 9$

toh02 $p(x) = \sum_{k=0}^{20} \frac{(10x)^k}{k!}$

toh03 monic polynomial with roots $\frac{10}{11} - 2^{-k}, k = 1, 2, \dots, 20$

toh04 $p(x) = (x - 10/11)^{20}$

toh05 monic polynomial with roots 2^{-k} , $k = 0, 1, \dots, 19$

toh06a $p(x) = 1 + x + x^2 + \dots + x^{20}$

toh06b $p(x) = (1 + x + x^2 + \dots + x^{10})^2$

toh06c $p(x) = (1 + x + \dots + x^5)^4$

11. Additional test polynomials proposed by the author [Zeng 2003].

(a) Polynomials with two nearby multiple roots 0.39 and 0.40.

twin01 $k = 4$, $p(x) = (x - 0.39)^k (x - 0.4)^k (x + 0.2)^k$

twin02 $k = 8$, $p(x) = (x - 0.39)^k (x - 0.4)^k (x + 0.2)^k$

twin03 $k = 12$, $p(x) = (x - 0.39)^k (x - 0.4)^k (x + 0.2)^k$

twin04 $k = 16$, $p(x) = (x - 0.39)^k (x - 0.4)^k (x + 0.2)^k$

(b) Polynomials with a cluster of three multiple roots $p(x) = (x - 0.9)^m (x - 1)^n (x - 1.1)^k$

triple01 $(m, n, k) = (5, 5, 5)$

triple02 $(m, n, k) = (10, 10, 10)$

triple03 $(m, n, k) = (18, 10, 16)$

triple04 $(m, n, k) = (20, 15, 10)$

(c) The polynomial $p(x) = \left(x - \frac{10}{11}\right)^5 \left(x - \frac{20}{11}\right)^3 \left(x - \frac{30}{11}\right)^2$ with inexact coefficients¹.

inex01 $p(x)$ above with 10 digits accuracy in coefficients

inex02 $p(x)$ above with 9 digits accuracy in coefficients

inex03 $p(x)$ above with 8 digits accuracy in coefficients

inex04 $p(x)$ above with 7 digits accuracy in coefficients

(d) Polynomials $p_\varepsilon(x) = (x - 1 - \varepsilon)^{20} (x - 1)^{20} (x + 0.5)^{20}$ with decreasing ε .

near01 $p_\varepsilon(x)$ with $\varepsilon = 0.1$

near02 $p_\varepsilon(x)$ with $\varepsilon = 0.01$

near03 $p_\varepsilon(x)$ with $\varepsilon = 0.001$

(e) Power of the monic polynomial $q(x)$ that is generated from simple roots $1, 1.2, -1 \pm .3i, -.9 \pm .4i, -.7 \pm .7i, -.4 \pm .9i, \pm 1.1i, .9 \pm .4i, .6 \pm .6i, .4 \pm .9i, \pm .8i$ with coefficients rounded up to 10 digits after decimal.

large01 polynomial $q(x)$

large02 square of large01

large03 square of large02

large04 square of large03

large05 square of large04

References

[Brugnano et al. 1995] Brugnanao, L. and Trigiane, D. 1995. Polynomial roots: the ultimate answer? *Linear Alg. and Its Appl.*, 225, 207-219.

¹The test polynomials in this category require manual adjustment of some control parameters of MULTROOT. For details, see [Zeng 2003]

- [Dvorčuk 1969] Dvorčuk, J. 1969. Factorization of a polynomial into quadratic factors by Newton method, *Apl. Mat.*, 14, 54-80.
- [Farmer et al. 1975] Farmer, M. R. and Loizou, G. 1975. A class of iteration functions for improving, simultaneously, approximation to the zeros of a polynomial, *BIT*, 15, 250-258.
- [Goedecker 1994] Goedecker, S. 1994, Remarks on algorithms to find oots of polynomials, *SIAM J. Sci. Comput*, 15, 1059-1063.
- [Igarashi et al.1995] Igarashi, M. and Ypma, T. 1995. Relationships between order and efficiency of a class of methods for multiple zeros of polynomials, *J. Comput. Appl. Math.*, Vol. 60, 101-113.
- [Iliev 2000] Iliev, A. I. 2000. A generalization of Obreshkoff-Ehrlich method for multiple roots of algebraic, trigonometric and exponential equations, *Math. Balkanica*, 14, 17-18.
- [Jenkins et al. 1975] Jenkins, M. A. and Traub J. F. 1975. Principles for testing polynomial zerofinding programs, *ACM Trans. Math. Software*, 1, 26-34.
- [Loizou 1983] Loizou, G. 1983. Higher-order iteration functions for simultaneously approximating polynomial zeros, *Intern. J. Computer Math.*, 14, 45-58.
- [McNamee 1993] McNamee, J. 1993. A bibliography on roots of polynomials, *J. Comput. Appl. Math.*, 47, 391-394.
- [Miyakoda 1989] Miyakoda, T. 1989. Iterative methods for multiple zeros of a polynomial by clustering, *J. Comput. Appl. Math.*, 28, 315-326.
- [Petkovic 1989] Petković, M. 1989. Iterative Methods for Simultaneous Inclusion of Polynomial zeros, Lecture Notes in Mathematics, 1387, Springer-Verlag,
- [Stolan 1995] Stolan, J. A. 1995. An improved Šiljak's algorithm for solving polynomial equations converges quadratically to multiple zeros, *J. Comput. Appl. Math.*, 64, 247-268.
- [Toh et al. 1994] Toh, K. C. and Trefethen, L. N. 1994. Pseudozeros of polynomials and pseudospectra of companion matrices, *Numer. Math.*, 68, 403-425.
- [Uhlig 1999] Uhlig, F. 1999. General polynomial roots and their multiplicities in $O(n)$ memory and $O(n^2)$ time, *Linear and Multilinear Algebra*, 46, 327-359.
- [Zeng 2003] Zeng, Z. 2003. Computing multiple roots of inexact polynomials, to appear, can be accessed at <http://www.neiu.edu/~zzeng/Papers/zroot.ps>.
- [Zhang 2001] Zhang, H. 2001. Numerical condition of polynomials in different forms, *Elec. Trans. Numer. Anal.*, 12, 66-87.

Zhonggang Zeng
 Department of Mathematics
 Northeastern Illinois University
 Chicago, IL 60625
 email: zzeng@neiu.edu