# THE SOLUTION OF ILL-CONDITIONAL POLYNOMIAL EQUATIONS

**Eglantina KALLUCI, Fatmir HOXHA**

Appl. Math. Department, Faculty of Natural Sciences, University of Tirana, Albania

**Abstract:** The problem of finding the roots of polynomial equations is encountered in many applications like image restoration, control theory and finance. A lot of well-known methods with good convergence rate are available, but in many real life problems we have to deal with multiple roots of polynomial equations which is an ill-conditional problem and the known methods do not give an accurate solution. In this paper we propose a method which at a first step reduces the conditional number and transforms the problem into a well-conditional one and at the second step involves a global convergence simultaneous method to approximate the solution. This technique is compared with best known simultaneous methods over Wilkinson polynomial, known for their ill-conditioning. The numerical tests are done in MATLAB and demonstrate the advantage of the proposed technique compared with other methods.

**Key words**: *polynomial, equation, conditional number, simultaneous, multiple, root.*

## 1. Introduction

The solution of polynomial equation is one of the most investigated topics in mathematics, and because of the missing general exact solvers for polynomial equations of power five and greater, the numerical methods lead the top solvers for them. Related with this aim we can use a vast literature as for example(Petkovic et al., 2014), (NcNamee, 2007), (Malek, F., Vaillancourt, R., (1995)) etc. One of the well-known numerical methods for solving not only polynomial equations, but even transcendent is Newton method. In certain cases, as in the presence of multiple roots and when the initial conditions are not near the exact root, this convergence of this method is slow or iterative process can diverge.

In this paper we will be focused not in improving a method with better convergence rate, but we will deal with another fundamental obstacle in the problem of finding the polynomial roots, which is known as ill-conditioning, so that small perturbation to the coefficients of the polynomial can result in large change in the roots. In general it is prudent to avoid ill-conditioning because it can increase sensitivity to round-off errors. The ill-conditioning is a

phenomenon which appears in many mathematical problems and algorithms including polynomial root finding.

## 2. The ill-conditional problem related to polynomial equations

A lot of authors take as a classic example of ill-conditioning in the context of finding polynomial roots the Wilkinson polynomial. This polynomial is defined by

$$f(x) = \prod_{i=1}^{20} (x - i). \tag{1}$$

Clearly, by definition the roots of this polynomial are the integers from 1 to 20 and the expanded form of this polynomial is given below

$$f(x) = \sum_{i=0}^{20} c_i x^i, \tag{2}$$

where the coefficients $c_i$ are given by

$$c_i = (-1)^i s_{20-i}(1, 2, 3, \cdots, 20). \tag{3}$$

It is convenient that small changes to any of the coefficients $c_i$ resulted in a similarly small change to the roots of the polynomial. Unfortunately, for this polynomial and as well for all ill-conditional polynomials this is far from the case. For these polynomials is proved that even a slight amount of perturbation up to $10^{-10}$ of the coefficients produces a drastic effect on the roots.

A well-known and often used metric for measuring the degree of ill-conditioning of a given problem or algorithm for solving that problem is the conditional number. The conditional number measures how sensitive a computed answer or output given by the algorithm is to changes in the input values or initial conditions. When defining the condition number, it is important to note that there are two ways in which changes to the initial and computed values may be measured; one may choose to use either the absolute change or the relative change.

**Definition**: Given a function $f.X \to Y$ where X and Y are normed vector spaces, the conditional number of f at any $x_0 \in X$ is defined by

$$\hat{\kappa} = \limsup_{\delta \to 0} \frac{\| f(x_0 + h) - f(x_0) \|}{\| h \|} \tag{4}$$

and the relative conditional number of f at $x_0$ is defined by

$$\hat{\kappa} = \limsup_{\delta \to 0} \frac{\dfrac{\| f(x_0 + h) - f(x_0) \|}{\| h \|}}{\dfrac{\| h \|}{\| x_0 \|}} . \tag{5}$$

In the case of polynomial root-finding, where X consists of the polynomial coefficients and Y consists of the roots we wish to obtain an expression for the condition number of a root of a polynomial with respect to a given coefficient.

The following theorem provides an expression for computing the condition number of any root with respect to any coefficient, provided that the root has multiplicity 1. Note that the conditional number of a multiple is always infinite because the derivative of a polynomial at a multiple root is 0, so that small changes to any coefficient can lead to arbitrary large changes in the roots.

**Theorem**: Let $f(x) = \sum_{i=0}^{n} c_i x^i$ be a degree n polynomial with coefficients $c_i$ for $0 \le i \le n$. If r is a nonzero root of $p(x)$ with multiplicity 1, and $c_i \neq 0$, then the relative conditional number of r with respect to $c_j$ is

$$\kappa = \frac{\left| c_j r^{j-1} \right|}{\left| p'(r) \right|} . \tag{6}$$

The proof is quite evident replacing the (6) on (5) an evaluating the limits.

At the beginning of this section we have taken as an example the Wilkinson polynomial. Using the definition (6) we see that for the root $r = 20$, the condition number with respect to $c_{20} = 1$ is $\kappa = \frac{20^{19}}{19!} \approx 4.31 \times 10^{7}$, whereas for $r = 1$ it is $\kappa = \frac{1}{19!} \approx 8.22 \times 10^{-18}$. Thus the root $r = 20$ is sensitive to changes with the leading coefficient $c_{20} = 1$ whereas the root $r = 1$ is virtually unaffected by those same changes.

### 3. Reducing the ill-conditioning for the polynomial equations

In this section we will deal with the case of reducing the condition number when we compute the multiple roots of polynomial equations.

In the following we shall consider the case when the order of multiplicity is not known. The idea most frequently used is to deflate all multiple roots into simple ones. So, if $\alpha$ is a multiple root of a polynomial $P$, then $\alpha$ is a simple root of the ratio $P / P'$.

**Corollary**. Assume that $P(z)$ has $n$ roots and among them there are $k$ distinct roots, each denoted by $\lambda_i$ with multiplicity $m_i$ for $i = 1, \cdots, k$, respectively. Then, $P(z)$ and its first derivative, $P'(z)$, have only one greatest common divisor (GCD)

$$P_c(z) = \prod_{i=1}^{k} (z - \lambda_i)^{m_i - 1}, \tag{7}$$

such that

$$P(z) = P_c(z) P_0(z) \qquad \text{and} \qquad P'(z) = P_c(z) P_1(z), \tag{8}$$

where $P_0(z)$ has exactly the same $k$ distinct roots, $\lambda_i$, as those of $P(z)$, which are all simple roots. The multiplicity of any root $\lambda_i$, can be determined by

$$m_i = \frac{P_1(\lambda_i)}{P_0^{'}(\lambda_i)}, \qquad \text{for } i = 1, \cdots, k. \tag{9}$$

Algorithm for finding simultaneously the polynomial roots with the respective multiplicities. Using the notations of the corollary we construct the following pseudocode.

**Step 1**. Compute $P'(z)$ of degree $n-1$.

**Step 2**. Find the GCD $d_{gc}(z)$ of $P(z)$ and $P'(z)$ using the Euclidean algorithm.

**Step 3.** Compute $q_p(z) = P(z)/d_{gc}(z)$ and $q_g(z) = P'(z)/d_{gc}(z)$.

**Step 4**. Employ the simultaneous method (Durand – Kerner (DK), or Erlich-Aberth(EA)) to determine all the $k$ roots $\lambda_i$, distinct and simple, of $q_p(z)$.

**Step 5**.The multiplicities $m_i = 1$ for $i = 1, \cdots, k$ if the GCD $d_{gc}(z)$ is a constant (polynomial). Otherwise, calculate the multiplicities $m_i = q_g(\lambda_i)/q_p'(\lambda_i)$.

**Step 6**. Output the $k$ roots $\lambda_i$ with their multiplicities $m_i$.

Two computational aspects of this algorithm are considered. Step 2 involves algebraic operations to search for the GCD of two polynomials. The step has computational complexity $O(n^2)$ with Euclidean algorithm, which is the extension to polynomials of the Euclidean algorithm for obtaining the GCD of two positive integers, or $O(n\log^2 n)$ with a fast version of the algorithm. Step 4 uses a simultaneous method to find simple roots of polynomials. The efficiency of this method that reveals its computational complexity is obtained by measuring the order of convergence. The order of convergence of EA method is three, and as we are using a simultaneous method for determining $k$ simple roots, this algorithm finds simultaneously even their multiplicities.

As we said in the previous section in the presence of the multiple root we have an ill-condition problem. Below we are going to present a method (numeric algorithm) *z-root-* for computing the simple roots of polynomials and *MultRoot-*

a numeric algorithm (not free) for computing the multiple roots of polynomials, presented by (Zeng, 2004A,B) in Numerical Recipes. The reason that this algorithm is not free, although it behaves very well in the presence with multiple roots, (Skowron and Gould, 2012)took the challenge to present an free algorithm.
It was for the same reasons that we tried to write a M-file in Matlab to deal with the case of ill-condition problem caused by the presence of multiple roots. We have compared with one of the most powerful algorithms that Matlab ofers for computing the polynomial root (the one that Zeng uses to compare his algorithm).

For making some numerical tests we have used the same polynomial that Zeng has used for testing MultRoot. Let be $p(x)$the following polynomial

$$p(x) = x^{10} - 17x^9 + 127x^8 - 549x^7 + 1521x^6 - 2823x^5 +$$

$$+3557x^4 - 3007x^3 + 163x^2 - 516x + 72$$

The Matlabcomand>>**roots**gives the following result,

```
>>roots([1 -17 127 -549 1521 -2823 3557 -3007 1634 -516 72])
ans =
   3.0000 + 0.0000i
   3.0000 - 0.0000i
   2.0003 + 0.0004i
   2.0003 - 0.0004i
1.9995
   1.0041
   1.0012 + 0.0039i
   1.0012 - 0.0039i
   0.9967 + 0.0024i
   0.9967 - 0.0024i
```

which graphically is presented in Figure 1.

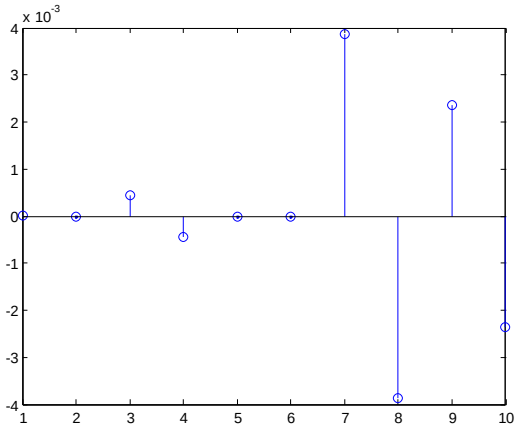


**Figure 1.** Graphical presentation of computed root of p(x) by the command >>roots.

We have writen a M-file of the algorithm presented before, tested for the same polynomial and it gives the following result:

>> modification([1 -17 127 -549 1521 -2823 3557 -3007 1634 -516 72])

| **Roots** | **Multiplicity** |
|---|---|
| 3.0000 | 2.0000 |
| 2.0000 | 3.0000 |
| 1.0000 | 5.0000 |

In Figure 2 the roots are presented graphically.

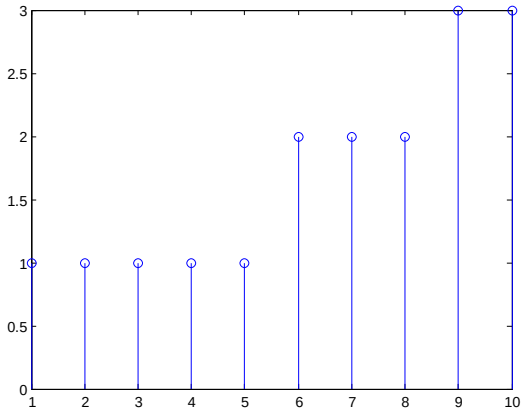


**Figure 2.** Graphical presentation of computed root of p(x) by the command **>>modification**.

The polynomial $p(x)$ is factorised as below

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

and it is obvious that the algorithm >> modification behaves very well in the presence of multiple roots, and the reason is that it reduces the condition numer and then computes the roots.

Most of the methods diverge when the multiplicity is greater than 10. If we take the sixth power of the upper polynomial we must deal with roots with respective multiplicities, 30, 18 and 12,

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

Our method gives the result below,

>> modification(p6)

**RootMuliplicity**

3.0002 12.0000
1.9997 18.0000
1.0002 30.0000

while the method>>***roots*** thatMatlab offers diverge.

**Conclusions**

In this paper we presented the advantages of a modified method for computing multiple roots of polynomial equations, which are classified in the group of ill-conditional problem. Most known methods diverge when multiplicity is higher than 10 and the aim of this paper was to present o method which at a first step reduces the conditional number and then computes the root and over all even the multiplicities of any root.

**References**

Malek, F., Vaillancourt, R., (1995), Polynomial zerofinding iterative matrix algorithms, Computers Math. Appl. 29 (1), 1-13.

McNamee, J. M., (2007), Numerical methods for roots of polynomials, Studies in Computational Mathematics 14, Elsevier.

Petkovic M., Neta B., Petkovic L., Dzunic J., (2014), Multipoint methods for solving nonlinear equations: A survey, Appl. Math. Comput. 226.

Skowron, J., Goulp, A. (2012), General complex polynomial root solver and its faster optimization for binary microlenses, arXiv:1203.

ZhengZ. (2004 A), Algorithm 835: MultRoot – A Matlab package for computing polynomial roots and multiplicities, ACM Trans. Math. Software 30, 218-236.

Zeng, Z. (2004B), Computing multiple roots of inexact polynomials, Math. Comp. 74, 869-903.