Scientific Research Publishing

Advances in Pure Mathematics, 2026, 16(6), 416-430
https://www.scirp.org/journal/apm
ISSN Online: 2160-0384
ISSN Print: 2160-0368

Method of Successive Polynomial Substitutions for Computing Roots of Polynomials

Serdar Beji

Faculty of Engineering and Natural Sciences, Department of Computer Engineering, Biruni University, Zeytinburnu, Istanbul, Türkiye
Email: sbeji@biruni.edu.tr

How to cite this paper: Beji, S. (2026) Method of Successive Polynomial Substitutions for Computing Roots of Polynomials. *Advances in Pure Mathematics*, **16**, 416-430.
https://doi.org/10.4236/apm.2026.166023

Received: April 23, 2026
Accepted: June 13, 2026
Published: June 16, 2026

Abstract

A recursive technique, termed method of successive polynomial substitutions for computing all the real and complex roots of a polynomial of any given degree, is introduced. The method proceeds by reducing the degree of polynomial by one at each stage of successive polynomial substitutions; extracts a root, and continues until reaching a second-degree polynomial whose roots are obtained analytically. Coefficients of the polynomial may be real or complex; no initial guess is needed and the results are highly accurate. Sample computations for polynomials of various degrees and the code used in computations are given.

Keywords

Real and Complex Roots of Polynomials, Method of Successive Polynomial Substitutions, Accurate and Efficient Computation of Zeros of Polynomials

1. Introduction

Numerous different techniques such as the bisection method, fixed-point iteration, Newton-Raphson method, are available for obtaining the roots of equations in general. To a lesser extent, there are solution techniques like Laguerre's method, Müller's method, and Bairstow's method, specific to the roots of polynomials (Chapra and Canale [1], pp. 123-202). The Weierstrass-Durand-Kerner method, essentially due to Weierstrass [2] but about 70 years later independently rediscovered by Durand [3] and Kerner [4], is also a well-known root-finding algorithm.

Based on the work of Traub [5], Jenkins and Traub [6] introduced an algorithm for computing zeros of polynomials with complex coefficients, programmed as the CPOLY algorithm [7]. A faster version of the algorithm for real polynomials was also formulated [8] and later presented as the RPOLY algorithm [9]. Cur-

rently, RPOLY is regarded as the standard in black-box polynomial root-finders due to its robustness, accuracy, and efficiency.

The present approach is a generalized and preconditioned version of a recursive method first introduced in Beji [10] for obtaining the roots of cubic polynomials. The method proceeds by reducing the degree of the polynomial by one at each stage of successive polynomial substitutions till the second degree. All the roots of a given polynomial, whether real or complex, are obtained in a single run without requiring any initial guess. The entire approach and the routine implementing it are found to be quite robust and accurate for polynomials of any degree with real or complex coefficients. When the implementation simplicity and efficiency of the method are considered, a professional version of the given generic algorithm may be a worthwhile competitor of the Jenkins-Traub algorithm.

2. Method of Successive Polynomial Substitutions

A polynomial of n^{th}-degree can be written as

$$P_n(x)=\sum_{j=0}^{n}a_j x^{n-j}=a_0x^n+a_1x^{n-1}+\cdots+a_{n-1}x+a_n, \tag{1}$$

where n is the degree of polynomial and a_j's are real or complex coefficients. Introducing a change of variable $x=\lambda z$ and setting $P_n(\lambda z)=0$ give

$$a_0(\lambda z)^n+a_1(\lambda z)^{n-1}+\cdots+a_{n-1}\lambda z+a_n=0, \tag{2}$$

where λ is a real or complex constant to be determined. Dividing the entire equation by $a_0\lambda^n$ results in

$$z^n+\frac{a_1}{\lambda a_0}z^{n-1}+\cdots+\frac{a_{n-1}}{\lambda^{n-1}a_0}z+\frac{a_n}{\lambda^n a_0}=0. \tag{3}$$

Following the approach introduced in Beji [10], we set $a_n/\lambda^n a_0=1$ and solve for $\lambda=(a_n/a_0)^{1/n}$ to transform Equation (3) into

$$z^n+\left(\frac{a_0}{a_n}\right)^{1/n}\frac{a_1}{a_0}z^{n-1}+\cdots+\left(\frac{a_0}{a_n}\right)^{1-1/n}\frac{a_{n-1}}{a_0}z+1=0. \tag{4}$$

Redefining the coefficients as $a_1\equiv(a_1/a_0)(a_0/a_n)^{1/n},\cdots,a_{n-1}\equiv(a_{n-1}/a_0)(a_0/a_n)^{1-1/n}$, Equation (4) is rewritten as

$$z^n+a_1z^{n-1}+\cdots+a_{n-1}z+1=0. \tag{5}$$

Depending on n being even (+) or odd (−), the multiplication of the roots of (5) is obviously $z_1z_2\cdots z_{n-1}z_n=\pm1$; therefore, Equation (5) has at least one root whose magnitude is less than unity, unless all the roots are equal in magnitude. Leaving the special case aside we can reason as follows. If z_i is the root less than unity, then,

$$|z_i|<1\Rightarrow|z_i|^n<|z_i|^{n-1}<\cdots<|z_i|^2<|z_i|. \tag{6}$$

In view of (6), a very rough approximation is possible by neglecting the term

with the highest power z^n in (5) and writing

$$a_1 z^{n-1} + \cdots + a_{n-1} z + 1 \approx 0, \tag{7}$$

which reduces the degree of polynomial to be solved by one. But it is possible to do much better by successive polynomial submissions. First, multiply Equation (5) by z and write it as

$$z^{n+1} = -a_1 z^n - a_2 z^{n-1} - \cdots - a_{n-1} z^2 - z. \tag{8}$$

Next, use (5) again to replace z^n on the right in (8):

$$z^{n+1} = -a_1 \left(-a_1 z^{n-1} - a_2 z^{n-2} - \cdots - a_{n-1} z - 1\right) - a_2 z^{n-1} - \cdots - a_{n-1} z^2 - z \tag{9a}$$

$$z^{n+1} = \left(a_1^2 - a_2\right) z^{n-1} + \left(a_1 a_2 - a_3\right) z^{n-2} + \cdots + \left(a_1 a_{n-2} - a_{n-1}\right) z^2 + \left(a_1 a_{n-1} - 1\right) z + a_1. \tag{9b}$$

We now have the term z^{n+1} on the left, and since $|z|^{n+1} < |z|^n$ for $|z| < 1$, neglecting z^{n+1} in (9b) would definitely result in a better approximation compared to (7), where z^n is neglected:

$$\left(a_1^2 - a_2\right) z^{n-1} + \left(a_1 a_2 - a_3\right) z^{n-2} + \cdots + \left(a_1 a_{n-2} - a_{n-1}\right) z^2 + \left(a_1 a_{n-1} - 1\right) z + a_1 \approx 0. \tag{10}$$

Repeating the substitutions as many times as desired according to a definite criterion eventually leads to a polynomial one degree less, $(n-1)$, than the original polynomial of degree n. By recalling that the sum of roots is equal to the opposite sign value of the coefficient of the second highest power term, the first root eliminated in the reduction process can easily be determined by a simple subtraction as $x_1 = -c_1 + d_2$. Here, c_1 is the coefficient of x^{n-1} term of the original polynomial of degree n and d_2 is the coefficient of x^{n-2} term of the reduced polynomial of degree $(n-1)$. Afterwards, the same recursion process is applied to the reduced polynomial to reduce it one degree more and subsequently the second root is computed. The recursive process is repeated until a second-degree polynomial is obtained and the last two roots are computed from the well-known analytical expression, the quadratic formula. Afterwards, roots of the original polynomial can be obtained from $x = \lambda z$. But it turns out that this transformation is not necessary at all because the inverse transformation following successive substitutions yields a polynomial which is identical to the one that can be obtained without transformation. Therefore, the change of variable, $x = \lambda z$, should be viewed only as a formal justification process of the present approach.

2.1. A Demonstrative Application

To clarify the method further we apply it to a general cubic polynomial $P_3(x) = a_0 x^3 + a_1 x^2 + a_2 x + a_3$ without transformation. First, set $P_3(x) = 0$, divide by a_0 and redefine its new coefficients as $a_1 = a_1 / a_0$, etc. so that

$$x^3 = -a_1 x^2 - a_2 x - a_3. \tag{11}$$

Neglecting the x^3 term results in a zeroth-order approximation with $a_1 x^2 + a_2 x + a_3 \approx 0$. For a better approximation multiply Equation (11) by x and replace the x^3 term by $-a_1 x^2 - a_2 x - a_3$ so that

$$x^4 = \left(a_1^2 - a_2\right)x^2 + \left(a_1 a_2 - a_3\right)x + a_1 a_3. \tag{12}$$

We get a first-order approximation now if the term x^4 is neglected,

$$\left(a_1^2 - a_2\right)x^2 + \left(a_1 a_2 - a_3\right)x + a_1 a_3 \approx 0. \tag{13}$$

The following second-degree polynomials listed in Table 1 are obtained for the first three approximations.

Table 1. Quadratic polynomials resulting from successive substitutions of a general cubic polynomial.

Degree Neglected	Corresponding Quadratic Polynomial
3rd	$a_1 x^2 + a_2 x + a_3 = 0$
4th	$\left(a_1^2 - a_2\right)x^2 + \left(a_1 a_2 - a_3\right)x + a_1 a_3 = 0$
5th	$\left[a_1\left(2a_2 - a_1^2\right) - a_3\right]x^2 + \left[a_2\left(a_2 - a_1^2\right) + a_1 a_3\right]x + a_3\left(a_2 - a_1^2\right) = 0$

Higher-order approximations are obtained in the same manner; however, the coefficients get larger and cause computational inaccuracies. To avoid this problem, the reduced polynomial should be divided by the coefficient of x^2 term at each step. Note that this coefficient would never become zero as we must have a second-degree polynomial to get two more roots besides the one eliminated through the degree-reduction process. In other words, the fundamental theorem of algebra ensures a non-zero coefficient for the highest term of the reduced polynomial.

As a numerical example, we now consider a particular cubic polynomial $P_3(x) = (x-1)(x-2)(x-3) = x^3 - 6x^2 + 11x - 6$ whose roots are obviously $x_1 = 1$, $x_2 = 2$, and $x_3 = 3$. Following the above outlined method and employing the suggested normalization at each step result in Table 2. Roots obtained from the last quadratic $x^2 - 2.964x + 1.964 = 0$ are $x_1 = 1.000$ and $x_2 = 1.964$ and the process is clearly leading to $x^2 - 3.000x + 2.000 = 0$ with roots $x_1 = 1.000$ and $x_2 = 2.000$.

Table 2. Quadratic polynomials resulting from successive substitutions of a particular cubic polynomial.

Degree Neglected	Corresponding Quadratic Polynomial
3rd	$x^2 - 1.833x + 1.000 = 0$
5th	$x^2 - 2.655x + 1.666 = 0$
10th	$x^2 - 2.964x + 1.964 = 0$

2.2. Formulation of Recursive Scheme

With the guidance of preceding examples, we can formulate the recursive scheme to be employed for determining coefficients of the reduced degree, x^{n-1}, polynomial. First, a single sweep is performed for introducing a second set of coefficients:

$$\begin{aligned} b_1 &= a_1^2 - a_2, \\ b_i &= \left(a_1 a_i - a_{i+1}\right)/b_1 \quad \text{for } i = 2,\cdots,n-1 \\ b_n &= a_1 a_n / b_1, \end{aligned} \tag{14}$$

where b_i's are the normalized coefficients exemplified in Table 2. Then, a succession of sweeps till a definite convergence criterion is met:

$$\begin{aligned} b_1 &= b_2 - a_1, \\ b_i &= \left(b_{i+1} - a_i\right)/b_1 \quad \text{for } i = 2,\cdots,n-1 \\ b_n &= -a_n / b_1, \end{aligned} \tag{15}$$

The scheme above reduces n^{th}-degree polynomial to the $(n-1)^{\text{th}}$-degree polynomial and therefore can be used only for reducing a cubic polynomial $n=3$ to a quadratic polynomial for obtaining all the roots. For higher degree polynomials $n>3$ it is necessary to reduce the degree of polynomial more than once till a quadratic polynomial is obtained. The generalized version, which reduces the degrees successively from n to $n-1, n-2, \cdots, 2$, is for the initial sweep

$$\begin{aligned} b_j &= a_j^2 - a_{j+1}, \quad \text{for } j = 1,\cdots,n-2 \\ b_i &= \left(a_i a_j - a_{i+1}\right)/b_j \quad \text{for } i = j+1,\cdots,n-1 \\ b_n &= a_j a_n / b_j, \end{aligned} \tag{16}$$

and for the succession of sweeps till convergence

$$\begin{aligned} b_j &= b_{j+1} - a_j, \quad \text{for } j = 1,\cdots,n-2 \\ b_i &= \left(b_{i+1} - a_i\right)/b_j \quad \text{for } i = j+1,\cdots,n-1 \\ b_n &= -a_n / b_j. \end{aligned} \tag{17}$$

Note that the special case formulated by Equations (14) and (15) corresponds to $n=3$ hence $j=1$ to $n-2=1$, just one-degree reduction in the general scheme of (16) and (17). FORTRAN code given in the Appendix basically uses Equations (16) and (17). At any stage of degree-reduction the repeated sweeps continue until the absolute value of the last coefficient of the polynomial, b_n, differs less than 10^{-14} from the one computed in the previous step. Checking only the last coefficient is sufficient as all the coefficients are computed interrelatedly and the convergence of a coefficient implies the convergence of all the coefficients. Computations are carried out entirely as complex numbers so that roots of polynomials in most general forms, including complex polynomial coefficients, can be computed.

3. Preconditioning

Virtually every numerical technique requires the satisfaction of a definite condition or conditions to prevent its failure. As it is obvious from Equations (8) and (9), for the present methodology this condition is $a_1 \neq 0$. By a simple shift, which may be called as the *preconditioning procedure*, this particular requirement can be handled and the scheme works virtually for all polynomials. Preconditioning is achieved by changing the independent variable, say x, to $z+\theta$ and then specify-

ing the shift θ in a way to set a specific coefficient of the polynomial to a desired value. In order to implement such a measure, it is first necessary to introduce a convenient way of formulating the intended shift.

3.1. Transformation of Polynomials via Taylor Series

A change of independent variable, $x = z + \theta$, transforms the general n^{th} degree polynomial $P_n(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n$ via the Taylor series expansion, into $P_n(z+\theta)$:

$$P_n(z+\theta) = P_n(\theta) + P_n'(\theta) z + P_n''(\theta)\frac{z^2}{2!} + \cdots + P_n^{(n-1)}(\theta)\frac{z^{n-1}}{(n-1)!} + P_n^{(n)}(\theta)\frac{z^n}{n!}, \quad (18)$$

where θ is a real or complex constant to be determined and primes indicate differentiation with respect to the independent variable. The above expression for instance is quite practical for implementing the so-called Tschirnhaus transformation [11] which makes the coefficient of z^{n-1} term zero; specifically, $P_n^{(n-1)}(\theta)/(n-1)! = 0$. Incidentally, the transformation attempted by Tschirnhaus was more general as he stated in the first sentence of his article: *"We have learned from DesCartes' geometry by what method the second term might reliably be removed from a given equation; but on the question of removing multiple intermediate terms I have seen nothing hitherto in the analytic arts."* but was not as much successful as he claimed. Thus, according to Tschirnhaus himself, DesCartes was the first to introduce the method to remove the *second term*; that is, $a_1 x^{n-1}$ term.

The use of (18) for easily obtaining the roots of quadratic equation $P_2(x) = a_0 x^2 + a_1 x + a_2$ by removing the *second term* is now demonstrated. In the transformed expression, $P_2(z+\theta) = P_2''(\theta) z^2/2! + P_2'(\theta) z + P_2(\theta)$, setting $P_2'(\theta) = 2a_0\theta + a_1 = 0$ gives $\theta = -a_1/2a_0$. Then, $P_2(\theta) = a_0 \cdot (-a_1/2a_0)^2 + a_1 \cdot (-a_1/2a_0) + a_2$ or $P_2(\theta) = (4a_0a_2 - a_1^2)/4a_0$. On the other hand, when $P_2'(\theta) = 0$, the transformed polynomial becomes $P_2''(\theta) z^2/2! + P_2(\theta) = 0$ hence $z = \pm\left[-2P_2(\theta)/P_2''(\theta)\right]^{1/2}$. Noting that $P_2''(\theta) = 2a_0$ and that $x = z + \theta$ we obtain the roots as $x = \theta \pm \left[-2P_2(\theta)/P_2''(\theta)\right]^{1/2}$, which, in turn, can be shown to be identical with the well-known quadratic formula. Applications of (18) to cubic, quartic, and quintic equations can be found in Beji [12].

3.2. Preconditioning Measure

Clearly, coefficient a_1 which multiplies x^{n-1} must be *nonzero* if the successive polynomial substitutions are to be carried out successfully as Equations (8) and (9) show. Equivalently, when the transformed polynomial is considered, the coefficient of z^{n-1} must never vanish to prevent a complete failure of the procedure. Thus, for ensuring a nonzero coefficient for z^{n-1} in the transformed polynomial, $P_n^{(n-1)}(\theta)/(n-1)!$ is set to a constant, $n(1+0.5i)$, where $i = \sqrt{-1}$ is the imaginary unit. Although in principle any constant would work, the value $n(1+0.5i)$ is selected based on systematic trial computations with polynomials of differing degrees.

The trials revealed that this particular value typically results in fastest convergence rates and that unequal real and imaginary parts is essential for convergence in special cases such as $a_0 x^n + a_n = 0$, where all the intermediate coefficients are zero. Accordingly,

$$\frac{P_n^{(n-1)}(\theta)}{(n-1)!} = \frac{n!a_0\theta + (n-1)!a_1}{(n-1)!} = n(1+0.5i) \Rightarrow \theta = \left[(1+0.5i) - a_1/n\right]/a_0 . \quad (19)$$

Once the shift θ is determined according to (19), all the coefficients of transformed polynomial can easily be computed with the help of (18), and this is precisely the procedure followed for the code given in the Appendix.

4. Sample Computations

The methodology is implemented in the FORTRAN program given in the Appendix. Maximum degree of polynomials to be solved is arbitrarily set to $n\max = 100$; likewise, the maximum number of allowed iterations is defined as $i\max = 10000$. In accord with the programming language and machine capacity, the computational precision for convergence at each degree-reduction is specified as $\epsilon = 10^{-14}$. The coefficients of polynomial are entered in complex number format in parenthesis as the code is written to accommodate the most general case possible. Results are given in 10 decimal places.

4.1. $P_3(x) = 6x^3 - 17x^2 - 5x + 6$

Figure 1 shows the execution outcome of the code for a third-degree polynomial constructed as $P_3(x) = (3x+2)(x-3)(2x-1) = 6x^3 - 17x^2 - 5x + 6$ with roots $x_1 = -0.6666666667$, $x_2 = 3.0000000000$, and $x_3 = 0.5000000000$, which are all real. For this case, only 56 iterations or sweeps are needed to reduce the degree of polynomial from 3rd to 2nd with $\epsilon = 10^{-14}$ precision and these 56 iterations are all the needed to compute three roots, which are exact to 10 decimal places. To check the accuracy of each computed root, the polynomial is evaluated too; the results are shown on the right.

```
  Enter the degree of polynomial:

3
  Starting from the highest degree term enter the coefficients of polynomial:

  a 0
(6,0)
  a 1
(-17,0)
  a 2
(-5,0)
  a 3
(6,0)
```

	Root		Value of Polynomial	
Root Number	Real	Imaginary	Real	Imaginary
x 1	-.6666666667	.0000000000	.0000000000	.0000000000
x 2	3.0000000000	.0000000000	.0000000000	.0000000000
x 3	.5000000000	.0000000000	.0000000000	.0000000000

Figure 1. Screenshot of code execution for $P_3(x) = 6x^3 - 17x^2 - 5x + 6$.

4.2. $P_4(x) = 3x^4 - 2x^3 + x^2 + 4x + 5$

A fourth-degree polynomial with arbitrarily selected real coefficients is considered and the code gives two pairs of complex conjugate zeros. Figure 2 shows the results which are quite accurate as revealed by the polynomial evaluations, all zero, given on the right. Reducing the degree of polynomial from 4th to 3rd requires 209 iterations and from 3rd to 2nd, 245 iterations; hence, altogether, obtaining all the roots takes 454 sweeps.

	Root		Value of Polynomial	
Root Number	Real	Imaginary	Real	Imaginary
x 1	-.6574201029	-.5792172500	.0000000000	.0000000000
x 2	-.6574201029	.5792172500	.0000000000	.0000000000
x 3	.9907534363	1.0906016925	.0000000000	.0000000000
x 4	.9907534363	-1.0906016925	.0000000000	.0000000000

Figure 2. Screenshot of code execution for $P_4(x) = 3x^4 - 2x^3 + x^2 + 4x + 5$.

4.3. $P_5(x) = (-2 + 3i)x^5 + (5 + 5i)x^4 + (0 - i)x^3 + (7 + 0i)x^2 + (1 - 2i)x + (-15 + 12i)$

A fifth-degree polynomial with arbitrarily selected real and complex coefficients is considered. The computed roots, seen in Figure 3, are all complex but none complex conjugate because the polynomial has complex coefficients. Iteration numbers for computational sequences, from 5th down to 2nd degree are 218, 4507, and 60, summing up to 4785 sweeps altogether for the computational precision requirement $\epsilon = 10^{-14}$. Functional evaluations shown on the right are all zero to 10 decimal places, verifying that the computed roots are quite accurate.

```
   Enter the degree of polynomial:

5
   Starting from the highest degree term enter the coefficients of polynomial:

   a 0
(-2,3)
   a 1
(5,5)
   a 2
(0,-1)
   a 3
(7,0)
   a 4
(1,-2)
   a 5
(-15,12)
```

	Root		Value of Polynomial	
Root Number	Real	Imaginary	Real	Imaginary
x 1	-.3631170006	-1.2294382569	.0000000000	.0000000000
x 2	-.8804916077	2.0220748005	.0000000000	.0000000000
x 3	-1.1233638605	.3412939289	.0000000000	.0000000000
x 4	1.0181480774	1.1678730283	.0000000000	.0000000000
x 5	.9642090068	-.3787265778	.0000000000	.0000000000

Figure 3. Screenshot of code execution for $P_5(x) = (-2 + 3i)x^5 + (5 + 5i)x^4 + (0 - i)x^3 + (7 + 0i)x^2 + (1 - 2i)x + (-15 + 12i)$.

4.4. $P_7(x) = x^7 - 6.01x^6 + 12.54x^5 - 8.545x^4 - 5.505x^3 + 12.545x^2 - 8.035x + 2.01$

A particularly difficult case used in Jenkins and Traub [8] is factored as $P_7(x) = (x - 0.5 + 0.5i)(x - 0.5 - 0.5i)(x - 1)^2(x + 1)(x - 2)(x - 2.01)$. Obviously,

the repeated roots originating from the factor $(x-1)^2$ and the very close magnitude roots from $(x-2)(x-2.01)$ may cause computational problems. Figure 4 shows the computed roots and corresponding polynomial evaluations to the 10 decimal places. For the five distinct zeros, the present algorithm works virtually perfectly well and definitely better than the Jenkins-Traub algorithm. Only the coincident roots, x_4 and x_5, resulting from $(x-1)^2=0$ could not be obtained with the accuracy of $\epsilon=10^{-14}$ despite the allowed maximum 10,000 iterations, simply because the code cannot decide which one of the roots to converge as both are the same. Note however that the close magnitude roots $x_6=2.01$ and $x_7=2$ are computed exactly to 10 decimal places of accuracy. Iteration numbers for computational sequences, from 7th down to 2nd degree are 58, 135, 108, 10,000, and 54, summing up to 10,355 sweeps altogether. The greatest portion of iterations, counting to 10,000 due to the repeated roots, could be reduced down to 5,198 by setting $\epsilon=10^{-8}$ without appreciably affecting the accuracy of the computed roots. Nevertheless, it is clear that repeated roots are the weakest spot of the present scheme and its source may be traced back to the paragraph between Equations (5) and (6).

	Root		Value of Polynomial	
Root Number	Real	Imaginary	Real	Imaginary
x 1	-1.0000000000	.0000000000	.0000000000	.0000000000
x 2	.5000000000	-.5000000000	.0000000000	.0000000000
x 3	.5000000000	.5000000000	.0000000000	.0000000000
x 4	.9999141757	-.0000499914	.0000000049	.0000000087
x 5	1.0000858243	.0000499914	.0000000049	.0000000087
x 6	2.0100000000	.0000000000	.0000000000	.0000000000
x 7	2.0000000000	.0000000000	.0000000000	.0000000000

Figure 4. Screenshot of code execution for $P_7(x) = x^7 - 6.01x^6 + 12.54x^5 - 8.545x^4 - 5.505x^3 + 12.545x^2 - 8.035x + 2.01$.

4.5. $P_9(x) = (-2 + i)x^9 + (1 + i)x^8 + (3 - 2i)x^7 + (5 + 0i)x^6 + (-4 + 3i)x^5 + (7 + 7i)x^4 + (6 + 0i)x^3 + (-3 + 0i)x^2 + (2 + 2i)x + (10 + 10i)$

A ninth-degree polynomial with arbitrarily selected real and complex coefficients is the last example treated. As in §4.3 the roots shown in Figure 5 are all complex but none complex conjugate because the polynomial has complex coefficients. The results are highly accurate as all the functional evaluations are zero to the 10 decimal places. Number of sweeps for sequences from 9th to 2nd degree are 369, 314, 875, 133, 108, 440, and 95, adding up to 2334 sweeps for the entire computations.

	Root		Value of Polynomial	
Root Number	Real	Imaginary	Real	Imaginary
x 1	-.9014082021	-1.0802519051	.0000000000	.0000000000
x 2	-1.2366213366	1.0934743900	.0000000000	.0000000000
x 3	-.7618747052	-.5321582789	.0000000000	.0000000000
x 4	-.9966108613	.4241798280	.0000000000	.0000000000
x 5	.5610334224	-.9542154792	.0000000000	.0000000000
x 6	.9184116497	-.6518456801	.0000000000	.0000000000
x 7	.0310745202	1.1244379595	.0000000000	.0000000000
x 8	1.8398052054	.4910645128	.0000000000	.0000000000
x 9	.7461903075	.6853146528	.0000000000	.0000000000

Figure 5. Screenshot of code execution for $P_9(x) = (-2 + i)x^9 + (1 + i)x^8 + (3 - 2i)x^7 + (5 + 0i)x^6 + (-4 + 3i)x^5 + (7 + 7i)x^4 + (6 + 0i)x^3 + (-3 + 0i)x^2 + (2 + 2i)x + (10 + 10i)$.

4.6. Some Special Cases

The most critical special case that could make the present scheme fail has been identified as $a_1 = 0$, and this pitfall is avoided by a shift that sets the corresponding term in the transformed polynomial to a constant. This preconditioning measure avoids quite a number of problems associated with a variety of special cases. In this section, we are going to enumerate some noteworthy special cases for which, despite unusual characteristics of polynomial, the code still performs well.

Polynomials like $a_0 x^n + a_n = 0$ may at first sight appear problematic since all the coefficients but the fist and the last, are zero. However, the scheme, in virtue of a transformation to set the coefficient of the second term to a constant, handles all these cases accurately. For instance, for $\epsilon = 10^{-14}$ with 90 iterations, roots of $x^3 + 1 = 0$ are computed as $x_1 = -1.0000000000 + 0.0000000000$, $x_2 = 0.5000000000 - 0.8660254038i$, $x_3 = 0.5000000000 + 0.8660254038i$, which may be regarded exact. Likewise, it takes 157 + 96 = 253 iterations to compute all the roots of $x^4 + 1 = 0$ as $x_1 = -0.7071067812 - 0.7071067812i$, $x_2 = -0.7071067812 + 0.7071067812i$, $x_3 = 0.7071067812 - 0.7071067812i$, and $x_4 = 0.7071067812 + 0.7071067812i$. Higher degree cases $x^5 + 1 = 0$, $x^6 + 1 = 0$, which are tested to the 10^{th} degree, are solved with the same ease and accuracy without problem.

For the present formulation, as observed in §4.4 of the Jenkins-Traub test, repeated or coincident roots of the form $(x + c)^n = 0$ poses a potential problem since the procedure is not able to decide for a root to converge. Unsurprisingly, for $(x+1)^3 = x^3 + 3x^2 + 3x + 1 = 0$ the code produces just barely acceptable results $x_1 = -1.0001999401 - 0.0000999701i$, $x_2 = -0.9998500497 - 0.0000499752i$, $x_3 = -0.9999500102 + 0.0001499453i$ after the maximum allowed 10,000 iterations. For $(x+1)^4 = x^4 + 4x^3 + 6x^2 + 4x + 1 = 0$ we obtain similar results after 10,000 + 10,000 = 20,000 iterations. For higher degrees we get acceptable results but clearly the repeated zeros are not easily handled. A remedy to this weakest spot should be implemented in the professional version of the code in an appropriate way.

Quite trivial case of $a_0 x^n = 0$ is problematic; again due to repeated roots, but can still be computed. For instance, $x^3 = 0$ is solved after the maximum allowed 10,000 iterations as $x_1 = -0.0001999412 - 0.0000999698i$, $x_2 = 0.0001499502 - 0.0000499782i$, and $x_3 = 0.0000499911 + 0.0001499480i$. The fourth-degree case $x^4 = 0$ produces similar near zero values after 10,000 + 10,000 = 20,000 iterations. Nevertheless, relatively low performance of the method for this most primitive form, $a_0 x^n = 0$, does not imply a serious defect at all since the solution is trivial.

The above screening of main special cases clearly shows that the repeated roots are the only special case that impair the accuracy of computations for the proposed root-finding algorithm. No case has been identified with a complete failure; though, naturally, there may remain some overlooked exceptional cases. Overall however, the methodology works remarkably well and produces quite accurate solutions for polynomials with real or complex coefficients.

5. Concluding Remarks

A new method of computing all the roots of a polynomial of any degree is introduced. The scheme proceeds by reducing the degree of the given polynomial by one at each stage of successive polynomial substitutions till reaching a quadratic. A simple shift via Taylor series expansion is introduced to implement a preconditioning measure for avoiding failure of the scheme due to the vanishing of the second highest-degree term. Computations are performed in the complex domain for the sake of generality and several demonstrative examples are included. Except for repeated roots, the code works virtually perfectly well with high efficiency and accuracy without any observed failure. A professionally restructured form of the present generic algorithm in different programming languages is expected to be very useful for practical uses as its performance proves to be quite competitive even against the Jenkins-Traub algorithm, which is the standard root-finder.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Chapra, S.C. and Canale, R.P. (2005) Numerical Methods for Engineers. McGraw-Hill Education.

[2] Weierstrass, K. (1891) Neuer Beweis des Satzes, dass jede ganze rationale Function einer Veranderlichen dargestellt werden kann als ein Product aus linearen Functionen derselben Veranderlichen. Sitzungsberichte der Königlich Preussischen Akademie der Wissenschaften zu Berlin. https://web.archive.org/web/20131102093616/http://bibliothek.bbaw.de/bibliothek-digital/digitalequellen/schriften/anzeige?band=10-sitz%2F1891-2&seite%3Aint=00000565

[3] Durand, E. (1960) Equations du type $F(x)=0$: Racines d'un polynome. In: *Solutions Numeriques des Equations Algebriques*, Vol. 1.

[4] Kerner, I.O. (1966) Ein Gesamtschrittverfahren zur Berechnung der Nullstellen von Polynomen. *Numerische Mathematik*, **8**, 290-294. https://doi.org/10.1007/bf02162564

[5] Traub, J.F. (1966) A Class of Globally Convergent Iteration Functions for the Solution of Polynomial Equations. *Mathematics of Computation*, **20**, 113-138. https://doi.org/10.1090/s0025-5718-1966-0192655-2

[6] Jenkins, M.A. and Traub, J.F. (1970) A Three-Stage Variable-Shift Iteration for Polynomial Zeros and Its Relation to Generalized Rayleigh Iteration. *Numerische Mathematik*, **14**, 252-263. https://doi.org/10.1007/bf02163334

[7] Jenkins, M.A. and Traub, J.F. (1972) Algorithm 419: Zeros of a Complex Polynomial. *Communications of the ACM*, **15**, 97-99. https://doi.org/10.1145/361254.361262

[8] Jenkins, M.A. and Traub, J.F. (1970) A Three-Stage Algorithm for Real Polynomials Using Quadratic Iteration. *SIAM Journal on Numerical Analysis*, **7**, 545-566. https://doi.org/10.1137/0707045

[9] Jenkins, M.A. (1975) Algorithm 493: Zeros of a Real Polynomial. *ACM Transactions on Mathematical Software*, **1**, 178-189. https://doi.org/10.1145/355637.355643

[10] Beji, S. (1992) Investigations on Cubic Polynomials. *International Journal of Mathematical Education in Science and Technology*, **23**, 167-173. https://doi.org/10.1080/0020739920230201.

[11] Tschirnhaus, E.W. (2003) A Method for Removing All Intermediate Terms from a Given Equation. *ACM SIGSAM Bulletin*, **37**, 204-207. https://sigsam.org/bulletin/issues/

[12] Beji, S. (2008) A Systematic Approach to the Exact Roots of Polynomials. *Mediterranean Journal of Mathematics*, **5**, 163-172. https://doi.org/10.1007/s00009-008-0141-6

Appendix: FORTRAN Code for Implementing Method of Successive Polynomial Substitutions: SPS

```
use msimsl
parameter(nmax=100,imax=10000,eps=1.e-14)
double complex a(0:nmax),b(1:nmax),c(0:nmax),x(1:nmax),xb(1:nmax)
double complex dlt,poly,tht
c
write(*,10)
10 format(3x,31HEnter the degree of polynomial:,/)
read(*,*)n
c
write(*,20)
20 format(3x,37HStarting from the highest degree term,
&38H enter the coefficients of polynomial:,/)
c
do i=0,n
write(*,30)i
30 format(3x,1Ha,i2)
read(*,*)a(i)
c(i)=a(i)
enddo
c
c       Divide all coefficients by a(0)
do i=1,n
a(i)=a(i)/a(0)
enddo
a(0)=dcmplx(1.,0.)
c
c       Preconditioning of polynomial by setting a(1) to (n,n/2)
tht=(dcmplx(1.,1./2.)-a(1)/n)/a(0)
c
c       Recompute polynomial coefficients for making a(1)=(n,n/2)
do j=n,1,-1
a(j)=a(j)*dfac(n-j)
do k=j-1,0,-1
a(j)=a(j)+a(k)*(dfac(n-k)/dfac(j-k))*tht**(j-k)
enddo; enddo
c
c       Coefficients of transformed polynomial
do j=n,1,-1
a(j)=a(j)/dfac(n-j)
enddo
c
```

```
c
c       Main Program to solve roots of transformed polynomial
c
do 50 j=1,n-2
c
c       A single substitution
b(j)=a(j)*a(j)-a(j+1)
do i=j+1,n-1
b(i)=(a(i)*a(j)-a(i+1))/b(j)
enddo
b(n)=a(j)*a(n)/b(j)
c
c       Repeat till convergence
c
k=0
c
40 xb(n)=b(n)
c
k=k+1
c
b(j)=b(j+1)-a(j)
do i=j+1,n-1
b(i)=(b(i+1)-a(i))/b(j)
enddo
b(n)=-a(n)/b(j)
c
if(abs(xb(n)-b(n)).gt.eps.and.k.lt.imax) then
goto 40
else; goto 99; endif
c
99 x(j)=-a(j)+b(j+1)
c
do i=j+1,n
a(i)=b(i)
enddo
c
dlt=cdsqrt(a(j+1)*a(j+1)-4.*a(j+2))
x(j+1)=(-a(j+1)+dlt)/2.
x(j+2)=(-a(j+1)-dlt)/2.[1]
c
50 continue
```

[1]These three lines may be moved outside the do-loop below 50 continue after setting j = n − 1 to avoid unnecessary computations.

```
c
c          Write the roots and corresponding values of the polynomial
c
write(*,100)
100 format(/,40x,4HRoot,30x,19HValue of Polynomial,/)
write(*,200)
200 format(8x,11HRoot Number,13x,4HReal,10x,9HImaginary,18x,4HReal,
&10x,9HImaginary,/)
c
do j=1,n
c
c          Evaluate polynomial for roots
poly=c(n)
do i=n-1,0,-1
poly=poly+c(i)*(x(j)+tht)**float(n-i)
enddo
c
write(*,300)j,x(j)+tht,poly
300 format(12x,1Hx,i2,8x,2f16.10,9x,2f16.10)
c
enddo
c
write(*,400)
400 format(/)
c
stop
end
```