An Exploration of Aberth's Method for Simultaneously Finding All of a Polynomial's Roots, featuring MATLAB implementation

Andrew Coleman

May 6, 2016

Abstract

We explore Aberth's iterative method for finding all zeros of polynomials with complex coefficients. Both Jacobi and Gauss-Seidel versions of the method are discussed, and the Jacobi version is derived in detail. We also carefully justify the procedure given by Aberth for selecting initial approximations. Graphs are presented in the appendices, which show the strengths of Aberth's method, and also demonstrate an issue that occurs if the initial approximations are selected such that they are symmetric about a line which the roots also happen to be symmetric about. This is crucial when approximating roots of polynomials with real coefficients, since such polynomials will always have roots which are symmetric about the real axis. Also provided in the appendices is a MATLAB implementation of both the Jacobi and Gauss-Seidel versions of Aberth's method, complete with selection of initial approximations.

1 Description of Problem

Our goal is to find approximations for all roots of a given polynomial of degree m with complex coefficients. We note that if the leading coefficient of the polynomial is not equal to one (i.e. if the polynomial is not monic), we may produce a monic polynomial with the same roots by dividing the entire polynomial by the leading coefficient. Thus, without loss of generality, we consider a monic polynomial of degree m,

$$P(z) = z^m + c_{m-1}z^{m-1} + \cdots + c_2z^2 + c_1z + c_0, \tag{1}$$

where all c_j are complex.

By the Fundamental Theorem of Algebra, there will exist exactly m (not necessarily distinct) roots of $P(z)$ in the complex plane. Thus, there exists $z_1^*, z_2^*, \ldots, z_m^* \in \mathbb{C}$ such that we may factor the polynomial as $P(z) = (z - z_1^*)(z - z_2^*)\cdots(z - z_m^*)$. Note that this implies that $P(z) = 0$ if and only if $z = z_j^*$ for some $j = 1, 2, \ldots, m$.

Given $P(z)$ in the form (1), our goal is to find close approximations to these roots.

2 Newton's Method

Before proceeding to Aberth's Method, we first recall Newton's method (see [6]) for finding a zero z^* of a differentiable function $f(z)$. This is an iterative method, and requires some initial approximation for the zero, z_0. With a known nth approximation, z^n and given differentiable f, Newton's method allows the next approximation z^{n+1} to be computed as

$$z^{n+1} = z^n - \frac{f(z^n)}{f'(z^n)}. \tag{2}$$

Formula (2) is based on the following considerations.

Since f is differentiable at z^n, f may be approximated by a linear function near z^n. In particular, we may approximate $f(z^{n+1})$, where z^{n+1} is near z^n, by

$$f(z^{n+1}) \approx f(z^n) + (z^{n+1} - z^n)f'(z^n).$$

Because we wish to find an approximation for some z^* such that $f(z^*) = 0$, we set $f(z^{n+1}) = 0$ in the approximation above. This gives

$$0 \approx f(z^n) + (z^{n+1} - z^n)f'(z^n). \tag{3}$$

Solving (3) for z^{n+1} produces the iterative step (2).

Whether Newton's method converges to a root or diverges is dependent upon the selection of the initial approximation, z^0. If for a given initial approximation z^0 the method converges to a simple root, we will have quadratic convergence; for multiple roots Newton's method yields a linear convergence.

3 Aberth's Method

A limitation of Newton's method is that it only approximates one zero at a time, which is determined by the initial approximation z_0 (assuming that the method converges for this choice of z_0). There is no easy way to choose z_0 which guarantees that it will converge to a particular root, so if we wish to find a root with a particular property, or we wish to find all roots, Newton's method will not be ideal.

Aberth's method is an iterative method which approximates all zeros of a polynomial simultaneously (see [1]). Without loss of generality (see Description of Problem), suppose we have a monic polynomial $P(z)$ of degree m, as in (1). For $n = 0, 1, 2, \ldots$ and each $j = 1, 2, \ldots, m$, let z_j^n be the nth approximation for the jth zero, z_j^*.

Using a Jacobi implementation of Aberth's method [3], we compute z_k^{n+1}, the $(n+1)$th approximation for the kth zero by using the formula

$$z_k^{n+1} = z_k^n - \frac{\mathcal{N}(z_k^n)}{1 - \mathcal{N}(z_k^n)\mathcal{A}_k(z_k^n)}. \tag{4}$$

where

$$\mathcal{N}(z_k^n) = \tfrac{P(z_k^n)}{P'(z_k^n)} \quad \text{and} \quad \mathcal{A}_k(z_k^n) = \sum_{\substack{j=0 \\ j\neq k}}^{m} \tfrac{1}{(z_k^n - z_j^n)}.$$

To use a Gauss-Seidel implementation of Aberth's method [4], we instead use

$$\mathcal{A}_k(z_k^n) = \sum_{j=0}^{k-1} \frac{1}{(z_k^n - z_j^{n+1})} + \sum_{j=k+1}^{m} \frac{1}{(z_k^n - z_j^n)}.$$

Suppose z_k^n is a root of the polynomial $P(z)$. Then $\mathcal{N}(z_k^n) = \frac{P(z_k^n)}{P'(z_k^n)} = 0$. When applying the iterative step (in either case), we find

$$z_k^{n+1} = z_k^n - \frac{\mathcal{N}(z_k^n)}{1 - \mathcal{N}(z_k^n)\mathcal{A}_k(z_k^n)} = z_k^n - \frac{0}{1-0} = z_k^n.$$

Thus, the roots of $P(z)$ are fixed points of the function which defines the iterative step of Aberth's method.

4 Derivation of Aberth's Method

We obtain formula (4) by applying Newton's method to functions

$$F_k(z) = \frac{P(z)}{\prod\limits_{\substack{j=0 \\ j\neq k}}^{m} (z - z_j)}. \tag{5}$$

where $z_1, z_2, \ldots, z_m \in \mathbb{C}$ are fixed.

By formula (2), given the nth approximation z^n, the iterative step for Newton's method applied to $F_k(z)$ is

$$z^{n+1} = z^n - \frac{F_k(z)}{F_k'(z)}. \tag{6}$$

Functions (5) are undefined for $z = z_j, j \neq k$; they are analytic and differentiable on $\mathbf{C} \setminus \{z_j \ : \ j \neq k\}$. If $z_j \approx z_j^*$ for all $j \neq k$, $F_k(z)$ can be approximated by

$$F_k(z) = \frac{\prod\limits_{j=1}^{m} (z - z_j^*)}{\prod\limits_{\substack{j=1 \\ j\neq k}}^{m} (z - z_j)} \approx \frac{\prod\limits_{j=1}^{m} (z - z_j^*)}{\prod\limits_{\substack{j=1 \\ j\neq k}}^{m} (z - z_j^*)} = z - z_k^*.$$

Next, the derivation of (4) is given.

First, we can write

$$\frac{F_k(z)}{F_k'(z)} = \left[\frac{F_k'(z)}{F_k(z)}\right]^{-1}.$$

From here, we notice that

$$\frac{F_k'(z)}{F_k(z)} = \frac{\mathrm{d}}{\mathrm{d}z} \ln |F_k(z)|,$$

and therefore

$$\left[\frac{F_k'(z)}{F_k(z)}\right]^{-1} = \left[\frac{\mathrm{d}}{\mathrm{d}z}\left(\ln|F_k(z)|\right)\right]^{-1}.$$

Now, substituting (5) for $F_k(z)$, we get

$$\frac{F_k(z)}{F_k'(z)} = \left[\frac{\mathrm{d}}{\mathrm{d}z}\left(\ln|F_k(z)|\right)\right]^{-1} = \left[\frac{\mathrm{d}}{\mathrm{d}z}\left(\ln\left|\frac{P(z)}{\prod_{\substack{j=0\\ j\neq k}}^{n}(z-z_j)}\right|\right)\right]^{-1}.$$

Recalling properties of logarithms allows us to rewrite this as

$$\frac{F_k(z)}{F_k'(z)} = \left[\frac{\mathrm{d}}{\mathrm{d}z}\left(\ln|P(z)| - \sum_{\substack{j=0\\ j\neq k}}^{n}\ln|z-z_j|\right)\right]^{-1}.$$

Differentiating this produces

$$\frac{F_k(z)}{F_k'(z)} = \left[\frac{P'(z)}{P(z)} - \sum_{\substack{j=0\\ j\neq k}}^{n}\frac{1}{(z-z_j)}\right]^{-1} = \frac{1}{\frac{P'(z)}{P(z)} - \sum_{\substack{j=0\\ j\neq k}}^{n}\frac{1}{(z-z_j)}}.$$

Now, we multiply this by $1 = \frac{\left(\frac{P(z)}{P'(z)}\right)}{\left(\frac{P(z)}{P'(z)}\right)}$, which gives us

$$\frac{F_k(z)}{F_k'(z)} = \frac{\left(\frac{P(z)}{P'(z)}\right)}{1 - \frac{P(z)}{P'(z)}\sum_{\substack{j=0\\ j\neq k}}^{n}\frac{1}{(z-z_j)}}$$

which we may write more succinctly as

$$\frac{F_k(z)}{F_k'(z)} = \frac{\mathcal{N}(z)}{1-\mathcal{N}(z)\mathcal{A}_k(z)} \quad \text{where } \mathcal{N}(z) = \frac{P(z)}{P'(z)} \text{ and } \mathcal{A}_k(z) = \sum_{\substack{j=0\\ j\neq k}}^{n}\frac{1}{(z-z_j)}. \tag{7}$$

Now, by substituting (7) into (6), and using z_k^n, the nth approximation of the kth root of $P(z)$, as our current approximation, we produce the Jacobi version of (4).

5 Selection of Initial Approximations

We have detailed the iterative step in Aberth's method, but the iterative step assumes that we are given approximations for each root. To apply Aberth's method in practice, it is necessary to select an initial approximation for each root. To do this, we find a disk $D(\zeta, r) = \{z \in \mathbb{C} \,:\, |z - \zeta| < r\}$ in the complex plane such that all roots of the polynomial are contained within the disk, and then select points arranged evenly around the boundary of the disk to use as our initial approximations [1]. Since the set of roots of any polynomial is finite, this set is bounded in the complex plane. So, such a disk will always exist.

Suppose, as before, that we have a monic polynomial $P(z) = z^m + c_{m-1}z^{m-1} + \cdots + c_2z^2 + c_1z + c_0$ with roots $z_1^*, z_2^*, \ldots, z_m^*$ (not necessarily distinct). Then we may write $P(z) = (z - z_1^*)(z - z_2^*)\cdots(z - z_m^*)$. By expanding, we find

$$\begin{aligned} P(z) = &\quad z^m + c_{m-1}z^{m-1} + \cdots + c_2z^2 + c_1z + c_0 \\ = &\quad (z - z_1^*)(z - z_2^*)\cdots(z - z_m^*) \\ = &\quad z^m + z^{m-1}\left(\sum_{k=1}^{m} -z_k^*\right) + z^{m-2}\left(\sum_{k=1}^{m}\sum_{j=k+1}^{m}(z_k^*)(z_j^*)\right) + \cdots + (-1)^m \prod_{k=1}^{m}(z_k^*). \end{aligned}$$

And so we see that

$$c_{m-1} = \sum_{k=1}^{m} -z_k^*$$

And thus, $\zeta_0 = -\frac{c_{m-1}}{m}$ is the average of all of the roots of $P(z)$ (weighted by multiplicity) [1]. This will act as the center of our disk.

Setting $z = w + \zeta_0 = w - \frac{c_{m-1}}{m}$, we can find coefficients $c'_{m-1}, c'_{m-2}, \ldots, c'_1, c'_0$, such that

$$P(z) = \hat{P}(w) = w^m + c'_{m-2}w^{m-2} + \cdots + c'_1w + c'_0.\text{(see [1])}$$

These coefficients are given by

$$c'_{m-k} = \sum_{j=0}^{k} \binom{m-j}{k-j}(c_{m-j})\left(-\frac{c_{m-1}}{m}\right)^{k-j} \quad \text{for} \quad k = 1, 2, \ldots, m.$$

Note $c'_{m-1} = 0$, which means that the average of all the roots of $\hat{P}(w)$ (weighted by multiplicity) is zero.

At this point, we make use of Rouché's Theorem [2, pp. 341–342].

Rouché's Theorem. *Let $D \subset \mathbb{C}$ be a Jordan domain (i.e., D is a simply connected, bounded, open subset of $\mathbb{C}$ such that its boundary ∂D is a simple closed curve). Suppose f and g be complex-valued functions which are analytic on some open set containing $\overline{D} = D \cup \partial D$. Then if $|f(z)| - |g(z)| > 0$ for all $z \in \partial D$, the functions f and $f + g$ have the same number of zeros in D (counted with multiplicity).*

We consider polynomials $f(w) = w^m$ and $g(w) = c'_{m-2}w^{m-2} + \cdots + c'_1 w + c'_0$, noting that since they are polynomials, both f and g are entire functions. The function $f(w)$ has a zero at $w = 0$ with multiplicity m, and no other zeros in the complex plane. And so for any disk $D = D(0, r)$, $r > 0$, f has m zeroes in D (counted with multiplicity). By noting that $\partial D(0, r) = \{w \in \mathbb{C} \, : \, |w| = r\}$ for any disk $D(0, r)$, and applying Rouché's Theorem to f and g, we see to show that $(f + g)(w) = \hat{P}(w)$ has exactly m zeros in the disk $D(0, r)$, it is sufficient to show that $|w| = r$ implies $|f(w)| - |g(w)| > 0$.

By the Triangle Inequality,

$$|g(w)| = \left|c'_{m-2}w^{m-2} + \cdots + c'_1 w + c'_0\right| \leq \left|c'_{m-2}w^{m-2}\right| + \cdots + \left|c'_1 w\right| + \left|c'_0\right|. \tag{8}$$

And by properties of complex moduli,

$$\left|c'_{m-2}w^{m-2}\right| + \cdots + \left|c'_1 w\right| + \left|c'_0\right| = \left|c'_{m-2}\right| |w|^{m-2} + \cdots + \left|c'_1\right| |w| + \left|c'_0\right|. \tag{9}$$

Combining (8) and (9) produces

$$|g(w)| \leq \left|c'_{m-2}\right| |w|^{m-2} + \cdots + \left|c'_1\right| |w| + \left|c'_0\right|.$$

Thus,

$$|f(w)| - |g(w)| \geq |w|^m - \left(\left|c'_{m-2}\right| |w|^{m-2} + \cdots + \left|c'_1\right| |w| + \left|c'_0\right| \right). \tag{10}$$

Now, for all $w \in \mathbb{C}$, $|w|$ is a non-negative real number. Setting $x = |w|$, we may consider the real-valued polynomial S defined on the non-negative reals by

$$S(x) = x^m - \left(\left|c'_{m-2}\right| x^{m-2} + \cdots + \left|c'_1\right| x + \left|c'_0\right| \right).$$

Since it is a polynomial, S is continuous on the non-negative reals. And for $x = 0$, $S(0) = -\left|c'_0\right| \leq 0$.

Now consider $\lim_{x\to\infty} \frac{x^m}{|c'_{m-2}|x^{m-2}+\cdots+|c'_1|x+|c'_0|}$.

By L'Hopital's Rule,

$$\lim_{x\to\infty} \frac{x^m}{\left|c'_{m-2}\right| x^{m-2} + \cdots + |c'_1| x + |c'_0|} = \lim_{x\to\infty} \frac{m(m-1)}{2\left|c'_{m-2}\right|} x^2 = +\infty.$$

Thus, there exists $R > 0$ such that $r > R$ implies that $S(r) > 0$.

Assuming that $\left|c'_j\right| \neq 0$ for at least one $j = 0, 1, \ldots, m-2$, we may use Descartes' Sign Rule [5] to see that S must have exactly one positive, real root. And so there exists a unique $R > 0$ such that $S(R) = 0$ and $r > R$ implies that $S(r) > 0$. And because $S(0) \leq 0$, this R will also be such that $S(r) < 0$ for $0 < r < R$.

Note that if $\left|c'_{m-2}\right| = \cdots = |c'_1| = |c'_0| = 0$, we have that $\hat{P}(w) = w^m$. This implies that $P(z) = \left(z - \frac{c_{m-1}}{m}\right)^m$. In this case, the roots of P are given by $z_1^* = z_2^* = \cdots = z_m^* = \frac{c_{m-1}}{m}$, and there is no need to perform Aberth's

method. We also remark that if $P(c_{m-1}/m) \neq 0$, we know that $c'_j \neq 0$ for at least one $j = 0, 1, \ldots, m-2$ (note that the converse is not true, $P(z) = z^3 - z = z(z+1)(z-1)$ provides a quick counterexample).

For our purposes, we do not need to find this unique R. It will suffice to instead use some r_0 which roughly approximates R [1]. The MATLAB implementation provided in the appendices first finds a positive real number x_0 such that $S'(x_0) > 0$. Then, it uses Newton's method, with initial approximation x_0 to find an approximation for R. Finally, the program adds 1 to this approximation until it finds a value r_0 such that $S(r_0) > 0$. This gives r_0 near R such that $r_0 > R$, and thus we will have that $S(r) > 0$ for all $r > r_0$.

Suppose now that we have found such an r_0.

Recall that by our definition of S,

$$S(|w|) = |w|^m - \left(\left|c'_{m-2}\right| |w|^{m-2} + \cdots + \left|c'_1\right| |w| + \left|c'_0\right| \right),$$

and so by (10), we have that $S(|w|) \leq |f(w)| - |g(w)|$. Now, suppose $|w| = r_0$. Then

$$0 < S(r_0) = S(|w|) \leq |f(w)| - |g(w)| .$$

We have already shown that this is sufficient to prove that all zeros w^* of $\hat{P}(w)$ must be such that $w^* \in D(0, r_0)$ (i.e., such that $|w^*| \leq r$).

Thus, since $w = z + \zeta_0 = z + \frac{c_{m-1}}{m}$, all zeroes z^* of $P(z)$ must be such that $z^* \in D(\zeta_0, r_0)$ (i.e., such that $\left|z^* + \frac{c_{m-1}}{m}\right| \leq r$).

For initial approximations, use $z_k^0 = \zeta_0 + r_0 e^{i((2\pi k/m)+\alpha)}$, where α is a real constant chosen to avoid selecting points symmetric about a line which the roots of P are also symmetric about. If P is a polynomial with real coefficients, and $x+iy$ is a non-real complex root of P, then that root's complex conjugate, $x-iy$, is also a root of P. Thus, a polynomial P with real coefficients will have roots symmetric about the real axis. To avoid selecting initial approximations symmetric about the real axis, we may use $\alpha = \frac{\pi}{2m}$. If using a Gauss-Seidel version of Aberth's method, this symmetry will merely reduce the efficiency with which the approximations approach the roots. However, if using a Jacobi version of Aberth's method, such symmetry may cause approximations to be unable to converge to a root.

This is because the Jacobi version updates all approximations at once. So, it is possible for the Jacobi iteration to preserve symmetry in a way that the Gauss-Seidel version does not. (If all approximations are symmetric about a line, performing the Gauss-Seidel step on one of the points will change the location of that point. Unless that point was already lying on the line of symmetry, and the next approximation also lies on this line, this will disrupt the symmetry.) In the Jacobi version, eventually aggregation of rounding errors results in a disruption to the symmetry, which may allow the approximations to converge [1]. Often once the symmetry is disrupted, convergence happens quite quickly, because some of the other approximations are already nearby roots, which causes $F_k(z)$ to behave more closely to a linear function, and thus Newton's method becomes more effective at approximating zeros for $F_k(z)$.

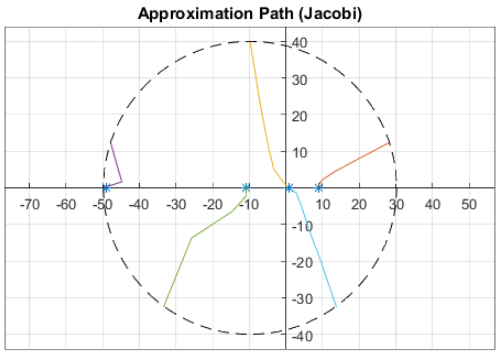
Appendix A Figures

Consider the polynomial

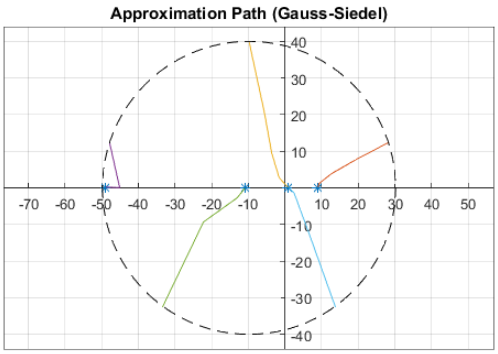
$$P_1(z) = z^5 + 49z^4 - 102z^3 - 4798z^2 + 9701z - 4851 = (z-1)^2(z-9)(z+11)(z+49),$$

which has a multiple root ($z_1^* = 1$, which has multiplicity 2).

Using $\alpha = \pi/2m$, after 5 iterations, we have



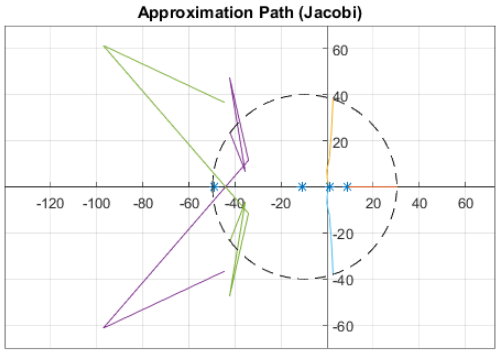
(a) Maximum Error (Jacobi): 0.748481367



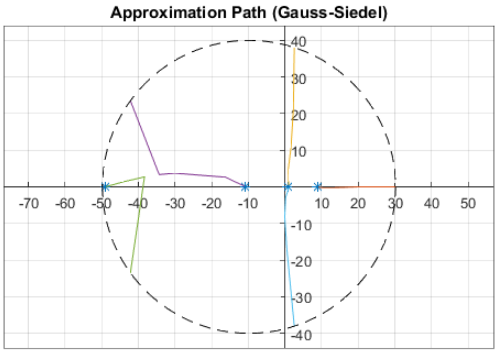
(b) Maximum Error (Gauss-Seidel): 0.242536420

Figure 1: P_1, $\alpha = \pi/2m$, 5 iterations

Using $\alpha = 0$, after 5 iterations, we have



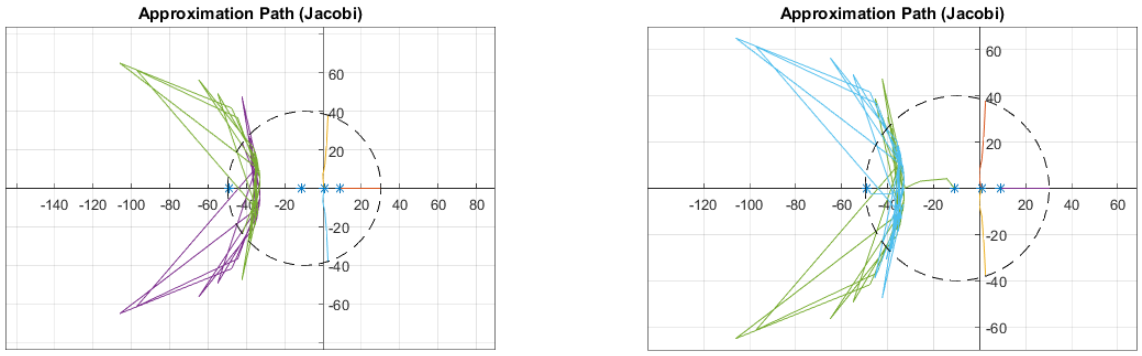
(a) Maximum Error (Jacobi): 49.6952314



(b) Maximum Error (Gauss-Seidel): 0.401149096

Figure 2: P_1, $\alpha = 0$, 5 iterations

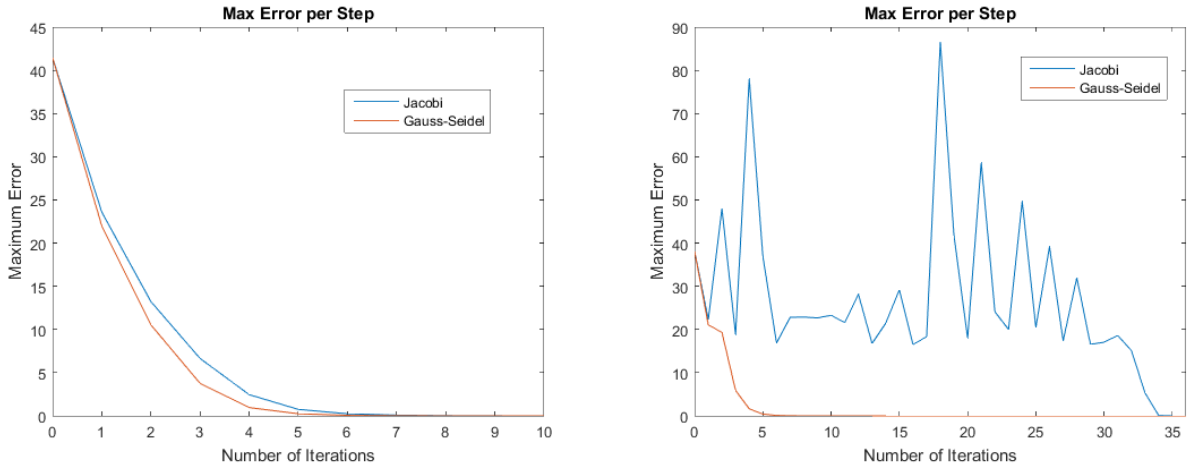
After 25 and 35 iterations, we have



(a) 25 iterations, Maximum Error (Jacobi): 27.2829933 (b) 35 iterations, Maximum Error (Jacobi): 1.6549E-07

Figure 3: P_1, $\alpha = 0$. Even though the Jacobi verson of Aberth's method meanders, it does eventually catch up to the Gauss-Seidel version, due to aggregation of error disrupting the symmetry. [1]

We can see from the graph that using $\alpha = 0$ for polynomials with real coefficients leads to erratic behavior, where the marginal decrease in error from one more iteration may be negative (i.e., one more iteration may cause error to increase). This is made even more clear by the following graphs, which plot the Maximum Error for a given iteration.



(a) $\alpha = \pi/2m$ (b) $\alpha = 0$

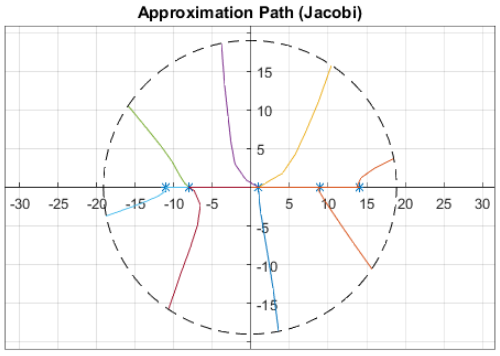
Figure 4: Maximum Error vs Number of Iterations Performed

Now, consider the polynomial

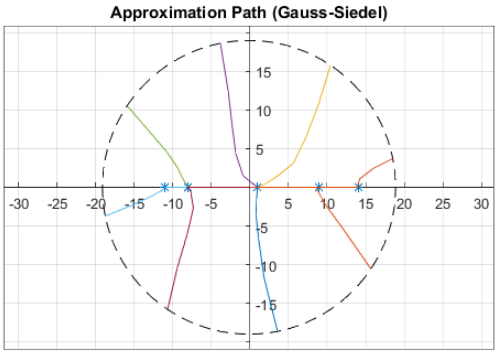
$P_2(z) = z^8+z^7-264z^6-638z^5+17521z^4+42573z^3-222554z^2+252064z-88704 = (z-1)^3(z+8)^2(z-9)(x+11)(z-14)$,

which has a root of multiplicity 3 and a root of multiplicity 2.

Using $\alpha = \pi/2m$, after 35 iterations we have



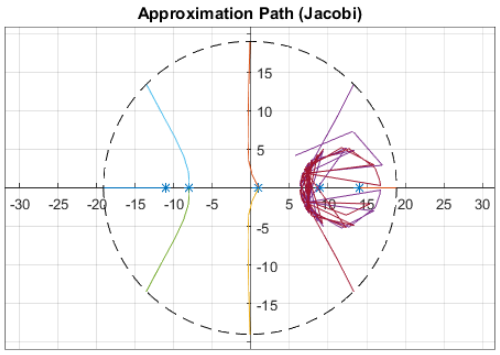
(a) Maximum Error (Jacobi): 0.00090785051



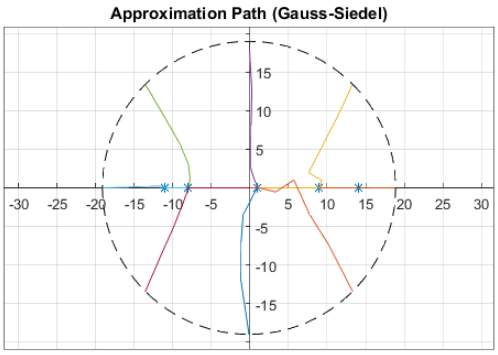
(b) Maximum Error (Gauss-Seidel): 0.000481606797

Figure 5: P_2, $\alpha = \pi/2m$, 35 iterations

Using $\alpha = 0$, after 35 iterations we have



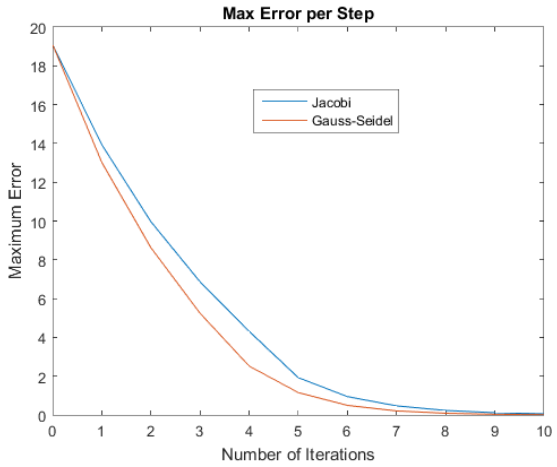
(a) Maximum Error (Jacobi): 25



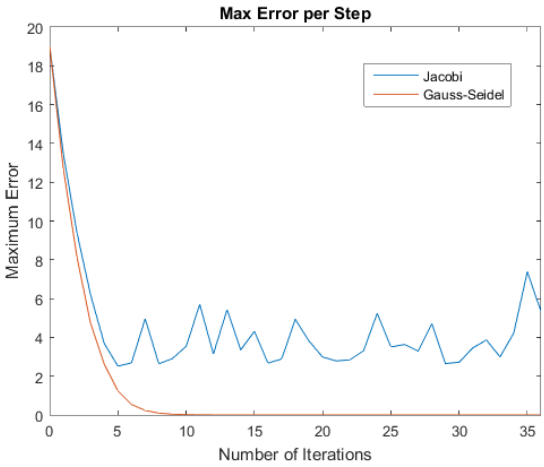
(b) Maximum Error (Gauss-Seidel): 0.000537438169

Figure 6: P_2, $\alpha = 0$, 35 iterations

Presented below are the Maximum error graphs for $\alpha = \pi/2m$ and $\alpha = 0$.



(a) $\alpha = \pi/2m$



(b) $\alpha = 0$

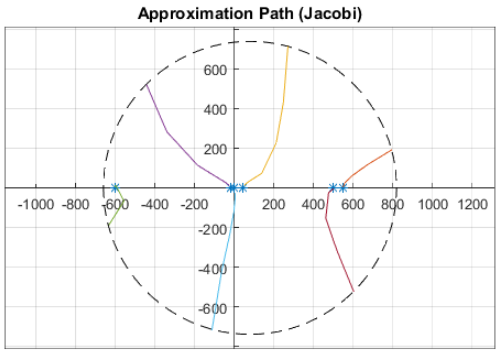
Figure 7: Maximum Error vs Number of Iterations Performed

We now consider the polynomial

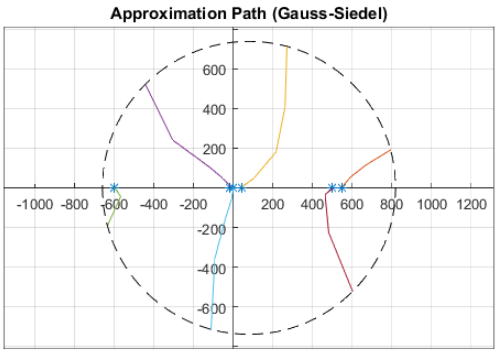
$$\begin{aligned} P_3(z) = &\quad z^6 - 476z^5 - 344175z^4 + 174624650z^3 - 3979780000z^2 - 144694500000z + 148500000000 \\ = &\quad (z + 600)(z - 550)(z - 500)(z - 45)(z + 20)(z - 1), \end{aligned}$$

which has roots with small moduli, as well as roots with large moduli.

Using $\alpha = \pi/2m$, after 5 iterations, we have



(a) Maximum Error (Jacobi): 23.5186339

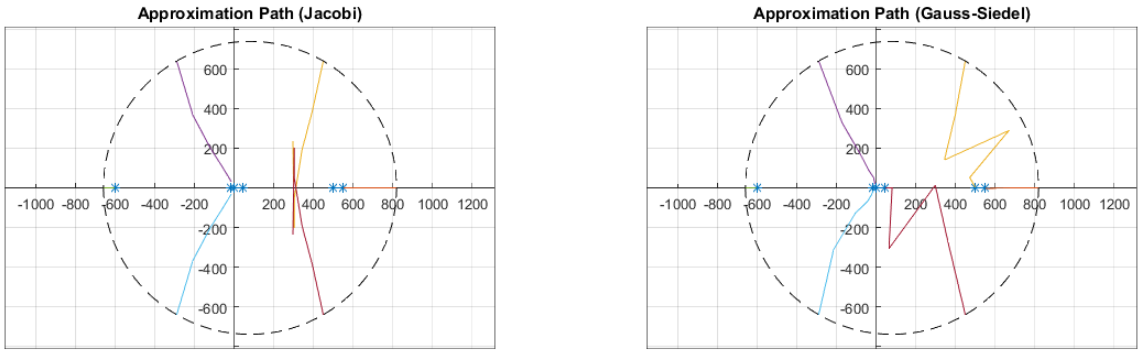


(b) Maximum Error (Gauss-Seidel): 3.68953574

Figure 8: P_3, $\alpha = \pi/2m$, 5 iterations

The minimum error for the initial approximations was greater than 748. Using the Gauss-Seidel version of Aberth's method, after 5 iterations, we have a maximum error of less than 4.

As before, we now attempt to use $\alpha = 0$,



(a) Maximum Error (Jacobi): 342.79217 (b) Maximum Error (Gauss-Seidel): 22.6738323

Figure 9: P_3, $\alpha = 0$, 5 iterations

Continuing for more iterations produces even more aberrant behavior from the Jacobi version.

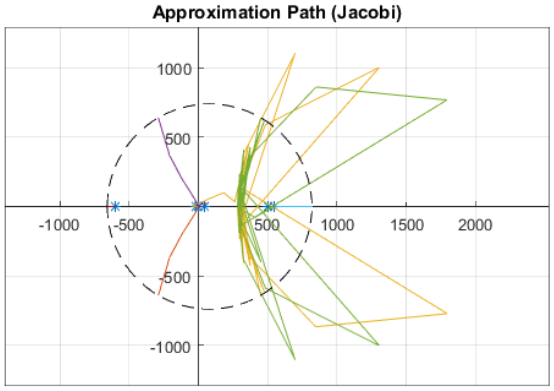
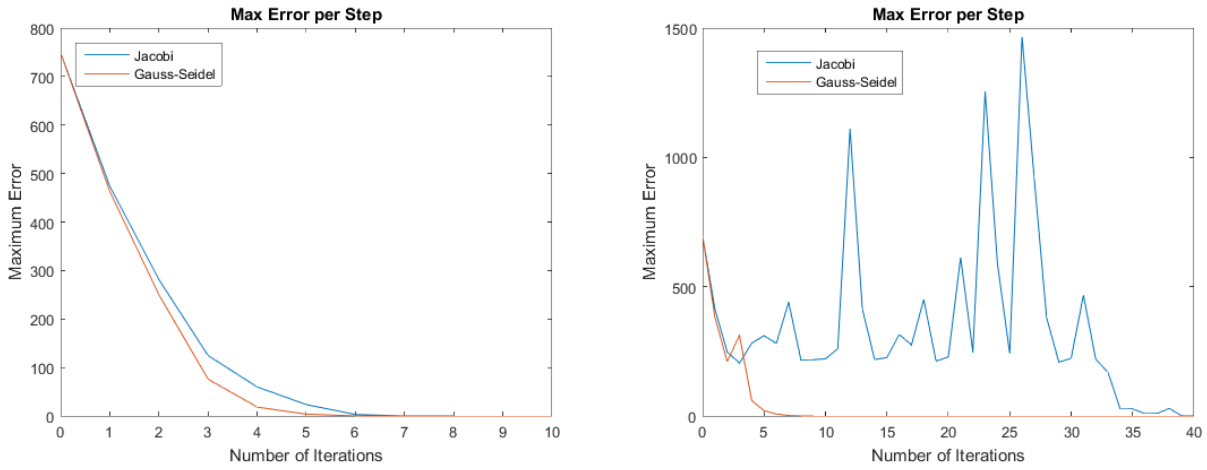


Figure 10: P_3, $\alpha = 0$, 40 iterations

Maximum error for the Gauss-Seidel version is not entirely predictable when using $\alpha = 0$, but it is not nearly as unpredictable as the Jacobi version.



(a) $\alpha = \pi/2m$ (b) $\alpha = 0$

Figure 11: Maximum Error vs Number of Iterations Performed

Appendix B MATLAB file: AberthJacobiWrapper.m

```
function [ Z ] = AberthJacobiWrapper( P , Iterations)
%AberthJacobiWrapper Calls various functions to perform Iterations
%iterations of the Jacobi version of Aberth's method on polynomial P.

StopCriteria = [Iterations, 0];

[Q1 , ZeroRootMultiplicity] = ZeroRootMultFinder( P );

Q = MonicPolyMaker(Q1);

Z0 = InitApprox(Q);

if Z0(1) == Z0(2)
   Z = Z0;
   return
end

Z1 = AberthMethodJacobi(Q,Z0,StopCriteria);

Z = zeros(size(Z1)+ZeroRootMultiplicity);
[~,I] = sort(abs(Z1));
Z(ZeroRootMultiplicity+1:end) = Z1(I);

end
```

Appendix C MATLAB file: AberthGaussSeidelWrapper.m

```
function [ Z ] = AberthGaussSeidelWrapper( P , Iterations)
%AberthGaussSeidelWrapper calls various functions to perform Iterations
%iterations of the Gauss-Seidel version of Aberth's method on polynomial P

StopCriteria = [Iterations, 0];

[Q1 , ZeroRootMultiplicity] = ZeroRootMultFinder( P );

Q = MonicPolyMaker(Q1);

Z0 = InitApprox(Q);

if Z0(1) == Z0(2)
   Z = Z0;
   return
end

Z1 = AberthMethodGaussSeidel(Q,Z0,StopCriteria);

Z = zeros(size(Z1)+ZeroRootMultiplicity);
[~,I] = sort(abs(Z1));
Z(ZeroRootMultiplicity+1:end) = Z1(I);
end
```

Appendix D MATLAB file: Script.m

```
clear all
format longG
close all

% Number of iterations to run if IterOrEps = 1
% Maximum number of iterations to run if IterOrEps = 0
% If IterOrEps = 0, may run fewer than Iterations iterations
Iterations = 35;

% Set to 0 to use alpha = 0.
% Set to -1 to use alpha = pi/2m.
Alpha = 0;

% How small MaxError must be for while loop in RootFinderAlgo to terminate
Epsilon = 10^(-3); % Only used if IterOrEps = 0

% Use IterOrEps = 1 to run RootFinderAlgo for Iterations iterations
% Use IterOrEps = 0 to run RootFingerAlgo until MaxError < Epsilon
IterOrEps = 0;

StopCriteria = [Iterations, Epsilon];

%% Randomly Chooses Roots %%%%%%%%%%%%%%%%%%%%%%%%%

n = randi(10,1,1)+2; % Randomly decide degree of polynomial, from 3 to 12
np1 = n+1;

k=5; % Change this to change how many orders of magnitude roots may be from zero

% Generates random complex numbers to use as roots
Roots = (10^k)*transpose(complex((-1)^(randi(2,1,1))*rand(n,1),(-1)^(randi(2,1,1))*rand(n,1))
    );

Multiplicity=2; %Change this to create a root with multiplicity Multiplicity
for i=1:Multiplicity
    Roots(i) = Roots(1);
end

%% If desired, manually set roots here

%P_1
%Roots = [1 1 9 -11 -49];

%P_2
%Roots = [1 1 1 -8 -8 9 -11 14];

%P_3
%Roots = [1 -20 45 500 550 -600];

%Roots = [1 2 3 4 5];

%% Arranges roots in order by moduli, required for error calculation

[~,I] = sort(abs(Roots));
Roots = Roots(I);
clear I

%% Generates polynomial from Roots
P = poly(Roots);


```

```
61
62 %% Prints Roots
63 fprintf('Roots of polynomial p\n')
64 disp(transpose(Roots));
65
66 %% Change polynomial p to be monic and have no roots at zero
67
68 % Converts p to a polynomial which does not have zero as a root, while
69 % keeping track of the multiplicity of zero as a root
70 [Q , m] = ZeroRootMultFinder( P );
71
72 % Changes polynomial Q into a monic polynomial
73 Q = MonicPolyMaker(Q);
74
75 %% Finds initial approximations for non-zero roots of p
76
77 if Alpha == -1 % Sets Alpha to pi/2m if Alpha was set to -1 earlier
78     Alpha = pi/(2*(size(Q,2)-1));
79 end
80
81 [Z0 , center , R] = InitApprox(Q,Alpha);
82
83 %% Set alternate initial approximations here
84 %Z0 = [];
85
86 %% Performs iterative steps to get better approximations
87 [Z1J,CountJ,Z1nJ] = AberthMethodJacobi(Q,Z0,StopCriteria,IterOrEps);
88 [Z1GS,CountGS,Z1nGS] = AberthMethodGaussSeidel(Q,Z0,StopCriteria,IterOrEps);
89
90 %% Replaces zero roots
91 ZJ = zeros(size(Z1J)+m);
92 [~,I] = sort(abs(Z1J));
93 ZJ(m+1:end) = Z1J(I);
94
95 ZGS = zeros(size(Z1GS)+m);
96 [~,I] = sort(abs(Z1GS));
97 ZGS(m+1:end) = Z1GS(I);
98
99 %Rearranges approximations used in Jacobi version
100 ZnJ = zeros(size(Z1nJ,1),size(Z1nJ,2)+m);
101 ZnJ(m+1:end,:) = Z1nJ(:,I);
102
103 %Rearranges approximations used in Gauss-Seidel version
104 ZnGS = zeros(size(Z1nGS,1),size(Z1nGS,2)+m);
105 ZnGS(m+1:end,:) = Z1nGS(:,I);
106
107 % Displays largest difference between initial approximations and actual
108 % value
109 fprintf('Initial Approximations all within %.5G of a root.\n', MaxDiffFinder(Z0,Roots))
110
111 % Displays largest difference between approximations and actual value
112 [MaxErrorJ,MaxErrorIndexJ] = max(abs(ZJ-Roots));
113 [MaxErrorGS,MaxErrorIndexGS] = max(abs(ZGS-Roots));
114
115 disp(CountJ)
116 disp(CountGS)
117
118 fprintf('       Maximum Error (Jacobi): %.9G\n' , MaxErrorJ)
119 fprintf(' Maximum Error (Gauss-Seidel): %.9G\n' , MaxErrorGS)
120
121 CreateFigures(Z0, Z1J, ZnJ, Z1GS, ZnGS, Roots, center , R , CountJ, CountGS , m);
```

Appendix E MATLAB file: ZeroRootMultFinder.m

```
function [ Q , m ] = ZeroRootMultFinder( P )
%ZeroRootMultFinder Finds m, multiplicity the polynomial p has zero
%as a root, and returns a polynomial q such that (z^m)*q(z) = p(z)

m=0;
while P(end-m)==0
    m = m+1;
end
Q = P(1:end-m);
end
```

Appendix F MATLAB file: MonicPolyMaker.m

```
function [ Q ] = MonicPolyMaker( P )
%MonicPolyMaker Takes input polynomial P and outputs a monic polynomial
%with the same zeros

n = size(P,2);
index = 1;

 while P(index) == 0 && index <= n;
     index = index + 1;
 end

Q = P(index:end)/P(index);
end
```

Appendix G MATLAB file: InitApprox.m

```
function [ Z0 , c , R ] = InitApprox( P , Alpha)
%InitApprox Finds initial approximations for polynomial zeros
% REQUIRES RFinderNewton

[c , R] = RFinderNewton(P);

if R == -1
   Z0 = c*ones(1,np1);
    return
end

np1 = size(P,2);

n = np1 - 1;

nm1 = n - 1;

if ~exist('a','var')
    Alpha = pi/(2*n);
end

Z0(1:n) = c + R*exp(((2*pi/n)*(0:nm1) + Alpha)*1i);

end
```

Appendix H MATLAB file: RFinderNewton.m

```
function [ c , R ] = RFinderNewton( P )
%RFinderNewton Uses a theorem by Cauchy to find a value R such that all zeros of
%the polynomial p lie within the closed disk of radius R centered at
%-c_{n-1}/n


np1 = size(P,2);
n = np1 - 1;
c = - P(2)/n;
Q = zeros(1,np1);
Q(1) = 1;

for k = 3:np1
    for l = k-1:-1:0
        Q(k) = Q(k) + P(k-l)*(c ^ (l))*nchoosek( n - (k-1-l), l);
    end
end

S = zeros(1,np1);
S(1) = 1;
S(3:end) = -abs(Q(3:end));

if S(2:end) == zeros(1,n)
    R = -1; % Used to pass program state: roots all equal to c
    return
else

    R = 2;
    t = polyder(S);

    while polyval(t,R) < 0
        R = 2*R;
    end

    while polyval(S,R)/polyval(t,R) > 0.00001
        R = R - polyval(S,R)/polyval(t,R);
    end

    while polyval(S,R) < 0
        R = R + 1;
    end
end
end
```

Appendix I MATLAB file: AberthMethodJacobi.m

```
function [ Z , Count , Zn1] = AberthMethodJacobi( P , Z , StopCriteria, IterOrEps,Roots)
%AberthMethodJacobi Performs the Jacobi version of iterative step of Aberth's
%method until the StopCriteria is met

if ~exist('Roots','var')
   IterOrEps = 1;
   Roots = 0;
end

if ~exist('IterOrEps','var')
    IterOrEps = 1;
end

if StopCriteria(1) < 1 || mod(StopCriteria(1),1) ~= 0
   fprintf('Error: StopCriteria(1) must be a positive integer.\n')
   return
end

n = size(Z,2);

if size(P,2) == 2
    Z = -P(2)/P(1);
    return
end

Q = polyder(P);
A = zeros(1,n);

% Initializes values so that while loop will run at least once
MaxError = realmax;
Count = 1;
CountStop = StopCriteria(1);
Zn1 = zeros(CountStop+1,n);
Zn1(1,:) = Z;

if IterOrEps == 1;
    ErrorStop = 0;
elseif IterOrEps == 0;
    ErrorStop = StopCriteria(2);
end

while Count <= CountStop && MaxError > ErrorStop

    for j = 1:n
        jm1 = j-1;
        A(j) = sum(1./(Z(j) - Z(1:jm1))) + sum(1./(Z(j) - Z(j+1:n)));
    end

    N = polyval(P,Z)./polyval(Q,Z);
    deltaZ =  - N./(1 - N.*A);
    Z = Z + deltaZ;
    Count = Count + 1;
    Zn1(Count,:) = Z;
    MaxError = MaxDiffFinder(Z,Roots);

end

Zn = Zn1(1:Count,:);
Count = Count - 1;

end
```

Appendix J MATLAB file: AberthMethodGaussSeidel.m

```
function [ Z , Count , Zn1] = AberthMethodGaussSeidel( P , Z , StopCriteria, IterOrEps,Roots)
%AberthMethodGaussSeidel Performs the GaussSeidel version of the iterative
%step of Aberths method until the StopCriteria is met


if ~exist('IterOrEps','var')
    IterOrEps = 1;
end

if StopCriteria(1) < 1 || mod(StopCriteria(1),1) ~= 0
   fprintf('Error: StopCriteria(1) must be a positive integer.\n')
   return
end

n = size(Z,2);

if size(P,2) == 2
    Z = -P(2)/P(1);
    return
end

Q = polyder(P);
A = zeros(1,n);
MaxError = realmax;
Count = 1;
CountStop = StopCriteria(1);

Zn1 = zeros(CountStop+1,n);
Zn1(1,:) = Z;

if IterOrEps == 1;
    ErrorStop = 0;
elseif IterOrEps == 0;
    ErrorStop = StopCriteria(2);
end

while Count <= CountStop && MaxError > ErrorStop

    for j = 1:n
        jm1 = j-1;
        A(j) = sum(1./(Z(j) - Z(1:jm1))) + sum(1./(Z(j) - Z(j+1:n)));
        N = polyval(P,Z(j))/polyval(Q,Z(j));
        Z(j) = Z(j) - N/(1 - N*A(j));
    end

    Count = Count + 1;
    Zn1(Count,:) = Z;
    MaxError = MaxDiffFinder(Z,Roots);
end

Zn = Zn1(1:Count,:);
Count = Count - 1;

end
```

Appendix K MATLAB file: MaxDiffFinder.m

```
function [ MaxDiff , Diffs, SRoots , SZn] = MaxDiffFinder( Zn , Roots )
%MaxDiffFinder For each iteration, finds the approximation farthest from
%any root, and returns the distance from that approximation to the nearest
%root (i.e., for each iteration, gives a real number R such that
%all approximations are within distance R of a root of the polynomial)

[~,I] = sort(abs(Roots));
Roots = Roots(I);
[Iter , n] = size(Zn);
[~,I] = sort(abs(Zn(Iter,:)));
Zn = Zn(:,I);
Diffs = zeros(n,n,Iter);

SZn = zeros(n,n,Iter);
for index = 1:n
    SRoots(:,index,1) = Roots(1,:);
end

for index = 1:Iter
   for index2 = 1:n
        SZn(index2,:,index) = Zn(index,:);
   end
end

for index = 1:Iter
   Diffs(:,:,index) = abs(SZn(:,:,index) - SRoots);
end

MaxDiff = reshape(max(min(Diffs,[],1),[],2),[],1);

end
```

Appendix L MATLAB file: CreateFigures.m

```
function [  ] = CreateFigures( Z0 , ZJ , ZnJ, ZGS, ZnGS, Roots, c , R , CountJ, CountGS , m)
%CreateFigures Creates various figures depicting how the algorithm performed

close all

scrsz = get(groot,'ScreenSize');

scale = 1.1; % Adjust this to zoom in and out

XRange = [real(c)-scale*R , real(c)+scale*R];
YRange = [imag(c)-scale*R , imag(c)+scale*R];

figure('name','Initial Approximations vs Roots','Position',[0.001*scrsz(3)/3 1.73*scrsz(4)/3
    scrsz(3)/3 scrsz(4)/3])
plot(real(Roots),imag(Roots),'*',real(Z0),imag(Z0),'d')
title('Initial Approximations vs Roots')
l1 = legend('Initial Approxmiations','Roots');
l1.Location = 'southoutside';

ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
ax.XLim = XRange;
ax.YLim = YRange;
axis equal;
ax.XGrid = 'on';
ax.YGrid = 'on';
Tick = min(ax.XTick(2)-ax.XTick(1),ax.YTick(2)-ax.YTick(1));
LowHighX = 2*floor(abs(XRange/Tick)).*(XRange./abs(XRange))*Tick;
LowHighY = 2*floor(abs(YRange/Tick)).*(YRange./abs(YRange))*Tick;
ax.XTick = LowHighX(1):Tick:LowHighX(2);
ax.YTick = LowHighY(1):Tick:LowHighY(2);

rectangle('Position',[real(c) - R, imag(c) - R, R*2, R*2],'Curvature',[1,1],'LineStyle','--')
%viscircles([real(c),imag(c)],R)

Title3 = sprintf('Approximated Roots after %i Iterations (Jacobi)',CountJ);

figure('name','Approximated Roots vs Roots','Position',[1.8*scrsz(3)/3 1.51*scrsz(4)/3 scrsz
    (3)/2.5 scrsz(4)/2.5])
plot(real(Roots),imag(Roots),'*',real(ZJ),imag(ZJ),'o')
title(Title3)
l2 = legend('Approximated Roots','Roots');
l2.Location = 'southoutside';

ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
ax.XLim = XRange;
ax.YLim = YRange;
axis equal;
ax.XGrid = 'on';
ax.YGrid = 'on';
Tick = min(ax.XTick(2)-ax.XTick(1),ax.YTick(2)-ax.YTick(1));
LowHighX = 2*floor(abs(XRange/Tick)).*(XRange./abs(XRange))*Tick;
LowHighY = 2*floor(abs(YRange/Tick)).*(YRange./abs(YRange))*Tick;
ax.XTick = LowHighX(1):Tick:LowHighX(2);
ax.YTick = LowHighY(1):Tick:LowHighY(2);

rectangle('Position',[real(c) - R, imag(c) - R, R*2, R*2],'Curvature',[1,1],'LineStyle','--')
%viscircles([real(c),imag(c)],R)
```

```
60
61 %% Plots Approximation Paths
62 figure('name','Approximation Path (Jacobi)','Position',[1.8*scrsz(3)/3 0.02*scrsz(4)/3 scrsz
       (3)/2.5 scrsz(4)/2.5])
63 plot(real(Roots),imag(Roots),'*',real(ZnJ),imag(ZnJ),'-')
64 title('Approximation Path (Jacobi)')
65 %l3 = legend('Approximated Roots','Roots');
66 %l3.Location = 'southoutside';
67
68 ax = gca;
69 ax.XAxisLocation = 'origin';
70 ax.YAxisLocation = 'origin';
71 ax.XLim = XRange;
72 ax.YLim = YRange;
73 axis equal;
74 ax.XGrid = 'on';
75 ax.YGrid = 'on';
76 Tick = min(ax.XTick(2)-ax.XTick(1),ax.YTick(2)-ax.YTick(1));
77 LowHighX = 2*floor(abs(XRange/Tick)).*(XRange./abs(XRange))*Tick;
78 LowHighY = 2*floor(abs(YRange/Tick)).*(YRange./abs(YRange))*Tick;
79 ax.XTick = LowHighX(1):Tick:LowHighX(2);
80 ax.YTick = LowHighY(1):Tick:LowHighY(2);
81
82 rectangle('Position',[real(c) - R, imag(c) - R, R*2, R*2],'Curvature',[1,1],'LineStyle','--')
83
84 Title4 = sprintf('Approximated Roots after %i Iterations (Gauss-Siedel)',CountGS);
85
86 figure('name','Approximated Roots (Gauss-Siedel)','Position',[0.59*scrsz(3)/3 1.51*scrsz(4)/3
       scrsz(3)/2.5 scrsz(4)/2.5])
87 plot(real(Roots),imag(Roots),'*',real(ZGS),imag(ZGS),'o')
88 title(Title4)
89 l2 = legend('Approximated Roots','Roots');
90 l2.Location = 'southoutside';
91
92 ax = gca;
93 ax.XAxisLocation = 'origin';
94 ax.YAxisLocation = 'origin';
95 ax.XLim = XRange;
96 ax.YLim = YRange;
97 axis equal;
98 ax.XGrid = 'on';
99 ax.YGrid = 'on';
100 Tick = min(ax.XTick(2)-ax.XTick(1),ax.YTick(2)-ax.YTick(1));
101 LowHighX = 2*floor(abs(XRange/Tick)).*(XRange./abs(XRange))*Tick;
102 LowHighY = 2*floor(abs(YRange/Tick)).*(YRange./abs(YRange))*Tick;
103 ax.XTick = LowHighX(1):Tick:LowHighX(2);
104 ax.YTick = LowHighY(1):Tick:LowHighY(2);
105
106 rectangle('Position',[real(c) - R, imag(c) - R, R*2, R*2],'Curvature',[1,1],'LineStyle','--')
107 %viscircles([real(c),imag(c)],R)
108
109 %% Plots Approximation Paths
110 figure('name','Approximation Path (Gauss-Siedel)','Position',[0.59*scrsz(3)/3 0.02*scrsz(4)/3
       scrsz(3)/2.5 scrsz(4)/2.5])
111 plot(real(Roots),imag(Roots),'*',real(ZnGS),imag(ZnGS),'-')
112 title('Approximation Path (Gauss-Siedel)')
113 %l3 = legend('Approximated Roots','Roots');
114 %l3.Location = 'southoutside';
115
116 ax = gca;
117 ax.XAxisLocation = 'origin';
118 ax.YAxisLocation = 'origin';
119 ax.XLim = XRange;
120 ax.YLim = YRange;
```

```
axis equal;
ax.XGrid = 'on';
ax.YGrid = 'on';
Tick = min(ax.XTick(2)-ax.XTick(1),ax.YTick(2)-ax.YTick(1));
LowHighX = 2*floor(abs(XRange/Tick)).*(XRange./abs(XRange))*Tick;
LowHighY = 2*floor(abs(YRange/Tick)).*(YRange./abs(YRange))*Tick;
ax.XTick = LowHighX(1):Tick:LowHighX(2);
ax.YTick = LowHighY(1):Tick:LowHighY(2);

rectangle('Position',[real(c) - R, imag(c) - R, R*2, R*2],'Curvature',[1,1],'LineStyle','--')

MJ = MaxDiffFinder(ZnJ,Roots);
MGS = MaxDiffFinder(ZnGS,Roots);
n = size(MJ,1);
nm1 = n-1;
np1 = n+1;
figure('name','Max Error per Step')
plot(0:nm1,MJ(:,1),'-',0:nm1,MGS(:,1),'-')
xlabel('Number of Iterations')
ylabel('Maximum Error')
title('Max Error per Step')
l3 = legend('Jacobi','Gauss-Seidel');
l3.Location = 'best';

ax=gca;
ax.XLim = [0 nm1];
Tick =ceil(ax.XTick(2)-ax.XTick(1));
ax.XTick = 0 : Tick : nm1;
end
```

References

[1] Oliver Aberth. "Iteration Methods for Finding all Zeros of a Polynomial Simultaneously." In: *Mathematics of Computation* 27.122 (1973), pp. 339–344. DOI: 10.1090/S0025-5718-1973-0329236-7.

[2] Bruce Palka. *An Introduction to Complex Function Theory*. Undergraduate Texts in Mathematics. Springer, 1991. ISBN: 978-1-4612-6967-0.

[3] Dario Bini. "Numerical Computation of Polynomial Zeros by means of Aberth's Method." In: *Numerical Algorithms* 13.2 (1996), pp. 179–200. DOI: 10.1007/BF02207694.

[4] Dario Bini and Vanni Noferini. *Solving Polynomial Eigenvalue Problems by means of the Ehrlich-Aberth Method.* Version 1. 2012. arXiv: 1207.6292v1.

[5] Eric Weisstein. *Descartes' Sign Rule.* From *Mathworld*–A Wolfram Web Resource. URL: http://~mathworld.wolfram.com/~DescartesSignRule.html.

[6] Eric Weisstein. *Newton's Method.* From *Mathworld*–A Wolfram Web Resource. URL: http://mathworld.wolfram.com/NewtonsMethod.html.