Parallel Schemes for Global Iterative Zero-Finding

A Thesis

presented to the Department of Computer Science

of The Chinese University of Hong Kong

in partial fulfillment of the requirements

for the Degree of Master of Philosophy

by

Luk Wai Shing

May 1993

UL

Thesis
QA
161
P59L84
1993

ABSTRACT

In considering the problem of finding all the zeros of the N-th degree polynomial, classical algorithms take the form of calculating the zeros one at a time, each followed by a deflation step to remove the calculated zero α_0 from the polynomial. Usually, synthetic division is used for the deflation step. That is, the polynomial is divided by $(z - \alpha_0)$ to yield the deflated polynomial. These algorithms seem to be hardly parallelizable since the deflation step in-between the iterations is inherently sequential.

Two alternative methods, which are natually parallelizable, are the Durand-Kerner method [13] and the Aberth method [1]. Both of the methods start with N distinct initial approximations and converge to all the zeros separately. However, we observe that the Aberth method is a modification of the Newton method by performing a pseudo-deflation with iterates instead of computed zeros. By this observation, we introduce parallel versions of a class of existing algorithms, such as the Halley method, the Laguerre method and the Cluster Adapted method. All the methods of our class are locally convergent and the convergent rates are increased by one from their sequential counterparts in case of simple zeros. Moreover, the fourth-order algorithms are robust in numerical examples. Also, we study the choice of initial approximations and the improvement on these algorithms in case of multiple zeros.

The global behavior of classical algorithms is investigated by using computer graphic and visualization techniques. The visualization process is time consuming and it often takes many hours for a workstation to display a graph. However, an implementation on the MasPar massively parallel computer shows that the program running on this machine can provide us a quick response. As a result, we can include some interactive facilities, such as zoom-in, zoom-out, in our program.

Recently, the use of homotopy methods for solving eigenvalue problems and solving polynomial systems has been studied by many researchers (eg. [15, 16, 17]), due to its naturally parallelizm and its robustness. In this thesis, an homotopy method that solves single polynomial for all zeros is presented. We illustrate that multiple bifurcation can occur at this simple case by some examples. A method to overcome bifurcation is also discussed.

ACKNOWLEDGMENTS

Many people contributed to this project, either directly or indirectly. Particularly, I would like to thank Mr. Wong-chuen Kwok, who spent many days and nights on discussing problems with me. Prof. E.V. Krishnamurthy made the helpful comments on the proposed parallel deflation scheme. Also, Prof. Tony Chan suggested the homotopy method for global convergent problem. My greatest thanks must go to Prof. Tien Chi Chen. His discovery and advice was invaluable to this project.

Table of Contents

CHAPTER 1. INTRODUCTION

Polynomial zero-finding has been studied for over a century. Earliest studies were concentrated on real solutions of polynomial with real coefficients only. As time went by, modern science convinced us that the imaginary part is equally important in nature. Also, many engineering applications often give rise to solving for complex roots of polynomials. Therefore, our focus will be concentrated on polynomials with complex roots.

It is well known that there does not exist a direct method that can find the roots of a polynomial with degree higher than four. It means that we can only use an iterative method to solve the problem numerically. The most famous iterative method is definitely the Newton method. Higher order iterative methods have also been developed, such as the Halley method, the Laguerre Method, and the Cluster Adapted formula that was newly developed by Chen [5]. In Section 2.1, we will review these formulas and discuss the problems of the classical theory.

Based on the classical theory, the local behavior of these methods has been fully understood. For example, one can determine whether a formula is locally convergent or not, and what the order of convergence is. But how about the global behavior? In the past, we could hardly answer the question because the global behavior is very complex. With the help of the computer graphics and computer visualization techniques, we may now realize the global behavior more easily than at any time in the past. In Section 2.2, we will describe how computer graphics give us the insight of the global view.

In this thesis, we want to develop parallel and globally convergent algorithms to find all the zeros of a polynomial. In order to find all the zeros, conventionally (i.e. not in parallel), a deflation process is used. As a zero α_0 has been identified, synthetic division of $P(z)$ by $z - \alpha_0$ yields the deflated polynomial $Q(z)$ such that $P(z) = (z - \alpha_0)$ $Q(z)$. The deflated polynomial has a degree one less than the original and has the roots exactly the same as the remaining roots of $P(z)$. To find additional zeros of $P(z)$, the zero-finding method can be applied again to this deflated polynomial. A more detailed revision of the deflation process will be given in Section 2.3. The deflation process has two advantages: it prevents the next iterate from approaching the same zero, and the problem size is deflated by one order after the process. However, the deflation has also two disadvantages. One is that the roundoff error is accumulated in each deflation process. Another one is that the algorithm will be inherently sequential. To overcome this problem, two different parallel schemes are proposed in Section 3 and Section 6.

In Section 3.1, two parallel iterative methods, the Durand-Kerner method and the Aberth method, are introduced. We will summarize recent researches on this topic. One important observation that should be pointed out is that the Aberth method is just a modification of the Newton method by introducing the concept of "pseudo-deflation". Therefore, it is not surprising that the Aberth method inherit many problems from the Newton method. According to this observation, we perform many improvements on the Aberth method in Section 3.3. Also, by applying the concept of "pseudo-deflation", we parallelize several other sequential iterative methods in Section 4.

Although this kind of algorithms can approximate all the zeros simultaneously without an explicit deflation, in certain circumstances, performing an explicit deflation may make the algorithm more efficient. For example, we observe that a single zero is converged to more quickly than a multiple zero by using these algorithms. Therefore, it is better to deflate the zeros which have been computed accurately enough. In Section 5, we will propose an algorithm for parallel deflation by using the Fourier Transform.

In recent years, many researches on solving system of polynomial equations are focused on using an homotopy method (see [15] for example). It is obvious that the polynomial root-finding is just a particular case of such system of polynomial equations, i.e., containing only one equation and only one unknown variable of degree N. In section 6, we intend to investigate the homotopy method on this particular case. Several modification and improvement are introduced for efficient convergence. The bifurcation problem will also be studied.

CHAPTER 2. DRAWBACKS OF CLASSICAL THEORY

2.1 Review of Sequential Iterative Methods

2.1.1 Consideration of the Symmetric Cluster [5]

Consider an N th degree polynomial with complex coefficients :

$$P(z) = \sum_{i=0}^{N} c_i \cdot z^i = C\prod_{j=1}^{n}(z-\alpha_j)^{m_j}$$

where

$$N = \sum_{j=1}^{n} m_j$$

α_j = the j th zero of $P(z)$

m_j = the multiplicity of α_j .

For convenience, we set $c_N = C = 1$ without loss of generality. We define three auxiliary functions:

$$s_1 = \frac{d\ln(P(z))}{dz} = \frac{P'(z)}{P(z)} = \sum_{j=1}^{n}\frac{m_j}{(z-\alpha_j)} \tag{2.1.1}$$

$$s_2 = -\frac{ds_1}{dz} = \frac{P'(z)^2 - P''(z)P(z)}{P(z)^2} = \sum_{j=1}^{n}\frac{m_j}{(z-\alpha_j)^2} \tag{2.1.2}$$

$$\mu = \frac{s_1^{\ 2}}{s_2} = \left(\sum_{j=1}^{n}\frac{m_j}{(z-\alpha_j)}\right)^2 \Bigg/ \sum_{j=1}^{n}\frac{m_j}{(z-\alpha_j)^2}\ , \tag{2.1.3}$$

which are used in classical iterative zero-finding techniques. Conventionally, μ estimates the multiplicity of the zero being approached.

We define a polynomial with the following form as a "symmetric cluster function" with n zeros, which all have the multiplicity m, placed symmetrically on a circle with center A and radius r:

$$C(z) = \left[(z-A)^n - r^n\right]^m = \prod_{j=1}^{n} \left[z - (A + r \cdot \exp(2\pi ij/n))\right]^m . \tag{2.1.4}$$

Let $t = (z - A) / r$, three regions are then defined with respect to the cluster:

1. The inner region: $|t| << 1$
2. The target region: $|t| \cong 1$
3. The outer region: $|t| >> 1$.

The classical theory states that the zeros could be found when the guess $z^{(k)}$ is within the target region. However, global convergence is harder to achieve. In earlier studies, we observed that a "rebound" phenomenon occurs in such cluster functions when using the multiple Newton method [4]:

$$z^{(k+1)} = NT(z^{(k)}; p) = z^{(k)} - \frac{p}{s_1}, \tag{2.1.5}$$

where p = estimated multiplicity. This phenomenon can be illustrated by a simple example $(z - 1.00001)(z - 0.99999)$ [4]. For $z^{(0)} = 1.1$, the iterate jumps near the centroid for $p = 2$ in the first iteration. However, due to the numerical cancellation, the next iterate for all $p \geq 1$ jumps further away from the zeros. The process will be repeated until the iterate approaches infinity. In [4], we conclude that :

1. The convergence of original Newton formula $NT(z^{(k)};1)$ is linear for $z^{(0)} = 1.1$, instead of quadratic, in case of this cluster zeros, as poor as in case of multiple zeros.

2. The multiple zero formula $NT(z^{(k)};p)$ may fail to converge to cluster zeros due to the "rebound" phenomenon.

Just like the classification between sparse matrix and dense matrix, the classification between cluster zero and single zero is somewhat fuzzy. It depends relatively on the distances between the individual zeros, and the distances between the guess and the individual zeros. Here, we roughly define a cluster zero as a group of zeros that can not be distinguished from a multiple zero based on the computed value of μ. Cluster zeros yields trap to the Newton method and many other iterative methods, that was seldom noticed in the past.

Another method is called the Halley method [12] :

$$z^{(k+1)} = HL(z^{(k)};p) = z^{(k)} - \frac{2}{s_1\left(\frac{1}{p}+\frac{1}{\mu}\right)} \tag{2.1.6}$$

which is cubically convergent. Similar to Newton's method, the Halley method suffers from the fact that a real initial guess can never converge to a complex root of a real coefficient polynomial. It is because there is no mechanism for the formulas to produce a complex value.

A better performance results by using the Laguerre method [12]:

$$z^{(k+1)} = LG(z^{(k)};p) = z^{(k)} - \frac{N}{s_1\left(1 \pm \sqrt{\left(\frac{N}{p}-1\right)\left(\frac{N}{\mu}-1\right)}\right)} \tag{2.1.7}$$

with a sign chosen to make the real part of the square-root positive. The Laguerre method works fine for lower degree of the symmetric cluster function. However, it tends to fail to converge for $n > 4$.

Based on this observation, Chen invented the cluster-adapted method [5] :

$$z^{(k+1)} = CMR(z^{(k)};p) = z^{(k)} - \frac{N(Q^{p/N}-1)}{s_1(Q-1)} \tag{2.1.8}$$

with

$$Q = \left(\frac{N}{\mu}-1\right) \Big/ \left(\frac{N}{p}-1\right)$$

and the principal value is chosen from the roots of $Q^{p/N}$. The cluster adapted method has the ability to reach a zero in one iteration if the polynomial is a symmetric cluster, and p exactly matches m. For arbitrary functions, the cluster adapted method also has an excellent performance compared with other methods.

2.1.2 Measure of Multiplicity

Classically, the function μ estimates the multiplicity. However, μ may be either infinity, or zero depending on where the guess is. Moreover, it is often a complex number with a considerable imaginary part. In order to reflect the multiplicity properly, its rounded real part, denoted by w, is used. Moreover, w still may not be directly substituted into p, since its extreme values N and 0 will degenerate $CMR(z;p)$ or $LG(z;p)$ into the Newton formula of which the convergence rate is quadratic only [5]. In [5], Chen suggested that w is qualified if and only if $(N-1) \geq w \geq 1$ and the real part of μ is two times larger than the imaginary part of μ. The default choice of p is one if w is not qualified. The practical scheme of multiplicity estimation is defined as below:

```
Step 1.  Let $w = \text{FLOOR}(0.5 + \text{Re}(\mu))$
Step 2.  Then $p = w$ iff $(N-1) \geq w \geq 1$ and $w \geq 2|\text{Im}(\mu)|$
                  $= 1$ otherwise
```

2.2 Visualization Techniques

The classical iteration theory cannot give us a global view of the iterative methods, and seldom considers the situation of the presence of cluster zeros. In this section, we employ a visualization technique for helping us to get some idea of the global behavior. See [20] for more details of discussions.

Undoubtedly, iterative function is an example of dynamical systems. Although the theory of dynamical systems is beyond the scope of this thesis, the graphical visualization technique, which is often used in that field to observe the chaotic behavior, will be helpful for observing the global behavior of iterative methods.

The number of iterations that is required for convergence depends on where the initial guess is in the complex plane. For some initial points, the iterative process will quickly converge; for some initial points however, the process may be divergent or cyclic. At some regions, a little change of the initial point will make a big difference in the number of iterations, or make the iterative process from being convergent to being divergent. For a global view of this behavior, we can develop a graph mapping similar to the Julia plot. Each initial guess in complex plane is represented by a pixel in graphical window. The vertical axis is the real axis and the horizontal axis is the imaginary axis. The number of iterations is then represented by a color index of the pixel. We developed a set of tools to create this kind of images and display them via the MATLAB software. The source codes are listed in Appendix A. All the programs can only be run on SUN Sparc system. Appendix B illustrates some results created by the tools.

Fig.A.1.a illustrates the graph mapping of solving a polynomial $z^7 - 1 = 0$ by the Newton method. The color of each pixel represents the number of iterations and is defined by the by the default colormap as shown in Fig.A.0.a. The default colormap is generated by a random generating function in MATLAB. We observe that the Newton method is not convergent with the initial guess on the symmetrical axis of the roots. Around the symmetrical axis, the Newton method exhibits a chaotic behavior and is "unstable". What "unstable" here means a small change of the position of the initial guess make a great difference in the number of iterations. By using the color mapping technique, we can enhance the unstable region with different colors. Fig.A.0.b and Fig.A.0.c show two colormaps, HSV and JET, which are used for this purpose. Fig.A.2.a is an enhanced graph of Fig.A.1.a.

Furthermore, we generate a graph mapping at the unstable region and discover that it is self-similar (see Fig.A.3.a). An object that has the property of self-similarity is called a Fractal. We can further scale down the graph theoretically at unlimited levels without loss of any detail. At the unstable region, we discover the "relatively" stable sub-regions. For example, as shown in Fig.A.3.a, we can see many "eyes" which are relatively more stable than the other regions in this graph.

Recently, Hong Kong has installed a MasPar MP-2 massively parallel computer. It is SIMD machine with 8,192 processor elements (PE's) and 64Kbytes memory on each PE. We have ported our software onto this machine and fully utilize the parallelism. Since the communication among the PE's is minimum, it turns out that the visualization process can quickly be completed. As a result, we can include some facilities, such as zoom-in, zoom-out in our program.

2.3 Review of Deflation

When a zero α_0 of a polynomial has been identified, we must do something to prevent another guess from converging to this zero. Traditionally, the polynomial is divided by $(z - \alpha_0)$ and the iterative method is then applied to the quotient polynomial. This process is called deflation. Conventionally, synthetic division is used for deflation process. In [12], detailed analysis of this deflation process was discussed and an alternative method of suppressing computed zeros without an explicit deflation was suggested as follows.

Suppose the first r zeros $(\alpha_1, \alpha_2, \cdots \alpha_r)$ with multiplicities $(m_1, m_2, \cdots m_r)$ have been identified, the deflated polynomial $h^{(r)}(z)$ is given by:

$$h^{(r)}(z) = \frac{P(z)}{\prod_{j=1}^{r} (z - \alpha_j)^{m_j}} . \tag{2.3.1}$$

To apply Newton method, we need only s_1, μ of $h^{(r)}(z)$ $(s_1^{(r)}, \mu^{(r)})$. Since

$$s_1^{(r)} = s_1 - \sum_{j=1}^{r} \frac{m_j}{z - \alpha_j} \tag{2.3.2}$$

$$s_2^{(r)} = s_2 - \sum_{j=1}^{r} \frac{m_j}{(z - \alpha_j)^2} \tag{2.3.3}$$

$$\mu^{(r)} = \frac{(s_1^{(r)})^2}{s_2^{(r)}} \tag{2.3.4}$$

$$F(z; p) = z - \frac{p}{s_1^{(r)}} \tag{2.3.5}$$

where p is the estimated multiplicity calculated by $\mu^{(r)}$ (see the Section 2.1.2), we can compute everything needed by the iterative methods using eq.(2.3.2)-(2.3.5). The tradeoff is that we must recompute these equations in each iteration.

CHAPTER 3. THE IMPROVEMENT OF THE ABERTH METHOD

3.1 The Durand-Kerner method and the Aberth method

In recent years, there has been much research on parallel root-finding. In this section, we introduce two iterative methods and summarize the work done on these methods. One of the iteration methods is called the Durand-Kerner method, which was proposed independently by E. Durand and I. O. Kerner [13]. The formula of the D-K method is the following:

$$DK_i(Z) = z_i - \frac{P(z_i)}{\prod_{j\neq i}^{N}(z_i - z_j)} \qquad i=1, \dots ,N \qquad (3.1.1)$$

with Z a vector of z_i $(z_1, z_2, \dots , z_N)$.

Another method proposed by O. Aberth, which is called the Aberth method, uses an approach which uses an analogy with electrostatics [1].

$$AB_i(Z) = z_i - \frac{1}{\frac{P'(z_i)}{P(z_i)} - \sum_{j\neq i}^{N}\frac{1}{z_i - z_j}} \qquad i=1, \dots ,N. \qquad (3.1.2)$$

The following are some properties of the Durand-Kerner method [1,7,8,14]:

1. The initial approximations must be distinct.

2. The centroid of the approximations after one iteration is equal to the centroid of the zeros of the polynomial and is invariant.

3. No two iterates approach the same zero in case of simple zeros. Only M iterates approach a zero with multiplicity M.

4. The convergence order is quadratic in case of simple zeros, but only linear in case of multiple zeros.

5. The convergence speed strongly depends on the choice of the starting points.

6. The several iterates which converge towards the same multiple zero have a tendency to approach their limit point from symmetrically distributed directions, as if they repell each other.

7. QLMC (Quadratic-Like Convergence of the Mean) :
   The convergence order of the mean of the components converging to the same multiple root is quadratic.

Properties 1,3,5 also hold in the Aberth method. The convergence in the Aberth method is cubic in case of simple roots and slow rate of convergence is observed to multiple zeros. (Note that the convergence rate is meaningful only if Z is sufficiently close to the zeros in both methods.) Because of the drawback of slow convergence in case of multiple zeros in both methods, many improvements have been suggested [8,18]. Fraigniand [8] made use of property 7 for speeding up the convergence in the D-K method. The algorithm cannot be generalized to other iteration methods unless they have the same property. Miyakoda [18] made use of property 6 for grouping guesses. However, cluster zeros also exhibits a similar property of multiple zero. This is the reason why Miyakoda's paper claimed that false grouping often occurred for double zeros (the simplest cluster zeros). Many iterative methods fail to converge due to the lack of consideration of cluster zeros.

3.2 Generalized Aberth Method

One can observe that the Aberth method is a modification of the Newton method although O. Aberth derived it by different approach. The additional summation term improves the convergency of Newton method with the help of z_j's, making it cubic instead of quadratic. What is more important, the summation term prevents the iterates from approaching the same zero, which conventionally can only be performed by means of deflation. Surprising enough, while comparing the Aberth method with eq.2.3.5, we find the great similarity between these two methods except the multiplicity. Hence the Aberth method performs a pseudo-deflation by using iterates instead of computed zeros. The argument is particularly true when all the z_j's are close to the zeros except z_i. Similarly we can apply the same argument to the D-K method.

In order to better explain the relationship between the Newton method and the Aberth method, we create a generalized Aberth method which use an "M" substituting "N" and M is between 1 and N:

$$G_i(Z) = z_i - \frac{1}{\dfrac{P'(z_i)}{P(z_i)} - \sum\limits_{j \neq i}^{M} \dfrac{1}{z_i - z_j}} \qquad i=1, \ldots ,M \qquad (3.2.1)$$

where $1 \le M \le N$.

When M is equal to 1, it is the Newton method. When M is equal to N, it is the Aberth method. The formula also suggests that it is not necessary to find all the zeros.

3.3 Modified Aberth Method for multiple-zero

Similar to the multiple-zero Newton method, Chen suggested the multiple-zero \berth method:

$$pNT_i(Z) = z_i - \frac{p_i}{\frac{P'(z_i)}{P(z_i)} - \sum_{j \neq i}^{N} \frac{p_j}{z_i - z_j}} \qquad i = 1, \dots, n \qquad (3.3.1)$$

where

p_j = estimated multiplicity of z_j,

$$\sum_{j=1}^{n} p_j = N$$

which can improve the convergence in case of multiple zeros. The method needs to group the guesses which are approaching the same multiple zero. The guesses are then "merged" to a single guess with an estimated multiplicity. Somehow we may need to "split" the guess in further iterations due to cluster zeros or mis-grouping.

An alternative method is to ignore the contribution of the nearest (p_i - 1) guesses in each iteration. Hence we slightly modify eq.(3.3.1) and the formula is:

$$MA_i(Z) = z_i - \frac{W_i}{\frac{P'(z_i)}{P(z_i)} - \sum_{V_i} \frac{1}{z_i - z_j}} \qquad i=1, \dots, N \qquad (3.3.2)$$

where

$$W_i = p_i \quad \text{if} \quad \sum_{j=1}^{n} p_j = N$$

$$= 1 \quad \text{otherwise}$$

V_i = {Set of all guesses, except z_i plus the nearest (W_i - 1) iterates of z_i}.

The additional cost is that we need to search the nearest (W_i - 1) iterates if W_i is not equal to 1. In consideration of parallel processing, it creates an unequal work amount in some processors due to different W_i. The need for synchronization of all processors in each iteration reduces the efficiency of parallel machine. On the other hand, the advantages of this scheme are the simplicity of implementation and the minimum communication cost. Therefore, all the iterative methods discussed below will be modified in this manner during implementation.

3.4 Choosing the initial approximations

By using the Durand-Kerner method or the Aberth method, the iteration processes are always convergent for almost any initial values. However, choosing proper initial approximations is also important since it affects the efficiency very much. As far as we are concerned, the initial approximations should be as close to the zeros as possible and should be calculated easily.

O. Aberth suggested that the initial approximations should be evenly distributed on a circle with the centre A equals the centroid of zeros [1]:

$$z_j^{(0)} = A + R \cdot \exp\left\{\frac{2\pi i j}{N} + i\phi\right\} \qquad j = 1,\ldots,N \qquad (3.4.1)$$

where

$$i = \sqrt{-1}$$

$$A = \frac{-c_{N-1}}{N} .$$

The angle ϕ is used to break the symmetry with respect to the real axis and usually be taken as $\pi/2N$. In most cases, we just put it as 0. Aberth also suggested that the radius R should be taken such that the circle just encloses all the zeros. However, some researchers advised that the optimum radius should be between the minimum length and the maximum length of zeros.

Y. Nagashima proposed a method for calculating R by introducing the concept of quasivariance [19], where

$$R_s = \sqrt{\frac{1}{N}\left|\left(1-\frac{1}{N}\right)c_{N-1}{}^2 - 2c_{N-2}\right|} .$$

Nevertheless, he did not mention the situation of $c_{N-1} = c_{N-2} = 0$, which could occur commonly.

H. Guggenheimer claimed that the radius calculated from his procedure produces results not far from the optimum [10]. However, if the centroid of polynomial is afar from the origin, you must take a translation from the origin to the centroid in order to use the procedure efficiently.

Chen introduced the concept of effective radius for sequential iterative methods [6], which can also be applied here:

$$R_{eff} = (-P(A))^{1/N}.$$

Note that R_{eff} is taken as a complex value. It can be shown that if the polynomial is a simple symmetric cluster function with m = 1, the initial approximations:

$$z_j^{(0)} = A + R_{eff} \cdot \exp\left\{\frac{2\pi ij}{N}\right\} \qquad j = 1,\ldots,N \tag{3.4.2}$$

are already the zeros of the polynomial.

The effective radius can be calculated more easily compared with the Guggenheimer's procedure and exhibits a good performance of convergence. Thus the effective radius is used in our scheme.

3.5 Multiplicity estimation

Multiplicity is an important quantity for improving the convergence to multiple zeros. Nevertheless, it is difficult to measure this quantity exactly. G. Kjellbery first observed that m iterates converge towards the same multiple zero with multiplicity m from symmetrically distributed direction in using Durand-Kerner method [14]. This property was later used for multiplicity judgment by T. Miyakoda [18]. Nevertheless, this method will take quite a long time in classifying and grouping. Moreover, the symmetric cluster shares the same property when the corresponding iterates are afar from the cluster. Actually, I suspect that one cannot distinguish cluster zeros from multiple zero by numerical means, until the corresponding iterates are sufficiently closed to it. So, the multiplicity must necessarily be estimated in each iteration. Here, we slightly modify Chen's scheme of multiplicity estimation (Section 2.1.2) to it's parallel counterpart. Let us define:

$$s_{1,i} = \frac{P'(z_i)}{P(z_i)}$$

$$s_{2,i} = \frac{P'(z_i) - P''(z_i)}{P(z_i)^2}$$

$$\mu_i = \frac{s_{1,i}^{\ 2}}{s_{2,i}} .$$

Since the multiplicity is a real positive integer, we have

$$w_i = \text{Floor}(0.5 + |\text{Re}(\mu)|)$$

$$\begin{aligned} p_i &= w_i && \text{if } 1 < w_i < N \text{ and } w_i > 2|\text{Im}(\mu_i)| \\ &= 1 && \text{otherwise} \end{aligned}$$

where

p_i = estimated multiplicity.

CHAPTER 4

THE HIGHER-ORDER ITERATIVE METHODS

4.1 Introduction

According to the discussion of Section 3, in principle we can modify other classical iterative methods, such as the Halley Method [12], the Laguaerre Method [12] and the Cluster Adapter method [6], to their parallel counterparts. Let

$$\tilde{s}_{1,i} = s_{1,i} - \sum_{j \neq i}^{n} \frac{p_j}{z_i - z_j} \tag{4.1.1}$$

$$\tilde{s}_{2,i} = s_{2,i} - \sum_{j \neq i}^{n} \frac{p_j}{(z_i - z_j)^2} \tag{4.1.2}$$

$$\tilde{\mu}_i = \frac{\tilde{s}_{1,i}^{\ 2}}{\tilde{s}_{2,i}} . \tag{4.1.3}$$

The parallel iterative formulas $pF(Z)$ for the i th iterate are:

(i) The parallel-Newton formula (the modified Aberth formula):

$$pNT_i(Z) = z_i - \frac{p_i}{\tilde{s}_{1,i}} . \tag{4.1.4}$$

(ii) The parallel-Halley formula:

$$pHL_i(Z) = z_i - \frac{2}{\tilde{s}_{1,i}\left(\dfrac{1}{p_i} + \dfrac{1}{\tilde{\mu}_i}\right)} . \tag{4.1.5}$$

(iii) The parallel-Laguerre formula:

$$pLG_i(Z) = z_i - \frac{N}{\tilde{s}_{1,i}\left(1 \pm \sqrt{\left(\frac{N}{p_i}-1\right)\left(\frac{N}{\tilde{\mu}_i}-1\right)}\right)}. \tag{4.1.6}$$

(vi) The parallel-Cluster Adapted formula:

$$pCMR_i(Z) = z_i - \frac{N(Q_i^{\,p_i/N}-1)}{\tilde{s}_{1,i}(Q_i-1)} \tag{4.1.7}$$

with $$Q_i = \left(\frac{N}{\tilde{\mu}_i}-1\right) \Big/ \left(\frac{N}{p_i}-1\right).$$

We will prove later (Section 4.2) that the orders of convergence of these formulas are increased by one from their sequential counterparts in case of all single zeros.

The overall parallel iterative algorithm are now proposed:

Step 1. Generate the initial iterates $Z^{(0)} = (z_1^{(0)}, z_2^{(0)}, \cdots, z_N^{(0)})$ in parallel:

$$z_j^{(0)} := A + R_{eff} \cdot \exp(2\pi i j / N) \qquad j = 1,\ldots,N.$$

Step 2. Compute the next set of iterates in parallel:

$$Z^{(k+1)} := pF(Z^{(k)}) \qquad \text{where } k = 0 \text{ at the beginning.}$$

Step 3. If $\left|P(z_i^{(k+1)})\right| \le \varepsilon$ for all i, then Exit.

Step 4. $k := k+1$; Goto Step 2.

4.2 Convergence analysis

Table 4.1 illustrates order of convergence of several iterative methods in case of single zeros. Some of the results had been proved and well known. In this section, we will verify these results, in addition to giving proofs for the rest of the methods that are proposed in this thesis. More importantly, we demonstrate the great similarity between the sequential algorithms and the pseudo deflation method.

Sequential formula	Order	Parallel formula	Order
Newton	2	Aberth	3
Halley	3	Parallel-Halley	4
Laguerre	3	Parallel-Laguerre	4
CMR	3	Parallel-CMR	4

Table 4.1 The convergent rate of different iterative formulas in case of single zeros.

Firstly, we need three equalities deduced from the Taylor expansion:

$$(1+\delta)^p = 1 + p\delta + \frac{p(p-1)}{2}\delta^2 + \frac{p(p-1)(p-2)}{3!}\delta^3 + O(\delta^4).$$

Supposing that $|\delta| << 1$. We have,

$$(1+\delta)^{-1} = 1 - \delta + \delta^2 + O(\delta^3) \tag{4.2.1}$$

$$(1+\delta)^{1/2} = 1 + \frac{1}{2}\delta - \frac{1}{8}\delta^2 + O(\delta^3) \tag{4.2.2}$$

$$(1+\delta)^{1/N} = 1 + \frac{1}{N}\delta - \frac{N-1}{2N^2}\delta^2 + \frac{(N-1)(2N-1)}{3!N^3}\delta^3 + O(\delta^4). \tag{4.2.3}$$

Substitute $s_{1,i}$ by eq.(4.2.4), we get

$$\tilde{s}_{1,i} = \frac{1}{\delta_i} + \sum_{j\neq i}^{N} \frac{1}{z_i - \alpha_j} - \sum_{j\neq i}^{N} \frac{1}{z_i - z_j}$$

$$= \frac{1}{\delta_i} + \sum_{j\neq i}^{N} \left(\frac{1}{z_i - \alpha_j} - \frac{1}{z_i - z_j} \right)$$

$$= \frac{1}{\delta_i} + \sum_{j\neq i}^{N} \left(\frac{1}{z_i - \alpha_j} - \frac{1}{z_i - \alpha_j - \delta_j} \right)$$

$$= \frac{1}{\delta_i} + \sum_{j\neq i}^{N} \left(\frac{(-\delta_j)}{(z_i - \alpha_j)(z_i - \alpha_j - \delta_j)} \right)$$

$$= \frac{1}{\delta_i} + \tilde{A}$$

where $\tilde{A} = \sum_{j\neq i}^{N} \left(\frac{(-\delta_j)}{(z_i - \alpha_j)(z_i - \alpha_j - \delta_j)} \right) = O(\sum_{j\neq i}^{N} \delta_j)$.

From eq.(4.1.2), we get

$$\tilde{s}_{2,i} = s_{2,i} - \sum_{j\neq i}^{N} \frac{1}{(z_i - z_j)^2}. \tag{4.2.9}$$

Substitute $s_{2,i}$ by eq.(4.2.5), we have

$$s_{2,i} = \frac{1}{\delta_i^2} + \sum_{j\neq i}^{N} \left(\frac{1}{(z_i - \alpha_j)^2} - \frac{1}{(z_i - z_j)^2} \right)$$

$$= \frac{1}{\delta_i^2} + \sum_{j\neq i}^{N} \left(\frac{1}{(z_i - \alpha_j)^2} - \frac{1}{(z_i - \alpha_j - \delta_j)^2} \right)$$

$$= \frac{1}{\delta_i^2} + \sum_{j\neq i}^{N} \left(\frac{(2z - 2\alpha_j - \delta_j)(-\delta_j)}{(z_i - \alpha_j)^2 (z_i - \alpha_j - \delta_j)^2} \right)$$

$$= \frac{1}{\delta_i^2} + \tilde{B}$$

where $\tilde{B} = \sum_{j\neq i}^{N} \left(\frac{(2z - 2\alpha_j - \delta_j)(-\delta_j)}{(z_i - \alpha_j)^2 (z_i - \alpha_j - \delta_j)^2} \right) = O(\sum_{j\neq i}^{N} \delta_j)$.

Therefore, we have

$$\tilde{s}_{1,i} = \frac{1}{\delta_i} + \tilde{A} \tag{4.2.10}$$

$$\tilde{s}_{2,i} = \frac{1}{\delta_i^2} + \tilde{B} \tag{4.2.11}$$

$$\tilde{\mu}_i = \frac{\tilde{s}_{1,i}^{\ 2}}{\tilde{s}_{2,i}} = \frac{(1+\tilde{A}\,\delta_i)^2}{1+\tilde{B}\delta_i^{\ 2}}. \tag{4.2.12}$$

Since the parallel formulas have the same structure of the sequential formulas except $s_{1,i}, s_{2,i}, \mu_i$ replaced by $\tilde{s}_{1,i}, \tilde{s}_{2,i}, \tilde{\mu}_i$, we can prove the convergence in the same manner.

(i) The Newton Method

From eq.(2.1.5), we get

$$\tilde{z}_i = z_i - \frac{1}{s_{1,i}}.$$

Substitute $s_{1,i}$ by eq.(4.2.6), we have

$$\begin{aligned}\tilde{z}_i &= \alpha_i + \delta_i - \frac{1}{\frac{1}{\delta_i} + A} \\ &= \alpha_i + \delta_i - \frac{\delta_i}{1 + A\,\delta_i} \\ &= \alpha_i + \frac{A\,\delta_i^{\ 2}}{1 + A\,\delta_i}.\end{aligned}$$

We assume that z_i is sufficiently close to α_i such that $A\delta_i << 1$, From eq.(4.2.1)

$$\begin{aligned}\tilde{z}_i &= \alpha_i + A\,\delta_i^2(1 - A\,\delta_i + O(\delta_i^{\ 2})) \\ &= \alpha_i + A\,\delta_i^{\ 2} + O(\delta_i^{\ 3})\end{aligned}$$

Therefore, the convergent rate of the Newton method is quadratic in case of single zero.

(ii) The Aberth method

From eq.(4.1.4), we get

$$\tilde{z}_i = z_i - \frac{1}{\tilde{s}_{1,i}}.$$

Similar to the Newton method, we have

$$\begin{aligned}\tilde{z}_i &= \alpha_i + \tilde{A}\delta_i^2 + O(\delta_i^3) \\ &= \alpha_i + O(\textstyle\sum \delta_j)\delta_i^2 + O(\delta_i^3)\end{aligned}$$

Again, we assume that z_i is sufficiently close to α_i for all i such that $\tilde{A}\delta_i << 1$ and $O(\sum \delta_j) = O(\delta_i)$, we have

$$\tilde{z}_i = \alpha_i + O(\delta_i^3).$$

Therefore, the convergent rate of the Aberth method is cubic in case of all single zeros.

(iii) The Halley Method

From eq.(2.1.6), we get

$$\tilde{z}_i = z_i - \frac{2}{s_{1,i}\left(1 + \frac{1}{\mu_i}\right)}.$$

Substitute $s_{1,i}$ and μ_i by eq.(4.2.6) and eq.(4.2.8), we have

$$\begin{aligned}\tilde{z}_i &= \alpha_i + \delta_i - \frac{2}{\left(1 + \frac{1 + B\delta_i^2}{(1 + A\delta_i)^2}\right)\left(\frac{1}{\delta_i} + A\right)} \\ &= \alpha_i + \delta_i - \frac{2\delta_i(1 + A\delta_i)}{(1 + A\delta_i)^2 + (1 + B\delta_i^2)} \\ &= \alpha_i + \frac{(A^2 + B)\delta_i^3}{2(1 + A\delta_i) + (A^2 + B)\delta_i^2}\end{aligned}$$

$$= \alpha_i + \frac{1}{2}(A^2 + B)\delta_i^3 + O(\delta_i^4).$$

Therefore, the convergent rate of the Halley method is cubic in case of single zeros.

(iv) The parallel-Halley method

From eq.(4.1.5), we get

$$\tilde{z}_i = z_i - \frac{2}{\tilde{s}_{1,i}\left(1 + \frac{1}{\tilde{\mu}_i}\right)}.$$

Similar to (iii), we have

$$\tilde{z}_i = \alpha_i + \frac{1}{2}(\tilde{A}^2 + \tilde{B})\delta_i^3 + O(\delta_i^4)$$
$$= \alpha_i + O(\delta_i^4).$$

Therefore, the convergent rate of the parallel-Halley method is fourth order in case of all single zeros.

(v) The Laguerre method

From eq.(2.1.7), we get

$$\tilde{z}_i = z_i - \frac{N}{s_{1,i}\left(1 \pm \sqrt{(N-1)\left(\frac{N}{\mu_i} - 1\right)}\right)}.$$

Substitute $s_{1,i}$ and μ_i by eq.(4.2.6) and eq.(4.2.8), we have

$$\tilde{z}_i = \alpha_i + \delta_i - \frac{N}{\left(\frac{1}{\delta_i} + A\right)\left(1 + \left[(N-1)\left(\frac{N(1+B\delta_i^2)}{(1+A\delta_i)^2} - 1\right)\right]^{\frac{1}{2}}\right)}$$

$$= \alpha_i + \frac{(N-1)B - A^2}{2(N-1)} \delta_i^3 + O(\delta_i^4)$$

Therefore, the convergent rate of the Laguerre method is cubic in case of single zeros.

(vi) The parallel-Laguerre method

From eq.(4.1.6), we get

$$\tilde{z}_i = z_i - \frac{N}{\tilde{s}_{1,i}\left(1 \pm \sqrt{(N-1)\left(\frac{N}{\tilde{\mu}_i} - 1\right)}\right)}.$$

Similar to (v), we have

$$\tilde{z}_i = \alpha_i + \frac{(N-1)\tilde{B} - \tilde{A}^2}{2(N-1)} \delta_i^3 + O(\delta_i^4)$$

$$= \alpha_i + O(\delta_i^4).$$

Therefore, the convergent rate of the parallel-Laguerre method is fourth order in case of all single zeros.

(vii) The CMR method

From eq.(2.1.8), we get

$$\tilde{z}_i = z_i - \frac{N(Q^{1/N} - 1)}{s_{1,i}(Q-1)}$$

with $$Q = \left(\frac{N}{\mu_i} - 1\right) \Big/ (N-1).$$

Substitute $s_{1,i}$ and μ_i by eq.(4.2.6) and eq.(4.2.8) and after some simplications, we have

$$\tilde{z}_i = \alpha_i - \frac{1}{2}\left[\frac{(N-5)A^2}{3(N-1)} - B\right]\delta_i^3 + O(\delta_i^4).$$

Therefore, the convergence rate of the CMR method is cubic in case of single zeros.

(viii) The parallel-CMR method

From eq.(4.1.7), we get

$$\tilde{z}_i = z_i - \frac{N(Q_i^{1/N} - 1)}{\tilde{s}_{1,i}(Q_i - 1)}$$

with $$Q_i = \left(\frac{N}{\tilde{\mu}_i} - 1\right) \Big/ (N-1).$$

Similar to (vii), we have

$$\begin{aligned}\tilde{z}_i &= \alpha_i - \frac{1}{2}\left[\frac{(N-5)\tilde{A}^2}{3(N-1)} - \tilde{B}\right]\delta_i^3 + O(\delta_i^4) \\ &= \alpha_i + O(\delta_i^4).\end{aligned}$$

Therefore, the convergence rate of the parallel-CMR method is fourth order in case of all single zeros.

Because of the great similarity between the pseudo deflation method and the sequential algorithms, the pseudo deflation has a potential to capture the rich knowledge from the classical theory.

4.3 Numerical Results

In [7], T.L. Freeman did a survey on several parallel algorithms which have orders of convergence two, three and four. According to this survey, the fourth order algorithm that he examined is not robust, with many failures to converge. It turns out that a proof of locally convergence does not imply that the algorithm has a good overall performance.

We implemented the pseudo deflation method in APL (Appendix A). The floating point precision that we used is double precision. For all the experiments below, the iterations were stopped when max| $P(z_i)$ | < 1E-10 for i = 1, 2, ... N. The initial approximations were selected to be symmetrically distributed in a circle (see the Section 3.4).

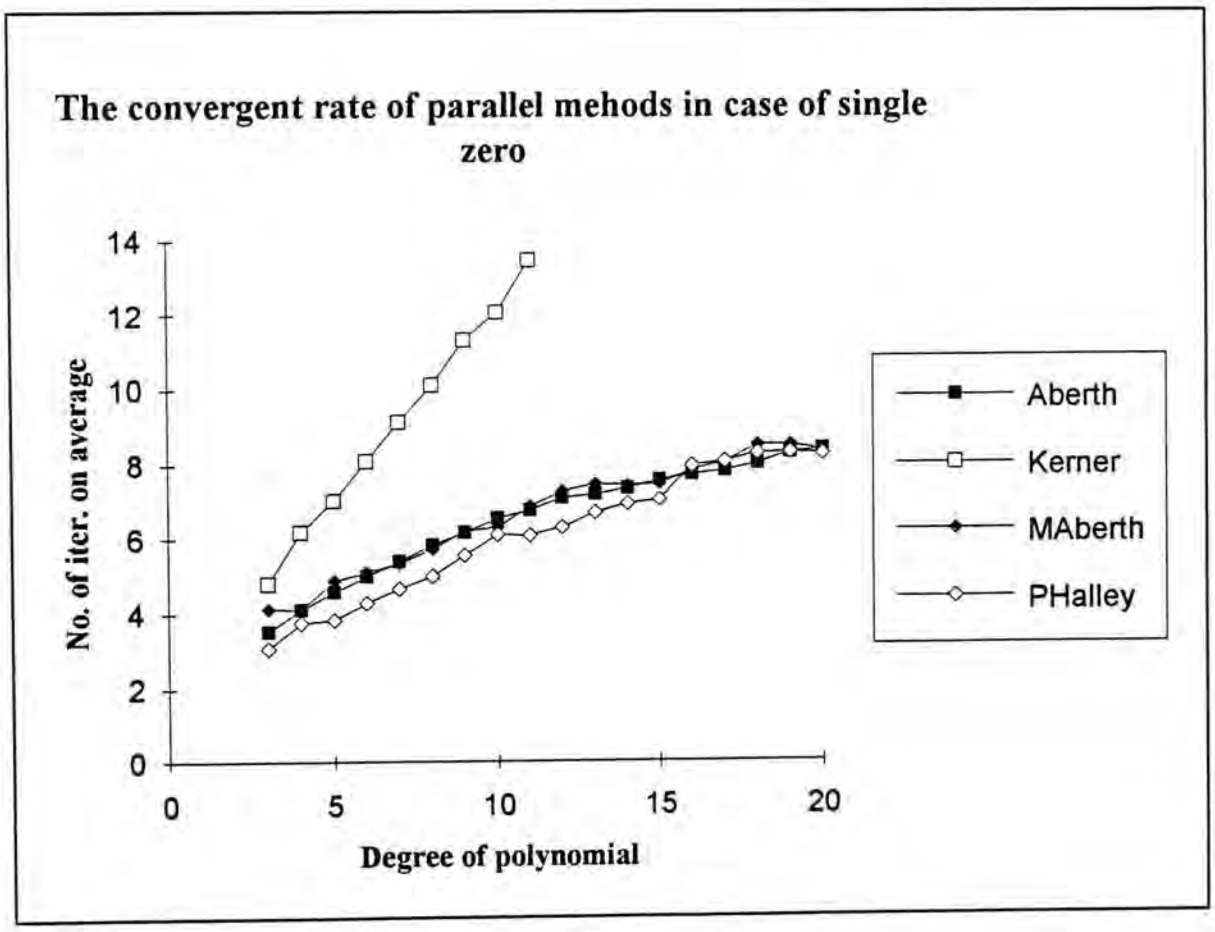


Fig. 4.1

In the first experiment, we randomly generated a wide range of degree of polynomials for our test data. Each polynomial had all distinct zeros which were in the range of $0 + 0i$ to $1 + 1i$. We used a hundred sample polynomials for each degree and took the average values at each measurement. Fig.4.1 shows the results of the convergence on several parallel iterative methods. The Durand-Kerner method had the slowest convergence rate among all the methods. The average number of iterations was increased almost linearly against the degree. The rest of the methods required more or less the same number of iterations although they had different local convergence rate. The convergence rate was sub-linearly increased against the degree. We may conclude that the rest of the methods have better performances than the Durand-Kerner method for large degree. The parallel Halley method converged in all he sample polynomials. It illustrated the robustness of this algorithm although it took longer computing time on each iteration than the Aberth method. However, it may be better than the Aberth method if we have good initial approximations in some applications.

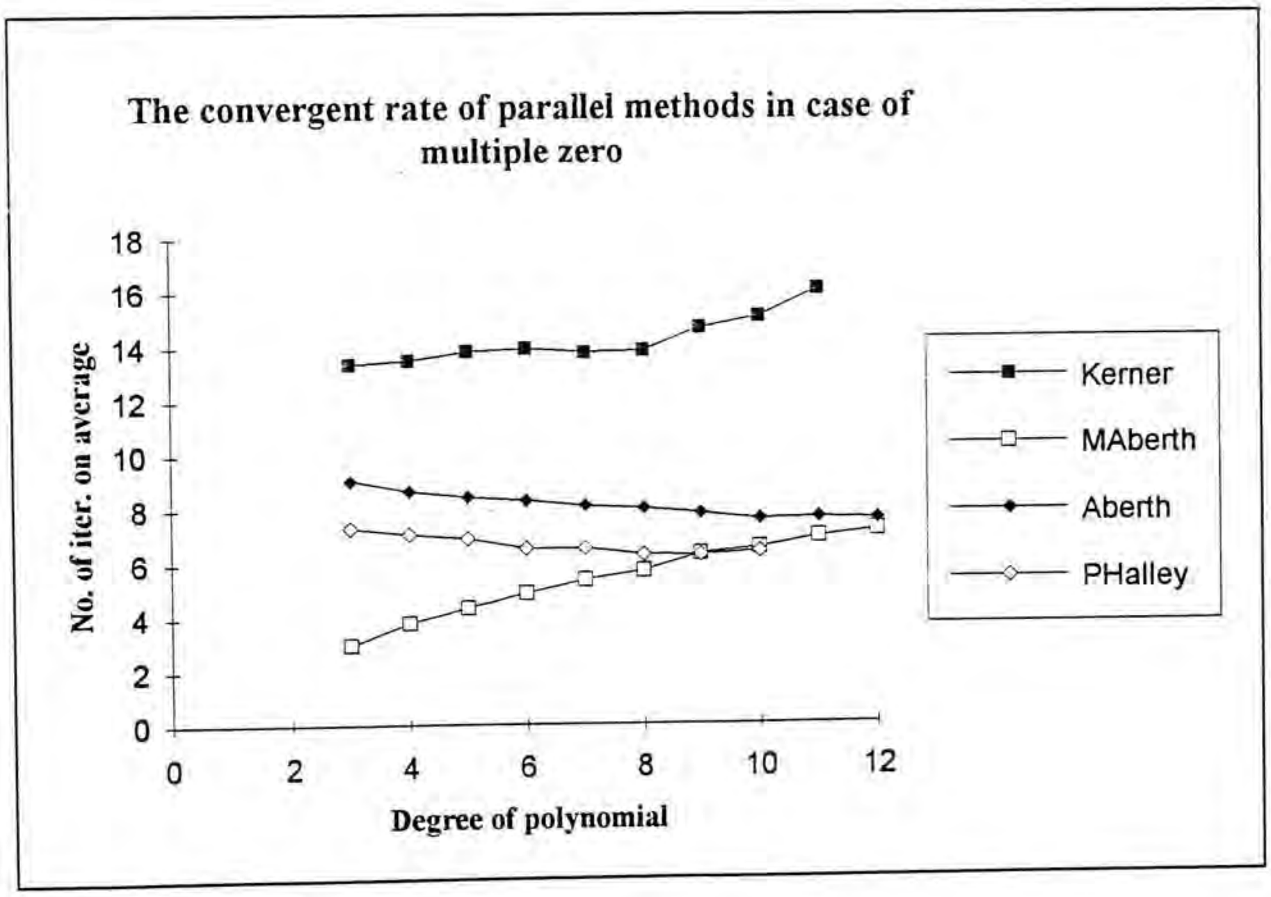


Fig. 4.2

In the second experiment, we randomly generated our test polynomials similar to the first experiment, except each polynomial had exactly one double zero and the rest of zeros were all distinct. As we expected, the Durand-Kerner method and the Aberth method converged slowly in case of multiple zero, as shown in Fig.4.2. Since the double zero dominated the longest convergent time, the number of iterations was more or less independant of the degree. The modified Aberth method tackled this problem in this case. Compare with Fig.4.1, we got a similar curve of the modified Aberth method as if the polynomials had only single zeros.

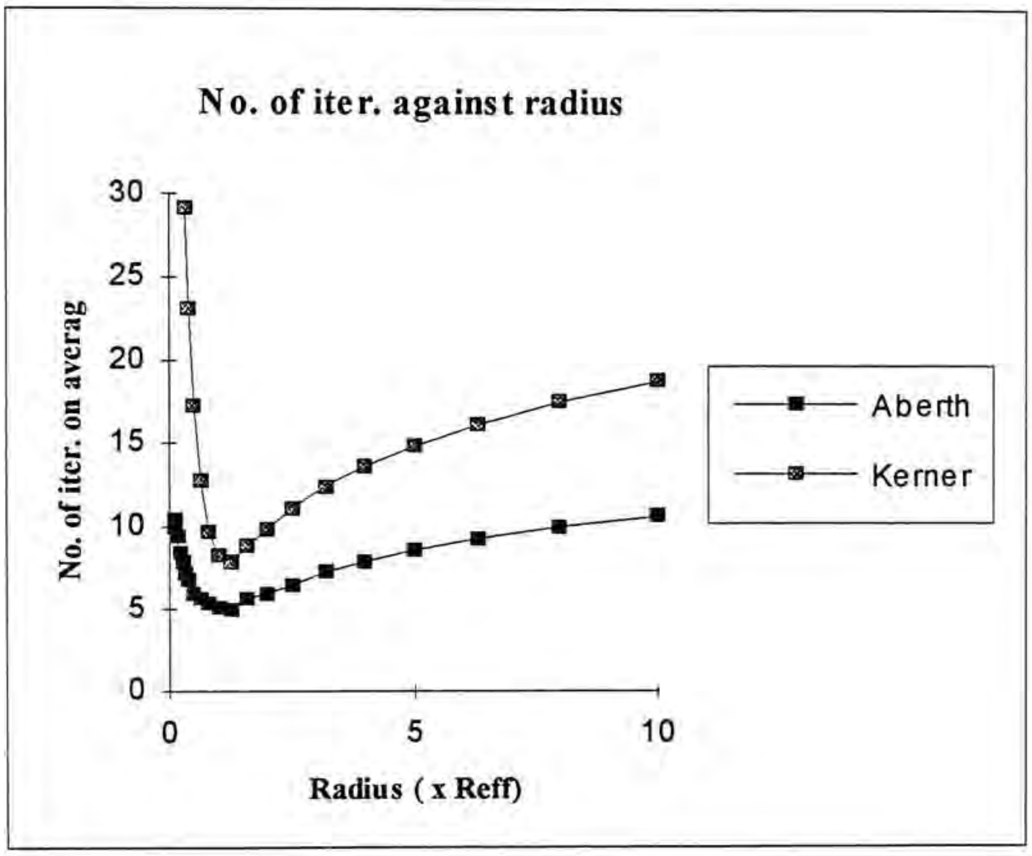


Fig. 4.3

For the choice of the initial approximations, we have discussed the effective radius in Section 3.4. We pointed out that the effective radius can be calculated more easily compared with the Guggenheimener's procedure. Moreover, it also exhibits a good performance of convergence. In the third experiment, we examined the number of iterations against a range of radius relative to the effective radius. Similar to the first experiment, we randomly generated a thousand sample polynomials with degree six. The Durand-Kerner method and the Aberth method were selected in this experiment. The results are plotted as Fig.4.3. Obviously the convergence was slow while the radius was too large such that the initial approximations were far from the zeros. We observed that the convergence was also slow while the radius was too small. In the Durand-Kerner method, it was because the iterates repelled each other in the first iteration such that the iterates were far from the zeros in the next step. In the Aberth method, we noted that the iterates were improved very little in each iteration as if the centriod attracted them. Fig.4.4 showed the same result with log scale in x axis. It is interesting that the optimum radius was near the effective radius in both methods in this experiment.

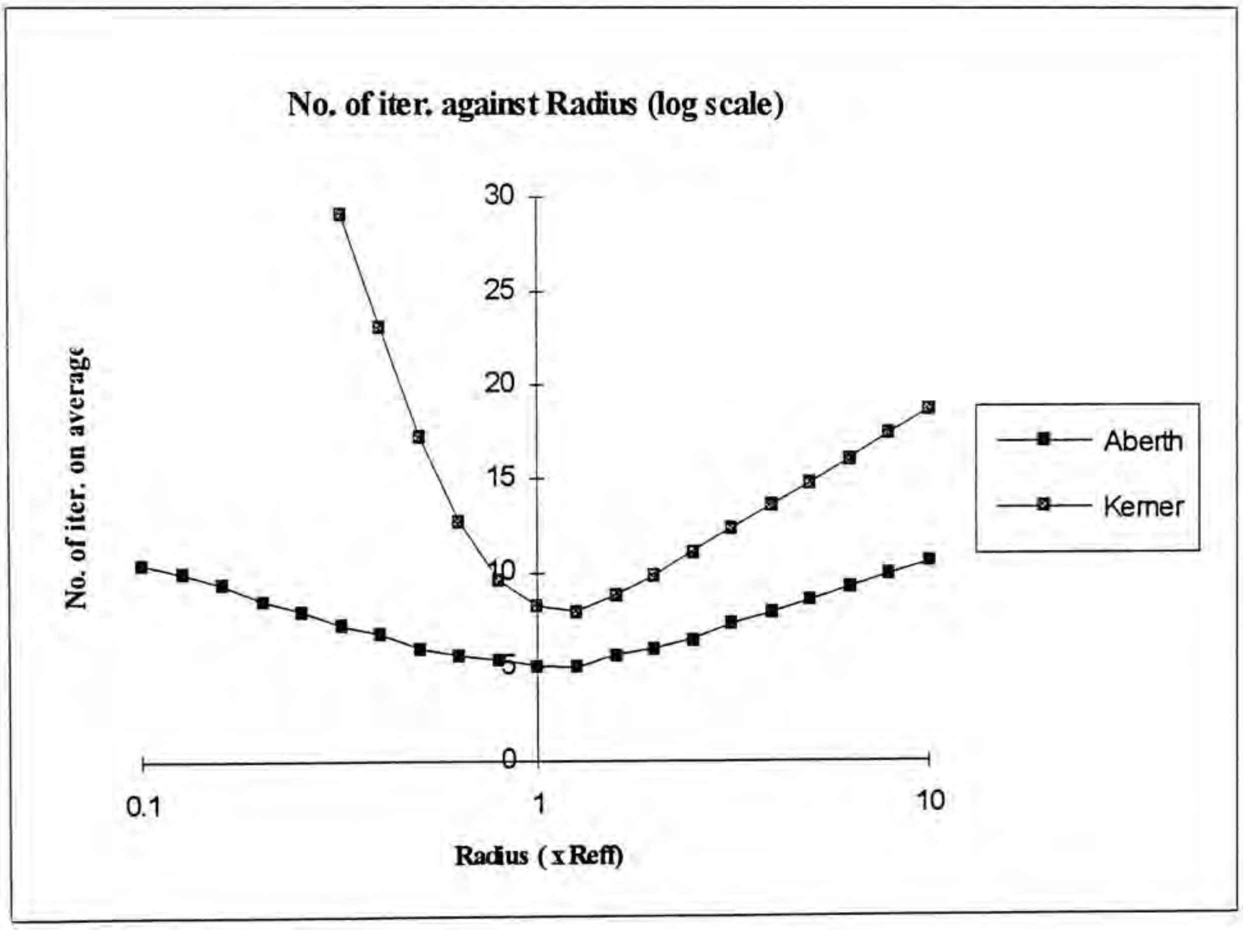


Fig. 4.4

CHAPTER 5. PARALLEL DEFLATION

In section 4, we have discussed a family of parallel algorithms which can approximate all the zeros simultaneously without an explicit deflation. However, the convergence rate of each zero may be different. For example, the convergence rate of a multiple zero is much slower that a single zero by using the Aberth method. It is wiser for us to explicitly deflate the polynomial if several iterates have been accurate enough in order to reduce the problem size and make the rest of the iterates converge more quickly. Conventionally, synthetic division is used for deflation. However, synthetic division is a sequential process and difficult to be parallelized on a parallel machine. In this section, we propose an algorithm for parallel deflation by utilizing the Fourier Transform.

5.1 The Algorithm

First, let us take an example to illustrate how the polynomial multiplication can be performed by Fast Fourier Transform, which has been mentioned in many textbooks (see [2] for example). Suppose we intent to multiply the two polynomials

$$(2z+1) \qquad \text{and} \qquad (3z+2).$$

Step 1: Take the FFT on coefficients of both polynomials:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 1+2i \\ -1 \\ 1-2i \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}\begin{pmatrix} 2 \\ 3 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 2+3i \\ -1 \\ 2-3i \end{pmatrix}.$$

Step 2: Componentwise multiply the corresponding elements of the resulting sequences:

$$\begin{pmatrix} 3 \\ 1+2i \\ -1 \\ 1-2i \end{pmatrix} \times \begin{pmatrix} 5 \\ 2+3i \\ -1 \\ 2-3i \end{pmatrix} = \begin{pmatrix} 15 \\ -4+7i \\ 1 \\ -4-7i \end{pmatrix}.$$

Step 3: Take the inverse FFT. The resulting sequence of numbers are the coefficients of the product polynomial:

$$\frac{1}{4}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}\begin{pmatrix} 15 \\ -4+7i \\ 1 \\ -4-7i \end{pmatrix} = \begin{pmatrix} 2 \\ 7 \\ 6 \\ 0 \end{pmatrix} \qquad \text{or} \qquad 2+7z+6z^2.$$

Generally, the Fourier Transform cannot be applied for polynomial division since the result may not be a polynomial. However in deflation process, we assume that the guess z_o is sufficiently close to the zero and $(z - z_o)$ is assumed to be a factor of the original polynomial. So we propose a method of parallel deflation by reversing the steps of above:

Step 1. Before iteration, compute the FFT of the original polynomial. This produces a sequence $\{p_0, p_1, p_2, \cdots, p_N\}$ where N is the degree of polynomial.

Step 2. Suppose we want to deflate $(z-\alpha_1)(z-\alpha_2)(z-\alpha_3)\cdots(z-\alpha_r)$ simultaneously, compute the FFT of each factor and componentwise multiply each corresponding elements. This produces a sequence $\{d_0, d_1, d_2, \cdots, d_N\}$.

Step 3. Compute the division p_j/d_j for $j = 0,1,2, \ldots N$.

Step 4. Compute the inverse FFT of the sequence $\{p_0/d_0, p_1/d_1, p_2/d_2, \cdots p_N/d_N\}$. The resulting sequence of numbers are the coefficients of the deflated polynomial.

By using butterfly processors, it takes only O(log$N$) time to perform deflation, compared with linear time in tradition.

5.2 The Problem of Zero Component

One of the problems in using this method is that if the original polynomial contains a zero which is a root of unity, w^j, where $w = e^{2\pi i/(N+1)}$, the p_j and d_j will become zero. The resulting p_j/d_j is undetermined. Fortunately, the missing component can be computed by the fact that since the deflated polynomial has a degree (N–1), the N-th coefficient of the deflated polynomial should be zero. Therefore,

$$\sum_{k=0}^{N}\left(\frac{p_k}{d_k}\right)w^k = 0$$

where $w = e^{2\pi i/(N+1)}$, and we have

$$\frac{p_j}{d_j} = \frac{-1}{w^j}\sum_{k\neq j}^{N}\left(\frac{p_k}{d_k}\right)w^k .$$

The problem is then solved although more effort will be added in this special case.

5.3 The Problem of Round-off Error

Due to round-off errors, the sought-after zero α_o is different from the real zero by ε. The FFT of (z - α_o) distributes this error to each component of the resulting vector. It makes the error analysis difficult.

We implemented this algorithm by using APL (Appendix A). Several polynomials were tested and the results are listed in Table 5.1. By this experiment, we may conclude that the method is quite acceptable.

Tested Polynomial	Computed zeros	Actual zeros	Deflated polynomial by using proposed parallel deflation scheme	Actual deflated polynomial
$z^4 + 4$	$1+i$, $1-i$	$1+i$, $1-i$	$z^2 + 2z + 2$	$z^2 + 2z + 2$
$z^4 + 4$	$1.00001+1.00001i$, $0.99999-0.99999i$	$1+i$, $1-i$	$z^2 + (1.999999998-5.024390237\text{E-}5\,i)z + (1.999999997-6.048780477\text{E-}6\,i)$	$z^2 + 2z + 2$
$z^3 - 76z^2 - 25z + 1900$	76.001	76	$z^2 + (4.328167807\text{E-}6)z - 24.999999858$	$z^2 - 25$
$(z - 2)^{10}$	nine (2.00001)'s	nine 2's	$z - 1.999910005$	$z - 2$

Table 5.1 Numerical results of proposed parallel deflation scheme.

CHAPTER 6. HOMOTOPY ALGORITHM

6.1 Introduction

Homotopies are a part of topology and have been applied for solving eigenvalue problems and solving polynomial systems [15,16,17,23] . In this section, we give a brief introduction for only considering the application on the polynomial root-finding problem. See [23] for more general descriptions and more applied areas. The homotopy that we will consider is:

$$H(z,t)=(1-t)Q(z)+tP(z) \tag{6.1.1}$$

where *P*(*z*) is a complex polynomial to be solved:

$$P(z)=z^N+c_{N-1}z^{N-1}+\cdots+c_0,$$

Q(*z*) is a predefined polynomial with the same degree such that its zeros are all known:

$$Q(z)=z^N+q_{N-1}z^{N-1}+\cdots+q_0,$$

and *t* is a real parameter varied from 0 to 1. At a fixed *t*, *H*(*z*,*t*) is also a polynomial with the same degree. As *t* is changed from 0 to 1, each zero of *H*(*z*,*t*) will form a root-path that starts from a known value to a unknown solution we want to solve (Fig. 6.1). The idea of homotopy method is that we follow the root-path to get the solution. Since the root-path is continuous, it makes the path-tracing possible. Moreover, the Transversality theorem from differential topology [23] says that the curves will be smooth, without bifurcations if *Q*(*z*) is choosen at random. This property makes the homotopy algorithm more practical because prediction techniques can be applied. For the parallel aspect, since each path can be followed individually without any communication overhead, it is also attractive.

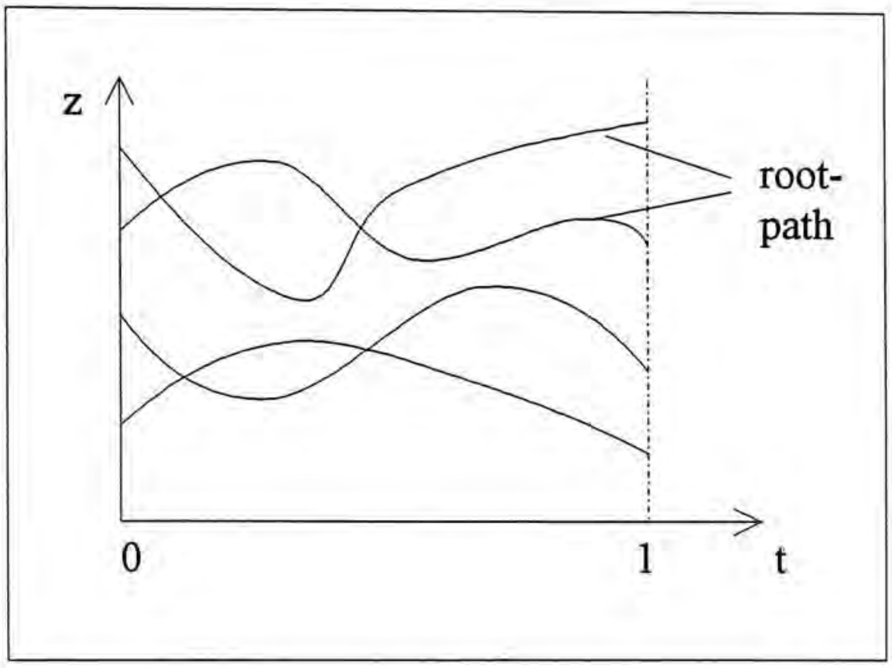


Fig.6.1.

6.2 Choosing Q(z)

In order to shorten the root-path, the zeros of $Q(z)$ should be as close to the zeros of $P(z)$ as possible. The circumstance is the same as the discussion of Section 3.4. Therefore, we suggest the following $Q(z)$:

$$Q(z) = \prod_{j=1}^{N}(z - z_j^{(0)}) = \prod_{j=1}^{N}\left\{z - \left[A + R_{eff} \cdot \exp(2\pi ij / N)\right]\right\} \tag{6.2.1}$$

where

$$i = \sqrt{-1}$$

$$A = \frac{-c_{N-1}}{N}$$

$$R_{eff} = \left(-P(A)\right)^{1/N}.$$

Obviously the centroid of $Q(z)$ is equal to that of $P(z)$, i.e., A. It implies that the centroid of $H(z,t)$ remains constant against t. Beside this, since

$$Q(A) = \prod_{j=1}^{N} \left\{ A - \left[A + R_{eff} \cdot \exp(2\pi ij/N) \right] \right\}$$

$$= R_{eff}{}^{N} \cdot (-1)^{N} \cdot \prod_{j=1}^{N} \exp(2\pi ij/N)$$

$$= -P(A) \cdot (-1)^{N} \cdot (-1)^{N+1}$$

$$= P(A).$$

Hence,

$$H(A,t) = (1-t)Q(A) + tP(A)$$

$$= (1-t)P(A) + tP(A)$$

$$= P(A).$$

Therefore, the effective radius of *H(z,t)* is also invariant with *t*.

6.3 Arclength Continuation Method

In this section, we describe an arclength continuation method for path-following [3]. Instead of parametrizing *z* in term of *t*, we use an arclength parameter *s*. The arclength condition is:

$$\|\dot{z}(s)\|^2 + |\dot{t}(s)| = 1. \qquad (6.3.1)$$

By differentiating eq. (6.1.1) with respect to *s*, we have

$$H_z \cdot \dot{z}(s) + H_t \cdot \dot{t}(s) = 0 \qquad (6.3.2)$$

or

$$\dot{z}(s) = -H_z^{-1} H_t \cdot \dot{t}(s)$$

where

$$H_z = (1-t)Q'(z) + tP'(z)$$

$$H_t = P(z) - Q(z).$$

Let

$$v = -H_z^{-1} H_t.$$

By solving eq.(6.3.1) and eq.(6.3.2), we have

$$\dot{t}(s) = \pm \frac{1}{\sqrt{1 + \|v\|^2}} \tag{6.3.3}$$

$$\dot{z}(s) = v \cdot \dot{t}(s). \tag{6.3.4}$$

By the fundamental theorem of Algebra, there can only exist one zero in each root-path for a fixed t, hence there does not exist a turning point in the path. Therefore, we can alway choose the positive value of $\dot{t}(s)$. Therefore, we can compute the unit tangent vector $(\dot{z}_0, \dot{t}_0)$ at a fixed point (z_0, t_0) and predict the next step (z_1', t_1') using the Euler prediction:

$$\begin{pmatrix} z_1' \\ t_1' \end{pmatrix} = \begin{pmatrix} z_0 \\ t_0 \end{pmatrix} + h \begin{pmatrix} \dot{z}_0 \\ \dot{t}_0 \end{pmatrix} \tag{6.3.5}$$

where h is a stepsize parameter, as shown in Fig.6.2. By using the iterative method such as the Newton method and using z_1' as an initial guess, the solution path can be corrected to (z_1, t_1). The selection of stepsize h could be adaptive for efficiency. For example, if the previous prediction is fine, we may double h at the next step, and half h if the previous prediction is poor. Of course, we can adjust the stepsize in a more intelligent way.

6.4 The bifurcation problem

The bifurcation (see Fig.6.3) can occur if and only if $H(z,t)$ has a multiple root at some t_b, i.e., $H_z(z_b, t_b) = 0$. We observed that the bifurcation points often occur when the roots of $P(z)$ and $Q(z)$ preserve some symmetry. In the same way as the Aberth method suffers from this symmetric problem [1], the bifurcation points produce many problems:

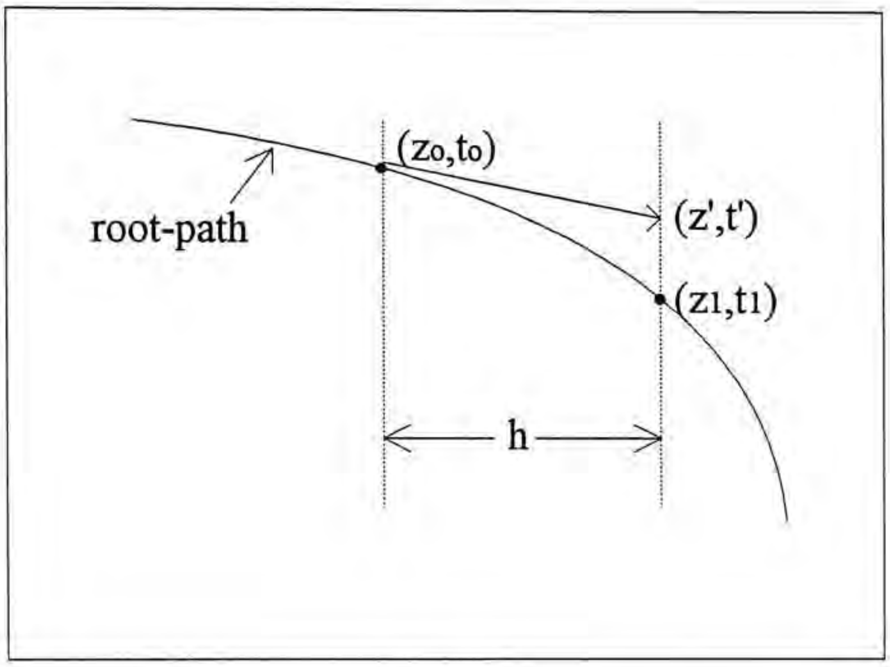


Fig. 6.2

(1) Path-jumping problem

By using the root-finding method for path-following, it has a possibility to converge to another root, especially near the bifurcation point.

(2) Path-stopping problem

Near the bifurcation point, $\dot{t}$ is very small. Hence the stepsize of t is small, too. Many efforts will be added in order to converge to the bifurcation point. Unfortunately, it is almost a must if we want to identify the bifurcation point.

(3) Lost-tracking problem

If we over-jump the bifurcation point with a larger stepsize, the iteration will probably not converge because of the symmetry. Although we may not intend to over-jump the bifurcation point on purpose in most cases, it could have occurred before we know that there is a bifurcation point.

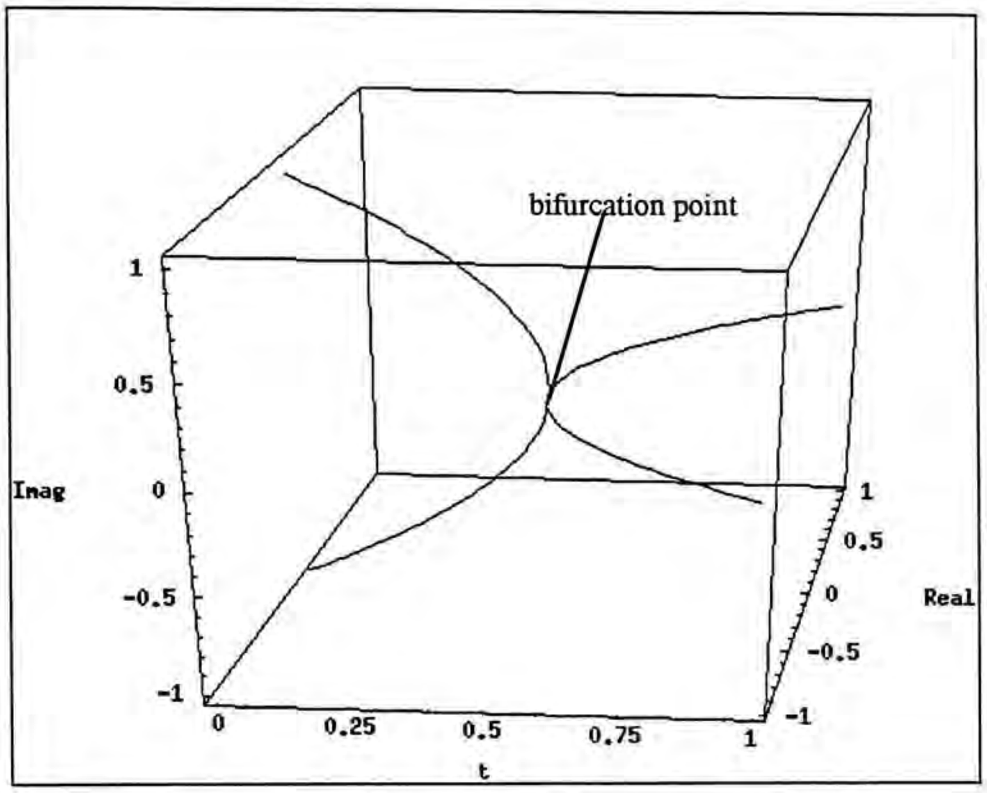


Fig. 6.3

6.5 The suggested improvement

To avoid the path-jumping problem described in the previous section, one of the suggestions is to utilize the pseudo-deflation method for path-following instead of the classical iterative method. We implemented a prototype program in APL (Appendix A). The program uses the Euler method for prediction and the parallel-Halley method for correction at each step. Several polynomials were tested. We report that the program will converge successfully in general. Also, the program always needs only one or two iterations to converge at each step. However, the investigation is preminary and it is difficult to compare with the pseudo-deflation method in current state. Any further improvement and comparsion are beyond the scope of this thesis.

CHAPTER 7. CONCLUSION

In [22], Saad concluded that, "It is interesting to observe that the successful ideas in parallel numerical methods have often been derived from existing techniques that are either adapted or slightly modified ...... The search for parallelism has forced researchers to take a second look at many old techniques, sometime resulting in remarkable success." This is the case of pseudo-deflation method. In this thesis, the general idea of the pseudo-deflation method have been presented. The advantage of this method is that it is reliable and well understood. According to this method, we parallelized a family of existing sequential algorithms. All these algorithms were proven to be locally convergent. Moreover, numerical experiments have shown that they are robust. Since the connection between the sequential algorithms and their parallel counterpart has been established, we expect that more classical theory and techniques will be captured by this new branch of parallel algorithms in the future. For example, the idea of the effective radius have been added to the parallel algorithms in considering the choice of initial approximations.

One of the challenges in the future may be the cases of cluster zeros and multiple zeros. We suggest that the two cases should not be considered separately because we observe that the cluster zeros often create difficulties to the root-finding methods. Moreover, the current technique of multiplicity estimation in classical theory has many pitfalls, as we have mentioned before. The searching for better estimation is needed. In case of multiple zeros, most of the existing parallel algorithms, which are modified from the Durand-Kerner method or the Aberth method, often use the grouping technique [8,18]. However, this technique is time consuming and may not be competitive with the original algorithms that assume all the zeros are simple.

The research of the homotopy method is active. In this thesis, we have presented the application of this method in parallel root-finding problem. The key problem of this method is the bifurcation problem. Without the bifurcation, this method is actually a globally convergent algorithm. In our case, multiple bifurcation occurs whenever $H(z,t)$ has multiple zero(s) at some t. Finally, we have suggested the improvement for solving the path-jumping problem.

REFERENCES

[1] O. Aberth. *Iteration methods for finding all zeros of a polynomial simultaneously*, Math. Comp. 27(22) (1973) pp339-344.

[2] Selim G. Akl. The Design and Analysis of Parallel Algorithms, Prentice-Hall International Editions (1989) pp232-233.

[3] Tony F. Chan. *Newton-like Psuedo-arclength Methods for Computing Simple Turning Point*, SIAM J. Sci. Stat. Comput. Vol. 5, No.1, March 1984, pp135-148

[4] Tien Chi Chen. *Global Iterative Convergence to Zeros in the Presence of Clusters*, Proc. Intern. Symp., Taipei, Taiwan (1988) pp. 270-276.

[5] Tien Chi Chen. *Iterative zero-finding revisited*, pp. 583-590 in W. L. Hogarth and B. J. Noye (Eds.), Computational Techniques and Applications: CTAC-89 (Proc. Computational Techniques and Applications Conference, Brisbane, Australia, July 1989), Hemisphere Pub. Corp. (New York 1990).

[6] Tien Chi Chen. *Globally Convergent Polynomial Iterative Zero-Finding using APL*, presented at APL92 International Conference, St. Petersburg, Russia, July 1992.

[7] M. Cosnard and P. Fraigniaud. *Finding the roots of a polynomial on an MIMD multicomputer*, Parallel Computing 15 (1990) pp.75-85.

[8] P. Fraigniaud. *The Durand-Kerner polynomial root-finding method in case of multiple roots*, BIT 31 (1991) pp. 112-123.

[9] T.L. Freeman. *Calculating polynomial zeros on a local memory parallel computer*, Parallel Comput. 12 (1989) pp. 351-358.

[10] H. Guggenheimer. *Initial Appoximations in Durand-Kerner's Root Finding Method*, BIT 26 (1986) pp. 537-539.

[11] E. Hansen and M. Patrick. *Estimating the multiplicity of a root*, Numerishe Math., vol. 27 (1976), pp. 121-131.

[12] E. Hansen and M. Patrick. *A family of root finding methods*, Numerishe Math., vol. 27 (1977), pp. 257-269.

[13] Immo O. Kerner. *Simultaneous Displacement of Polynomial Roots if real and simple*, Comm. ACM 9(4), April (1966), pp. 273.

[14] Göran Kjellberg. *Two observations on Durand-Kerner root-finding method*, BIT 24 (1984) pp556-559.

[15] T. Y. Li and Tim Sauer. *Regularity Results for Solving Systems of Polynomials by Homotopy Method*, Numerishe Math., vol. 50 (1987), pp. 283-289.

[16] T. Y. Li and N.H. Rhee. *Homotopy Algorithm for Symmetric Eigenvalue Problems*, Numerishe Math., vol. 55 (1989), pp. 265-280.

[17] T.Y. Li, Z.G. Zeng and L. Cong. *Solving Eigenvalue Problems of Real Nonsymmtric Matrices with Real Homotopys*, SIAM J. Numer. Anal., vol 29, No. 1, Feb (1992), pp. 229-248.

[18] T. Miyakoda. *Iterative methods for multiple zeros of a polynomial by clustering*, J. Comput. Appl. Math. 28 (1989) pp315-326.

[19] Y. Nagashima, Y. Kanda and H. Nagashima. *Improvement on Aberth's method for choosing initial approximations to zeros of polynomial*, IEE Proceeding, vol.136, Pt. E, No.2(1989), pp. 101-106.

[20] Clifford A. Pickover. *A Note on Chaos and Halley's Method*, Comm. ACM 31(11), Nov. (1988), pp. 1326-1329.

[21] A. Ralston and P. Rabininowitz. A First Course in Numerical Analysis, 2nd. Ed., McGraw-Hill (New York 1978). pp371, 372.

[22] Y. Saad. *Krylov Subspace methods on Supercomputers*, SIAM J. Sci. Stat. Comput., vol. 10, No. 6, Nov. 1989, pp1200-1232.

[23] L. T. Watson, S.c. Billups and A. P. Morgan. *Algorithm 652: HOMPACK: A suite of codes for globally convergent homotopy algorithms*, ACM Trans. Math. Software, vol. 13 (1987), pp. 281-310.

[24] J.H. Wilkinson. The Algebraic Eigenvalue Problem, Clarendon Press Oxford 1965, pp464-465, pp475-476.

APPENDIX A - PROGRAM LISTING

A-1 datagen.cc

```
// datagen.cc : This program is used to generate a bitmap file
//           according to a description file.
// by Danny Luk. 21/1/93
#include <stdio.h>
#include <stdlib.h>
#include <stream.h>
#include <complex.h>
#include <rand48.h>

#define             MAXDEGREE   20
#define             tolrences   1.4E-4
#define             PI2         6.28318530718
unsigned int        width,height;

double        real_min,real_max,imag_min,imag_max;
int           max_iterations;
int           method;
int           scheme;
char          outfile[25];
int           color_cycle;
int           degree;
complex       C[MAXDEGREE+1];
complex       CP[MAXDEGREE];
complex       CPP[MAXDEGREE-1];
int           perturb;

complex       formula(complex z, int k, double *fmag);

main(int argc, char *argv[])
{
      FILE          *img,*dpt;
      int           k;
      double        r,i;
      double        deltax,deltay;
      int           col,row;
      complex       z;
      double        fmag;
      char          line[100];

      if (argc != 2) {
            printf("No. of arguments not match!\n");
            exit(0);
      }

      if ((dpt = fopen(argv[1],"r")) == NULL) {
            printf("Can't read Description file!\n");
            exit(0);
      }
// Read in the description.
      fscanf(dpt,"%d %s",&width, line);
      fscanf(dpt,"%d %s",&height, line);
      fscanf(dpt,"%lf %s",&real_min, line);
      fscanf(dpt,"%lf %s",&real_max, line);
      fscanf(dpt,"%lf %s",&imag_min, line);
      fscanf(dpt,"%lf %s",&imag_max, line);
      fscanf(dpt,"%d %s",&max_iterations, line);
      fscanf(dpt,"%d %s",&method, line);
      fscanf(dpt,"%d %s",&scheme, line);
```

```
        fscanf(dpt,"%s %s",outfile, line);
        fscanf(dpt,"%d %s",&color_cycle, line);
        fscanf(dpt,"%d %s",&degree, line);

        for (k=0; k<=degree; k++)
        {
              fscanf(dpt,"%lf %lf",&r,&i);
              C[k] = complex(r,i);
        }
        fscanf(dpt,"%d %s",&perturb, line);
        printf("perturb = %d\n",perturb);
        fclose(dpt);

        if ((img = fopen(outfile,"wb")) == NULL) {
              printf("Can't open Image output file!\n");
              exit(0);
        }

        for (k=0; k<=degree-1; k++)
        {
              CP[k] = C[k+1] * (k+1);
        }
        for (k=0; k<=degree-2; k++)
        {
              CPP[k] = CP[k+1] * (k+1);
        }
        deltax = (real_max - real_min)/width;
        deltay = (imag_max - imag_min)/height;
        for (col=0; col<width; col++) {
            for (row=0; row<height; row++) {
              z = complex(real_min + col * deltax, imag_max - row *
deltay);
              k = 1;
              while (k<max_iterations) {
                 z = formula(z,k,&fmag);
                 if (fmag < tolrences) break;
                 if (perturb) {
                    double a;
                    a =  PI2 * drand48();
                    z = z + tolrences * fmag * complex(cos(a),sin(a));
                 }
                 k++;
              }
              fputc(k,img);
            }
        }
        fclose(img);
}
```

A-2 methods.cc

```
#include <stdio.h>
#include <stdlib.h>
#include <stream.h>
#include <complex.h>
#include "descriptor.h"

complex         S;
complex         MU;
int             P;

complex newton(complex z)
{
        return(z - P/S);
}

complex laguerre(complex z)
{
      return(z - degree/(1+sqrt((degree/MU-1)*((double)degree/
(double)P-1)))/S);
}

complex cmr(complex z)
{
      complex Q;
        Q = (degree/MU-1)/((double)degree/(double)P-1);

      return(z - degree*(pow(Q,(double)P/(double)degree)-1)/(Q-1)/S);
}

complex halley(complex z)
{
      return(z - 2.0/S/(1+1/MU));
}

complex formula(complex z, int k, double *fmag)
{
      int i;
      complex F,FP,FPP;
      complex z_new,F_new;
      complex U;
      complex Aeff;
      int qx,qc,W,V;


      F     = complex(0.0,0.0);
      FP    = complex(0.0,0.0);
      FPP   = complex(0.0,0.0);

      for (i=degree; i>=0; i--)
         F = F*z + C[i];
      for (i=degree-1; i>=0; i--)
         FP = FP*z + CP[i];
      for (i=degree-2; i>=0; i--)
         FPP = FPP*z + CPP[i];
      S = FP / F;
        MU = 1/(1- F*FPP/FP/FP);

      W = (int) (0.5 + real(MU));
      if (W>1 && W<degree && (((double)W) >= 2.0*fabs(imag(MU))))
           qx = W;
        else
```

```
            qx = 1;

      Aeff = -C[degree-1]/C[degree]/degree;
      U = (degree-S*(z-Aeff))/(1-S*(z-Aeff)/MU);
      V = (int) (0.5 + real(U));
      if (V>1 && V <degree && (((double)V) >= 2.0*fabs(imag(U))))
            qc = V;
      else
            qc = 1;

      switch (scheme) {
         case 1:
               P = 1;
               break;
         case 2:
               switch (k % 3) {
                  case 1: P = qx; break;
                  case 2: if (qx < P) P = qx; break;
                  case 0: P = 1; break;
               }
               break;
         case 3:
               if (k==1)
                  P = qc;
               else
                  P = qx;
               break;
         case 4:
               P = qx;
               break;
         default:
               printf("Unknown P scheme!\n");
               exit(0);
      }
      switch (method) {
         case 1: z_new = newton(z); break;
         case 2: z_new = laguerre(z); break;
         case 3: z_new = cmr(z); break;
         case 4: z_new = halley(z); break;
         default:
            printf("Unknow method!\n");
            exit(0);
      }
      F_new = complex(0.0,0.0);
      for (i=degree; i>=0; i--)
         F_new = F_new*z_new + C[i];
         *fmag = abs(F_new);
      return(z_new);
}
```

A-3 descriptor.h

```
#define             MAXDEGREE   20
extern      unsigned int    width,height;
extern      double          real_min,real_max,imag_min,imag_max;
extern      int             max_iterations;
extern      int             method;
extern      int             scheme;
extern      int             degree;
extern      complex         C[MAXDEGREE+1];
extern      complex         CP[MAXDEGREE];
extern      complex         CPP[MAXDEGREE-1];
```

A-4 Makegen

```
# Makegen - make file for generate executable file "datagen"
# This make file needs SUN C++ complier with complex library
# type "make -f Makegen" at command line
EXEC= datagen
CFLAGS= -O
LIBS=   -I/usr/CC/sun/incl -L/usr/CC/sun4/ -lX11  -lcomplex -lm -lC
OBJECTS=        methods.o   \
                datagen.o
$(EXEC):            $(OBJECTS)
                 cc -o $(EXEC) $(OBJECTS) $(LIBS)
```

A-5 newton01.dpt - Sample description file

```
512                 width
512                 height
-3.5                real_min
+3.5                real_max
-3.5                imag_min
+3.5                imag_max
200                 max_iterations
1                   method=Laguerre
1                   scheme
newton01.img        output_file
200                 color_cycle
7                   degree
-1.0   0.0
0.0    0.0
0.0    0.0
0.0    0.0
0.0    0.0
0.0    0.0
0.0    0.0
1.0    0.0
0                   perturbation

Comments:
I      F(z) = z**7 - 1
II     scheme = 1  p=1 always
              = 2  generalise Laguerre
              = 3  cluster adapted
              = 4  p=qx always
III    method = 1  Newton
              = 2  Laguerre
              = 3  Cluster Adapted
              = 4  Halley
```

A-6 display.m - Matlab M-file

```
function r = display(fname);
% display image file generated by "datagen"
% This function can only be worked on MATLAB 4.0
% By Danny Luk, 21 Jan 93
%
% Usage : display("img_file")
fid = fopen(fname,'r');
ima = fread(fid,[512,512]);
h = image(ima);
axis('square');
axis('off');
% load the file mymap.mat
load mymap;
colormap(mymap);
r = h;
```

A-7 Subroutines for the pseud-deflation method in APL

```
     ∇PITER[⎕]∇
 D←PITER;T;K
  K←0 ◇ 'POLYNOMIAL COEFFICENTS ?' ◇ C←⎕ ◇ 'INITIAL GUESSES ?' ◇ Z←⎕
 LT:'0 EXIT/1 D-K/2 ABERTH/3 MABERTH/4 PHALLEY/5 MPHALLY/6 PLAG/7 PCMR ?'
  →(0,LK,LA,LM,LH,LQ,LL,LC)[1+⎕]AS T←'K = ',⍕K←K+1
 LK:→LZ AS Z←C KERNER Z AS ⎕←T ⍝ DURAND-KENRER METHOD
 LA:→LZ AS Z←C ABERTH Z AS ⎕←T ⍝ GENERALIZED ABERTH METHOD
 LM:→LZ AS Z←C MABERTH Z AS ⎕←T ⍝ MODIFIED ABERTH METHOD
 LH:→LZ AS Z←C PHALLEY Z AS ⎕←T ⍝ PARALLEL HALLEY METHOD
 LQ:→LZ AS Z←C MPHALLEY Z AS ⎕←T ⍝ MULTIPLE PHALLEY METHOD
 LL:→LZ AS Z←C PLAG Z AS ⎕←T ⍝ PARALLEL LAGUERRE METHOD
 LC:→LZ AS Z←C PCMR Z AS ⎕←T ⍝ PARALLEL CMR METHOD
 LZ:⎕←'Z =' ◇ CX Z ◇ ⎕←'|FZ| = ',⍕CXMAG C CXPOLY Z ◇ →LT

     ∇KERNER[⎕]∇
 Z2←C KERNER Z;PS;ZJS;ZI;P;T
  PS←,⍉C CXPOLY Z ◇ Z←,⍉Z ◇ Z2←⍴1
 L:→0 IF(0≥⍴PS)
  ZJS←2↓Z AS ZI←2↑Z AS P←2↑PS
  →L4 IF(1E¯10<CXMAG P)
  →L5 AS Z2←Z2,ZI
 L4:Z2←Z2,ZI-P ZDIV CXPRO ZI CXMINUS ZJS
 L5:→L AS Z←ZJS,ZI AS PS←2↓PS

     ∇ABERTH[⎕]∇
 Z2←C ABERTH Z;P0S;P1S;ZJS;ZI;P0;P1;T
  Z2←⍴1 ◇ P0S←,⍉C CXPOLY Z ◇ P1S←,⍉(CXDIFF C)CXPOLY Z
  Z←,⍉Z
 L:→0 IF(0≥⍴P0S)
  ZJS←2↓Z ◇ ZI←2↑Z ◇ P0←2↑P0S ◇ P1←2↑P1S
  →L1 IF(0≠P0)
  →L2 AS Z2←Z2,ZI
 L1:Z2←Z2,ZI-P0 ZDIV P1-P0 ZTIMES CXSUM 1 0 CXDIV ZI CXMINUS ZJS
 L2:Z←ZJS,ZI ◇ P0S←2↓P0S ◇ P1S←2↓P1S
  →L

     ∇MABERTH[⎕]∇
 Z2←C MABERTH Z;P0S;P1S;ZJS;ZI;P0;P1;M;MI;N;T;DZI
  P0S←,⍉C CXPOLY Z ◇ P1S←,⍉(CXDIFF C)CXPOLY Z
  N←¯1+⍴M←C MULTI Z ◇ Z←,⍉Z ◇ Z2←⍴1
 L:→0 IF(0≥⍴P0S)
  T←ZJS←2↓Z ◇ ZI←2↑Z ◇ P0←2↑P0S ◇ P1←2↑P1S ◇ MI←1↑M
  →L1 IF(0≠P0) ◇ Z2←Z2,ZI ◇ →L2
 L1:→L3 IF(1=MI) ◇ DZI←CXMAG ZI CXMINUS ZJS ◇ DZI←(2,N)⍴DZI,⍳N
  T←(CX ZJS)[;,(1 0)↓(0,MI-1)↓DZI[;⍋⍉DZI]]
 L3:Z2←Z2,ZI-MI ZTIMES P0 ZDIV P1-P0 ZTIMES CXSUM 1 0 CXDIV ZI CXMINUS T
 L2:Z←ZJS,ZI ◇ P0S←2↓P0S ◇ P1S←2↓P1S ◇ M←1↓M
  →L

     ∇PHALLEY[⎕]∇
 Z2←C PHALLEY Z;P0S;P1S;P2S;ZJS;ZI;P0;P1;P2;MU;S1;S2;DZ
  P0S←,⍉C CXPOLY Z ◇ P1S←,⍉(CXDIFF C)CXPOLY Z
  P2S←,⍉(CXDIFF CXDIFF C)CXPOLY Z ◇ Z←,⍉Z ◇ Z2←⍴1
 L:→0 IF(0≥⍴P0S)
  ZJS←2↓Z AS ZI←2↑Z AS P0←2↑P0S AS P1←2↑P1S AS P2←2↑P2S
  →L1 IF(0≠P0) ◇ Z2←Z2,ZI ◇ →L2
 L1:S2←(S1 ZTIMES S1←P1 ZDIV P0)-P2 ZDIV P0
  S1←S1-CXSUM 1 0 CXDIV DZ←ZI CXMINUS ZJS
  MU←S1 ZTIMES S1 ZDIV S2←S2-CXSUM 1 0 CXDIV DZ CXTIMES DZ
  Z2←Z2,ZI- 2 0 ZDIV S1 ZTIMES 1 0 + 1 0 ZDIV MU
 L2:→L AS Z←ZJS,ZI AS P0S←2↓P0S AS P1S←2↓P1S AS P2S←2↓P2S
```

```
      ∇MPHALLEY[⎕]∇
[0]   Z2←C MPHALLEY Z;P0S;P1S;P2S;ZJS;ZI;P0;P1;P2;MU;S1;S2;DZ;PI;MI;M;N;T;DZI
[1]    Z2←ρ1 ⋄ P0S←,⍉C CXPOLY Z ⋄ P1S←,⍉(CXDIFF C)CXPOLY Z
[2]    P2S←,⍉(CXDIFF CXDIFF C)CXPOLY Z ⋄ N←¯1+ρM←C MULTI Z
[3]    Z←,⍉Z
[4]   L:→0 IF(0≥ρP0S)
[5]    T←ZJS←2↓Z ⋄ ZI←2↑Z ⋄ P0←2↑P0S ⋄ P1←2↑P1S ⋄ P2←2↑P2S ⋄ MI←1↑M
[6]    →L1 IF(0≠P0)
[7]    →L2 AS Z2←Z2,ZI
[8]   L1:→L3 IF(1=MI)
[9]    DZI←(2,N)ρ(CXMAG ZI CXMINUS ZJS),ιN
[10]   T←(CX ZJS)[;,(1 0)↓(0,MI-1)↓DZI[;⍋⍉DZI]]
[11]  L3:S2←(S1 ZTIMES S1←P1 ZDIV P0)-P2 ZDIV P0
[12]   S1←S1-CXSUM 1 0 CXDIV DZ←ZI CXMINUS T
[13]   MU←S1 ZTIMES S1 ZDIV S2←S2-CXSUM 1 0 CXDIV DZ CXTIMES DZ
[14]   Z2←Z2,ZI- 2 0 ZDIV S1 ZTIMES((1÷MI),0)+ 1 0 ZDIV MU
[15]  L2:Z←ZJS,ZI ⋄ P0S←2↓P0S ⋄ P1S←2↓P1S ⋄ P2S←2↓P2S ⋄ M←1↓M
[16]   →L

      ∇PLAG[⎕]∇
[0]   Z2←C PLAG Z;P0S;P1S;P2S;ZJS;ZI;P0;P1;P2;MU;S1;S2;DZ;N;ZQ
[1]    P0S←,⍉C CXPOLY Z ⋄ P1S←,⍉(CXDIFF C)CXPOLY Z ⋄ N←¯1+0.5×ρ,C
[2]    P2S←,⍉(CXDIFF CXDIFF C)CXPOLY Z ⋄ Z←,⍉Z ⋄ Z2←ρ1
[3]   L:→0 IF(0≥ρP0S)
[4]    ZJS←2↓Z AS ZI←2↑Z AS P0←2↑P0S AS P1←2↑P1S AS P2←2↑P2S
[5]    →L1 IF(0≠P0) ⋄ Z2←Z2,ZI ⋄ →L2
[6]   L1:S2←(S1 ZTIMES S1←P1 ZDIV P0)-P2 ZDIV P0
[7]    S1←S1-CXSUM 1 0 CXDIV DZ←ZI CXMINUS ZJS
[8]    MU←S1 ZTIMES S1 ZDIV S2←S2-CXSUM 1 0 CXDIV DZ CXTIMES DZ
[9]    ⎕←ZQ←ZSQRT((N-1),0)ZTIMES ¯1 0 +(N,0)ZDIV MU ⋄ ZQ←ZQ××CXREAL ZQ
[10]   Z2←Z2,ZI-(N,0)ZDIV S1 ZTIMES(1 0)+ZQ
[11]  L2:→L AS Z←ZJS,ZI AS P0S←2↓P0S AS P1S←2↓P1S AS P2S←2↓P2S

      ∇PCMR[⎕]∇
[0]   Z2←C PCMR Z;P0S;P1S;P2S;ZJS;ZI;P0;P1;P2;MU;S1;S2;DZ;N;Q
[1]    P0S←,⍉C CXPOLY Z ⋄ P1S←,⍉(CXDIFF C)CXPOLY Z ⋄ N←¯1+0.5×ρ,C
[2]    P2S←,⍉(CXDIFF CXDIFF C)CXPOLY Z ⋄ Z←,⍉Z ⋄ Z2←ρ1
[3]   L:→0 IF(0≥ρP0S)
[4]    ZJS←2↓Z AS ZI←2↑Z AS P0←2↑P0S AS P1←2↑P1S AS P2←2↑P2S
[5]    →L1 IF(0≠P0) ⋄ Z2←Z2,ZI ⋄ →L2
[6]   L1:S2←(S1 ZTIMES S1←P1 ZDIV P0)-P2 ZDIV P0
[7]    S1←S1-CXSUM 1 0 CXDIV DZ←ZI CXMINUS ZJS
[8]    MU←S1 ZTIMES S1 ZDIV S2←S2-CXSUM 1 0 CXDIV DZ CXTIMES DZ
[9]    Q←(¯1 0 +(N,0)ZDIV MU)÷(N-1)
[10]   Z2←Z2,ZI-(N,0)ZTIMES(¯1 0 +Q CXPOWER 1÷N)ZDIV S1 ZTIMES Q- 1 0
[11]  L2:→L AS Z←ZJS,ZI AS P0S←2↓P0S AS P1S←2↓P1S AS P2S←2↓P2S

      ∇MULTI[⎕]∇
[0]   P←C MULTI Z;CC;N;M;A;FPPZ;FPZ;CXM;FZ;S;MU;T;U;V;W;Q;K
[1]    A←-(2↑¯4↑CC)ZDIV(¯2↑CC)×N AS M←(0.5×ρ,Z←CX Z)ρN←¯1+0.5×ρCC←,⌽C←CX C
[2]    CXM←M J 0 ⋄ FPZ←(CXDIFF C)CXPOLY Z ⋄ FPPZ←(CXDIFF CXDIFF C)CXPOLY Z
[3]    ⎕←S←FPZ CXDIV 1E¯14+FZ←C CXPOLY Z AS ⎕←'S ='
[4]    ⎕←MU←T CXDIV(T←FPZ CXTIMES FPZ)CXMINUS FPPZ CXTIMES FZ AS ⎕←'MU ='
[5]    ⎕←P←1⌈W×(W<M)×W>|2×CXIMAG MU AS W←⌊0.5+CXREAL MU AS ⎕←'QX ='
[6]    P←ADJUSTP P

      ∇ADJUSTP[⎕]∇
[0]   P←ADJUSTP T;K
[1]    K←1 ⋄ P←T
[2]   L1:→L2 IF 0≠K|+/1=T ⋄ K←K+1 ⋄ →0 IF 0=+/0<T←T-1 ⋄ →L1
[3]   L2:P←Nρ1
```

A-8 Subroutines for the parallel deflation method by FFT in APL

```
 ∇PDEFLATE[⎕]∇
C2←C PDEFLATE Z;Q;INVQ;T;N;FC;M;MZ
 Q←PURIFY(,1 R○1×2÷N)CXPOWER(N|T∘.×T←¯1+⍳N←0.5×⍴,C)J 0
 INVQ←CXCONJ Q÷N ⋄ FC←Q CXMATTIMES(N,1)CXRESHAPE C ⋄ M←0.5×⍴,Z
 MZ←Q CXMATTIMES(N,M)CXRESHAPE CX(,⍉-Z),((2×M)⍴ 1 0),(2×M×N-2)⍴0
 FR←PURIFY INVQ CXMATTIMES PURIFY FC CXDIV(N,1)CXRESHAPE CXPRO MZ
 C2←FR CXDIV ¯2↑,⍉FR←CXRAVEL((N-M),1)CXRESHAPE FR

 ∇PURIFY[⎕]∇
A←PURIFY B
 A←B×(1E¯9≤|B)
```

A-9 Subroutines for the homotopy method in APL

```
 ∇HTEST[⎕]∇
D←P HTEST Z;EPS;COUNT;L;K;V;B;T;H;Q;T2;SS;Z2;H3;H4;A
 EPS←1E¯10 ⋄ SS←0.1 ⋄ M←0.5×⍴,Z
 ⎕←'Program is Running!!!!!!! Please Do NOT interrupt...'
 V←⎕CURSOR ⋄ L←0 ⋄ ⎕←Z←CX Z ⋄ T←0 ⋄ Q←ZTOPOLY Z
L1:T2←1⌊T+SS ⋄ SS←T2-T ⋄ H←(P×T2)+Q×1-T2 ⋄ K←0 ⋄ Z2←T EULER Z
L2:K←K+1 ⋄ ERR←CXMAG H CXPOLY Z2←CX H PHALLEY Z2
L4:→L2 IF(EPS≤ERR)
 ⎕←K,SS ⋄ ⎕←Z←Z2 ⋄ T←T2
 →L1 IF(T<1)

 ∇EULER[⎕]∇
Z2←T EULER Z
 Z2←Z+SS×T DZDT Z

 ∇DZDT[⎕]∇
D←T DZDT Z
 D←((Q CXPOLY Z)-P CXPOLY Z)CXDIV(CXDIFF(T×P)+Q×1-T)CXPOLY Z

 ∇CXPOLY[⎕]∇
P←C CXPOLY CXZ
 P←¯2↑C←,⍉C
L:→0 IF(0≥⍴C←¯2↓C)
 →L AS P←(¯2↑C)CXPLUS CXZ CXTIMES P

 ∇CXDIFF[⎕]∇
C2←CXDIFF C;N
 C2←CX C×,⍉(2,N)⍴⍳N←0.5×⍴C←2↓C←,⍉C
```

APPENDIX B COLOR PLATES

Fig A.0.a Default colormap

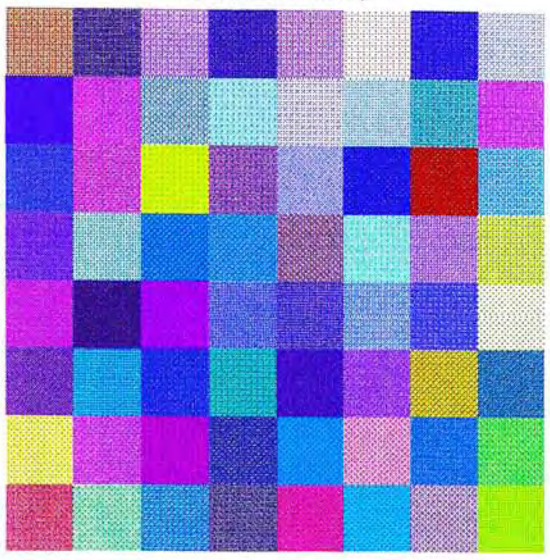


Fig A.0.b JET colormap

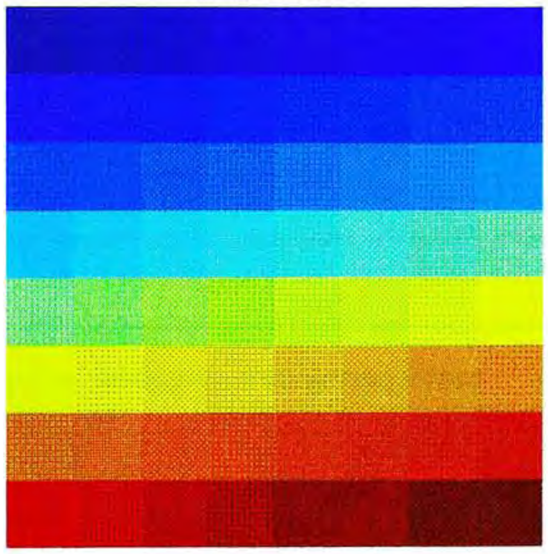
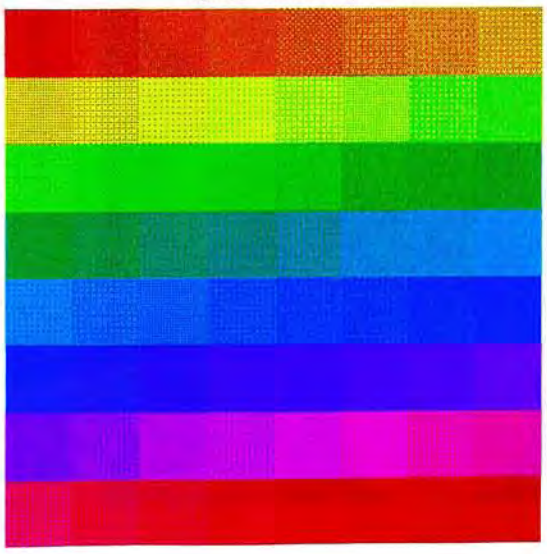


Fig A.0.c HSV colormap



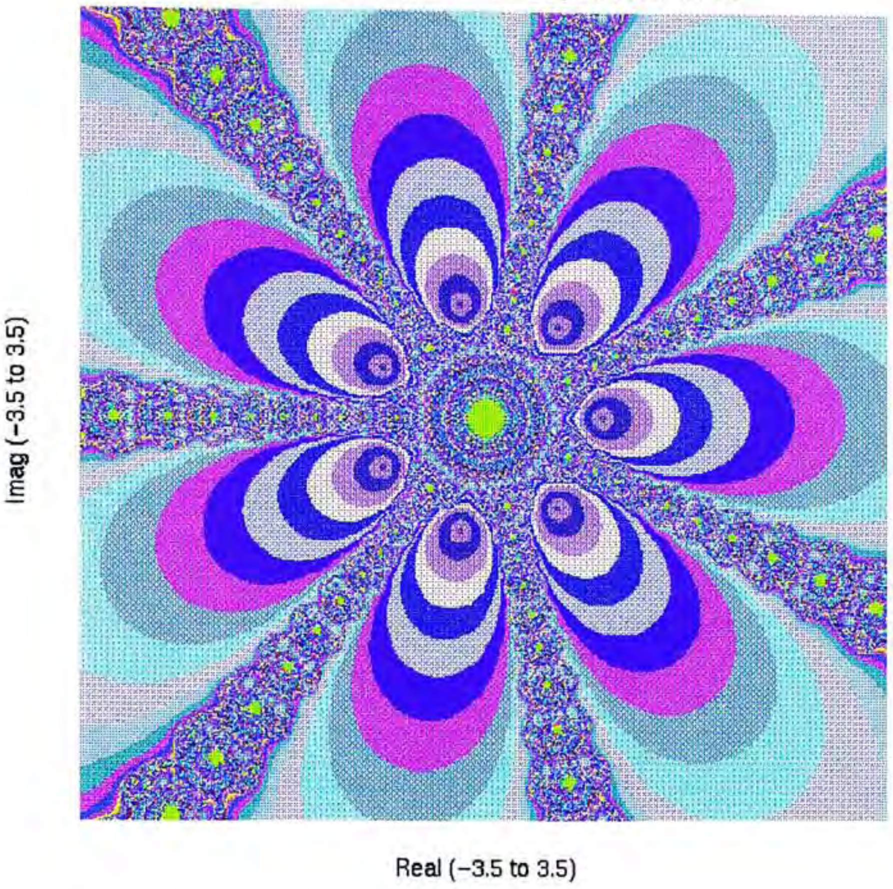


Fig A.1.a Solving z^7 − 1 = 0 by Newton method

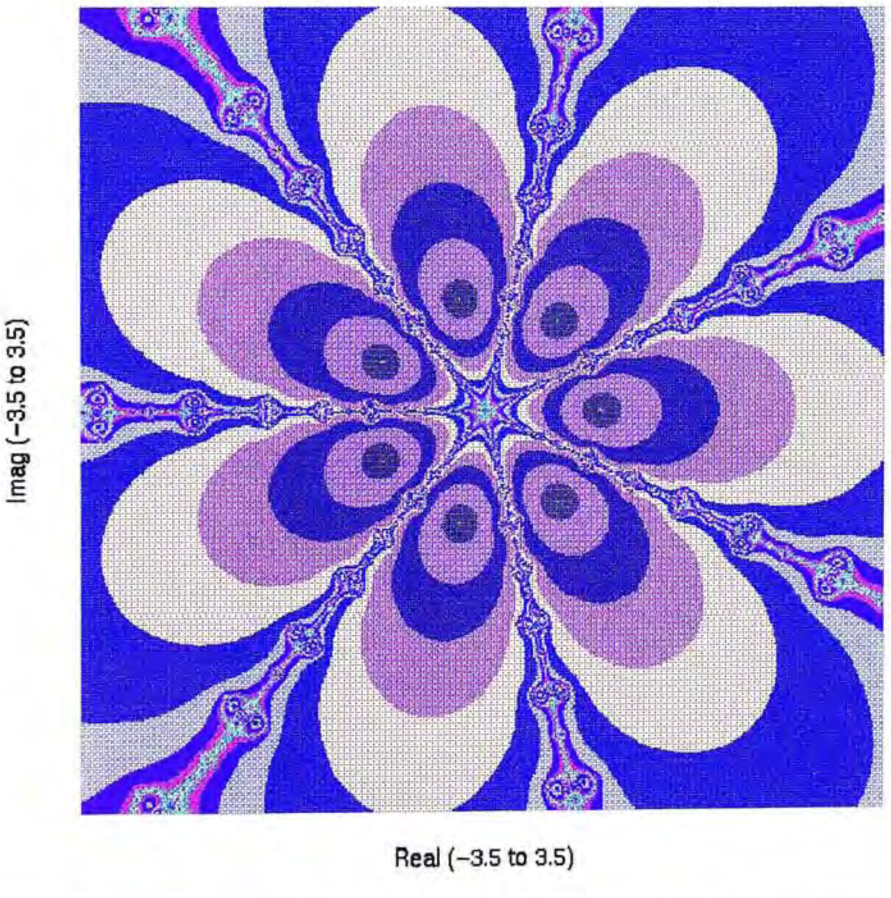


Fig A.1.b Solving z^7 − 1 = 0 by Halley method

Fig A.1.c Solving z^7 - 1 = 0 by Laguerre method

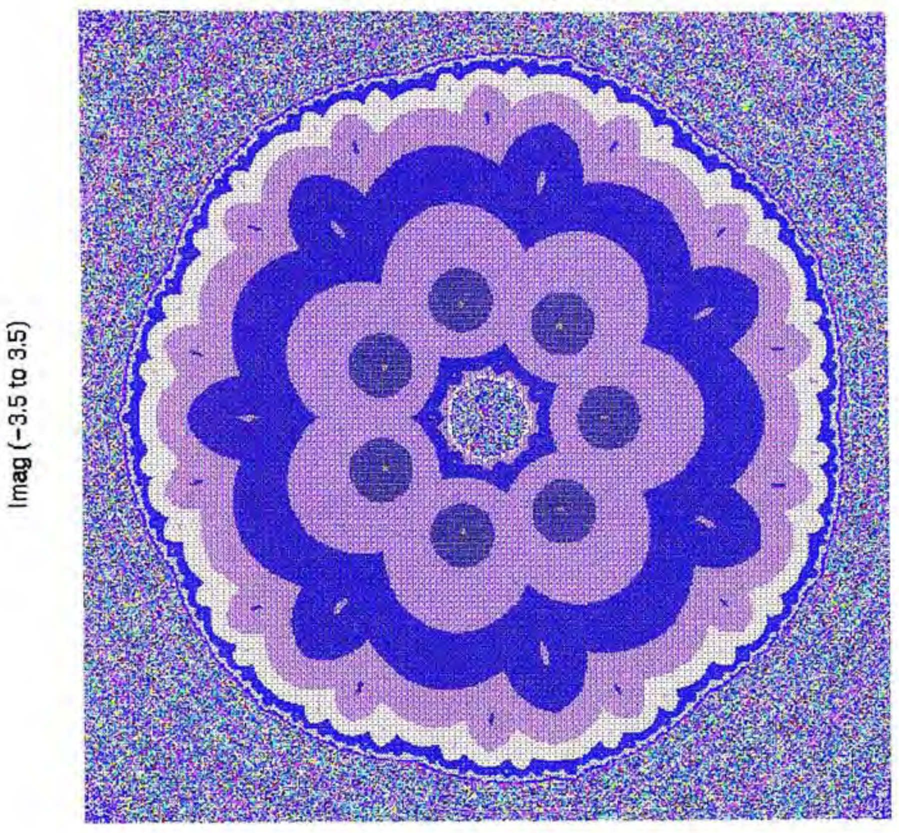


Fig A.1.d Solving z^7 - 1 = 0 by CMR method

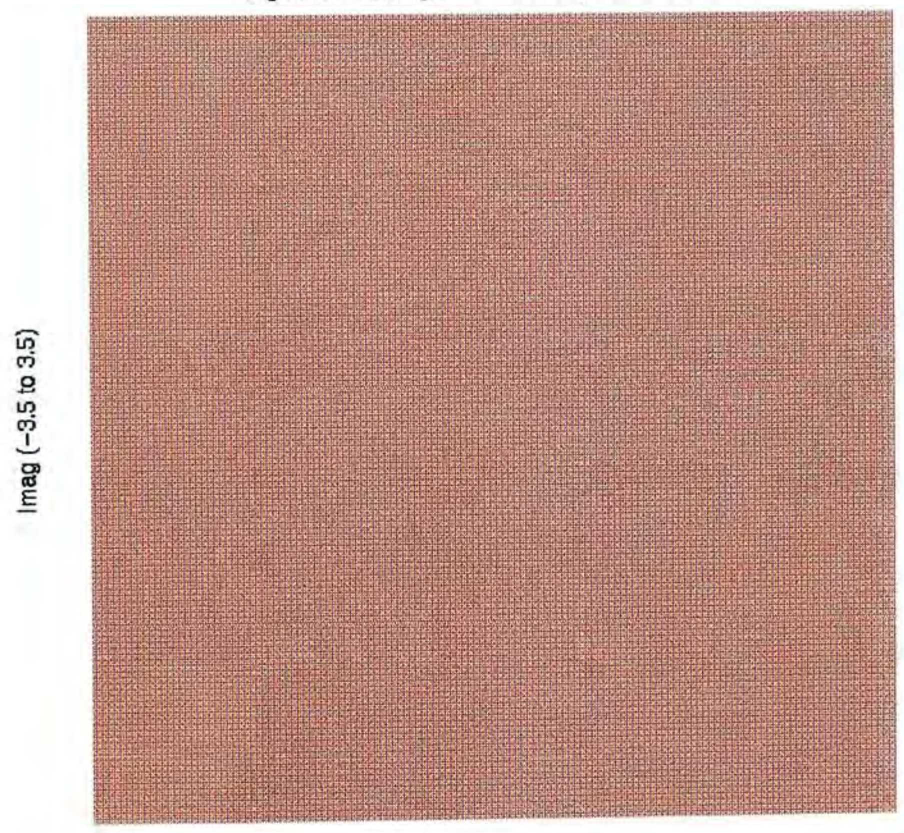


Fig A.2.a Solving z^7 − 1 = 0 by Newton method (Colormap: jet)

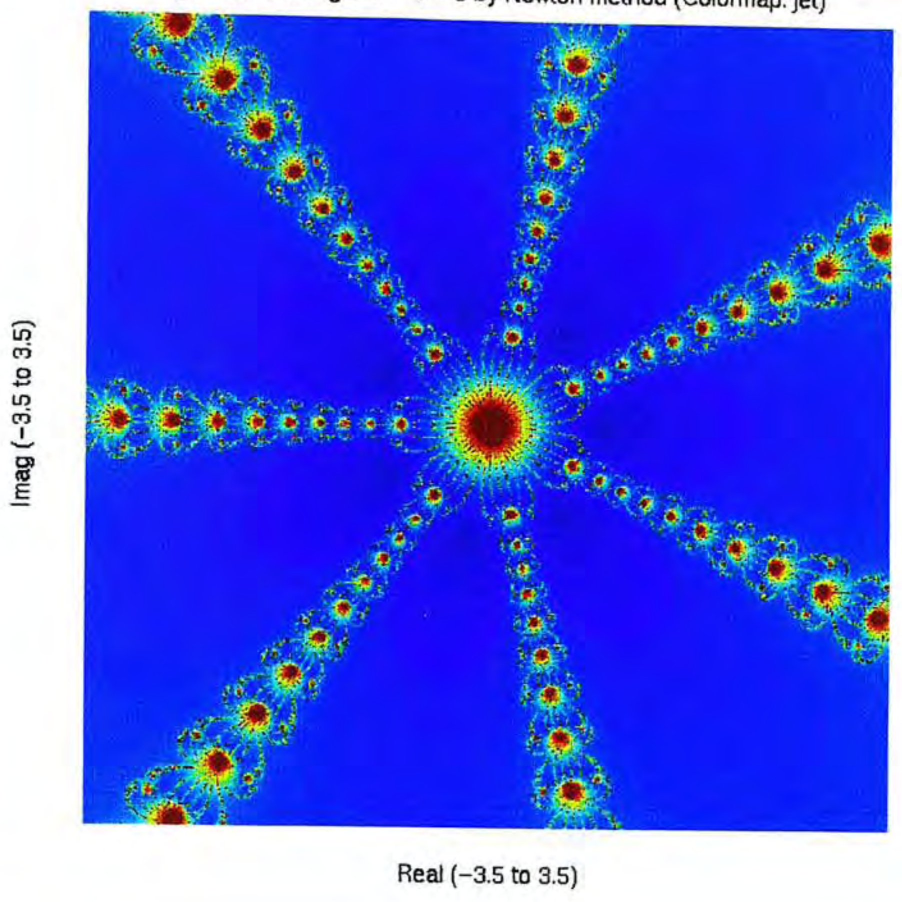


Fig A.2.b Solving z^7 − 1 = 0 by Halley method (Colormap: hsv)

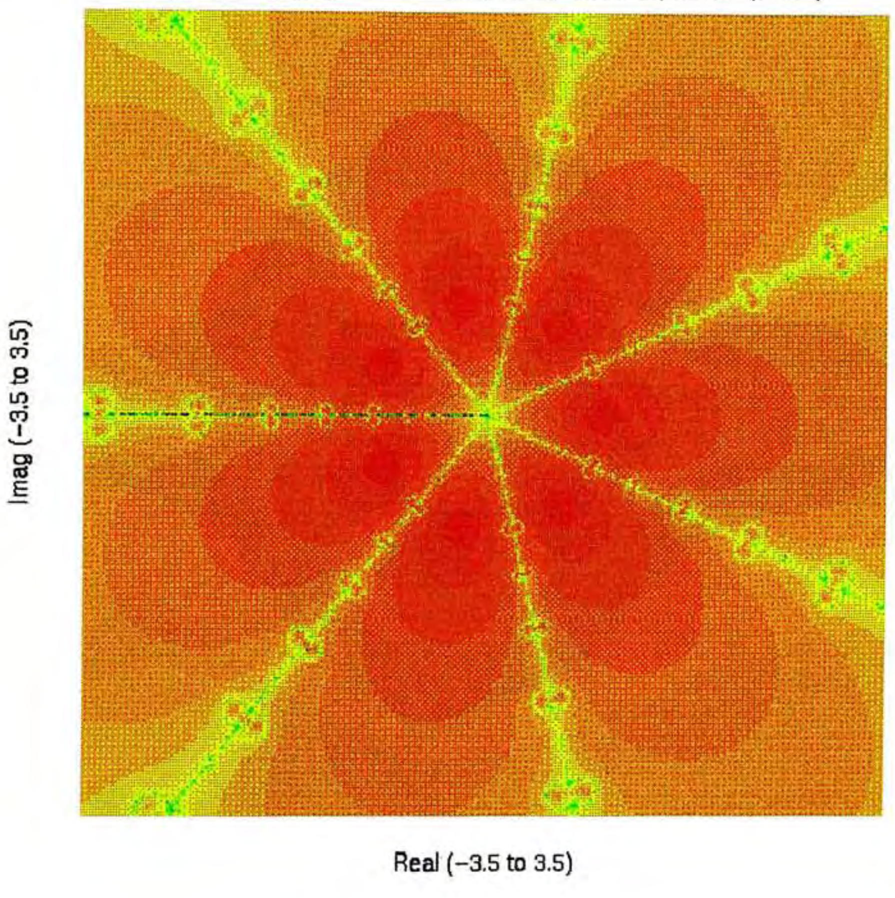


Fig A.3.a Magnifying part of Fig A.1.a

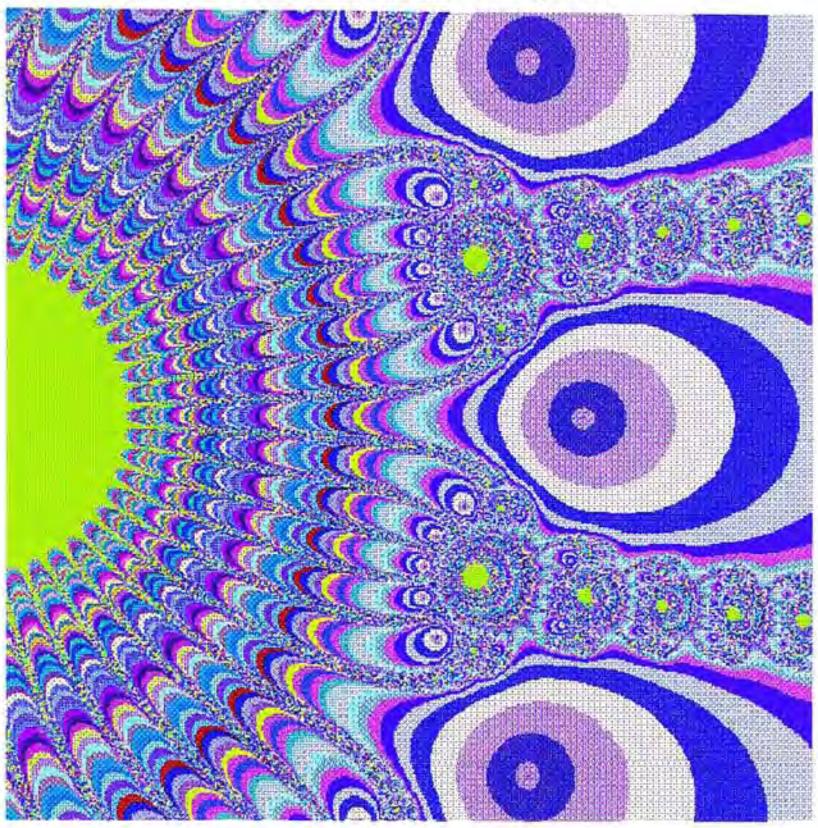


Fig A.3.b Magnifying part of Fig A.1.b

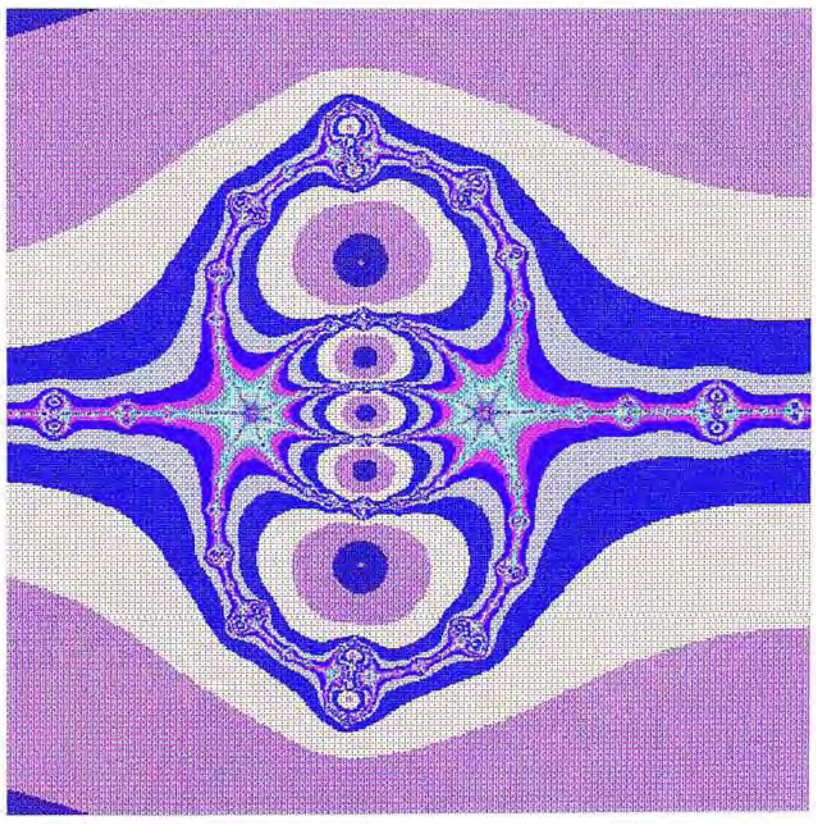


Fig A.4.a Magnifying part of Fig A.1.a (Colormap: jet)

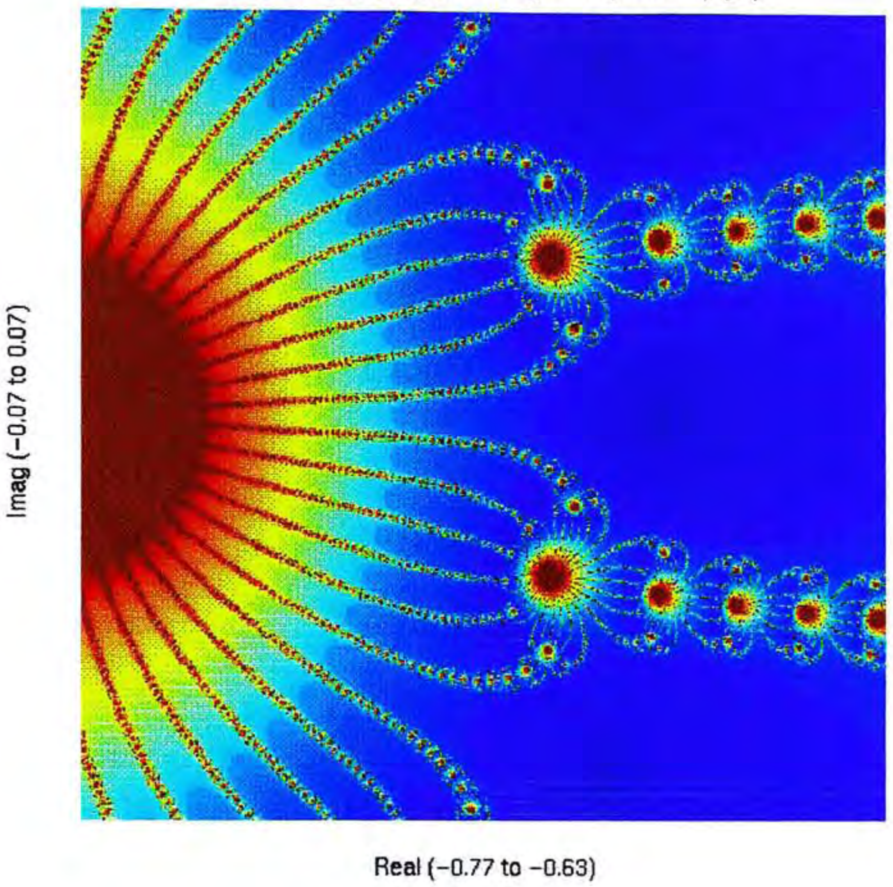


Fig A.4.b Magnifying part of Fig A.1.b (Colormap: hsv)

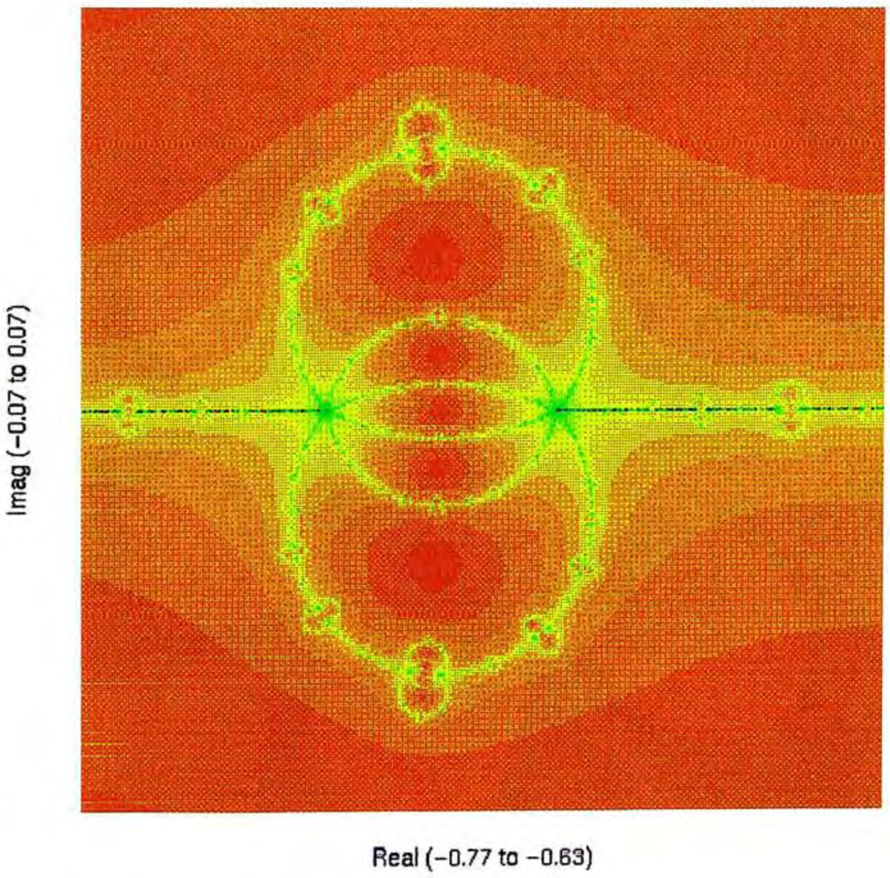


Fig A.5.a Solving z^7 + z^2 - 1 = 0 by Newton method

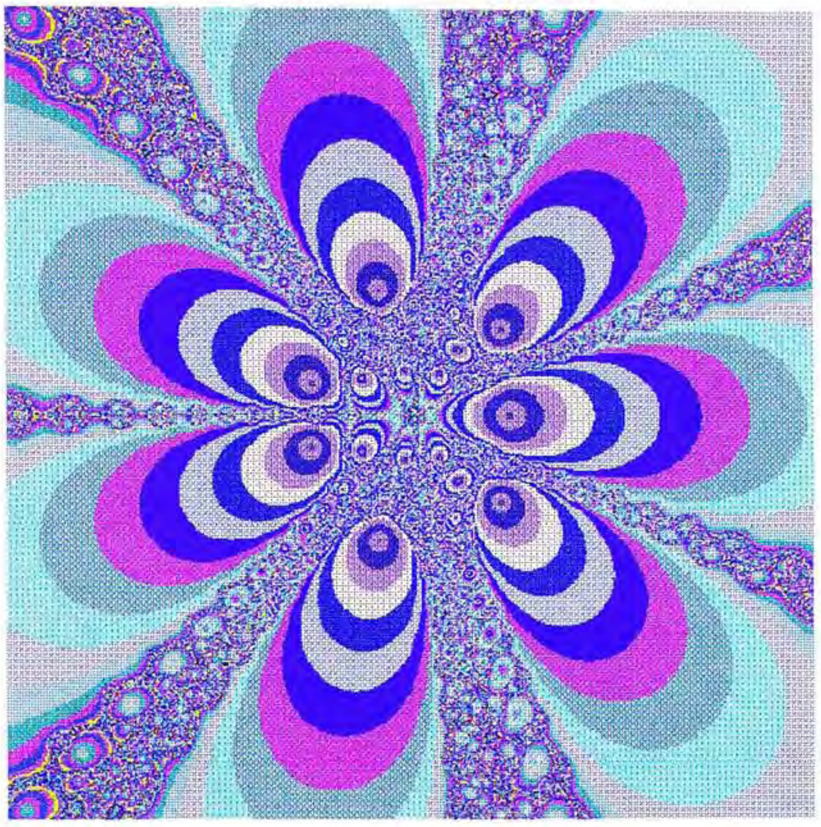


Fig A.5.b Solving z^7 + z^2 - 1 = 0 by Halley method

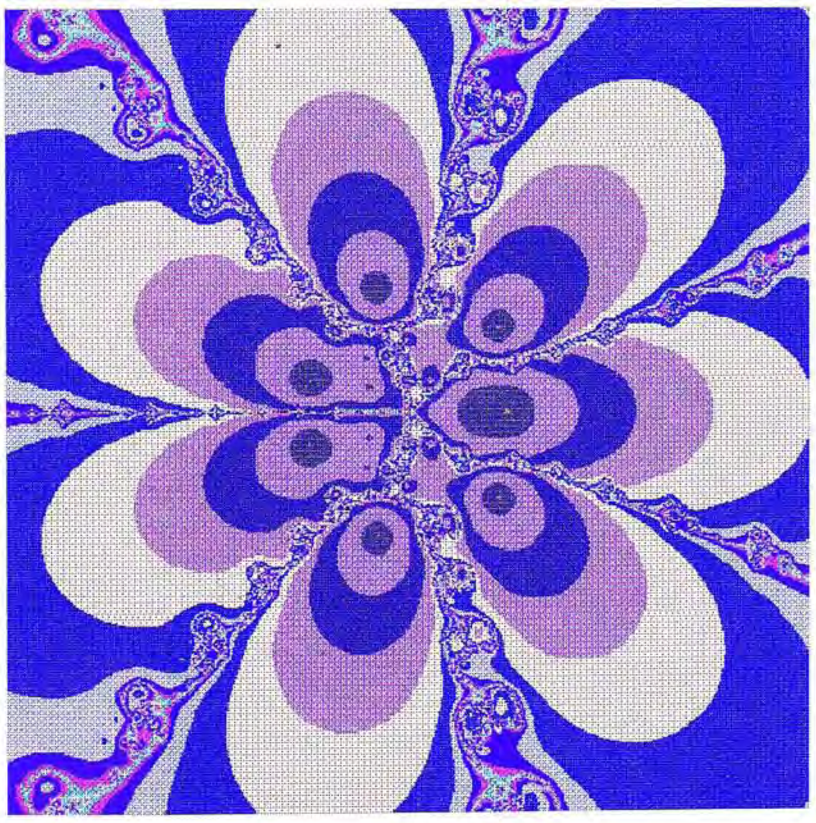


Fig A.5.c Solving z^7 + z^2 - 1 = 0 by Laguerre method

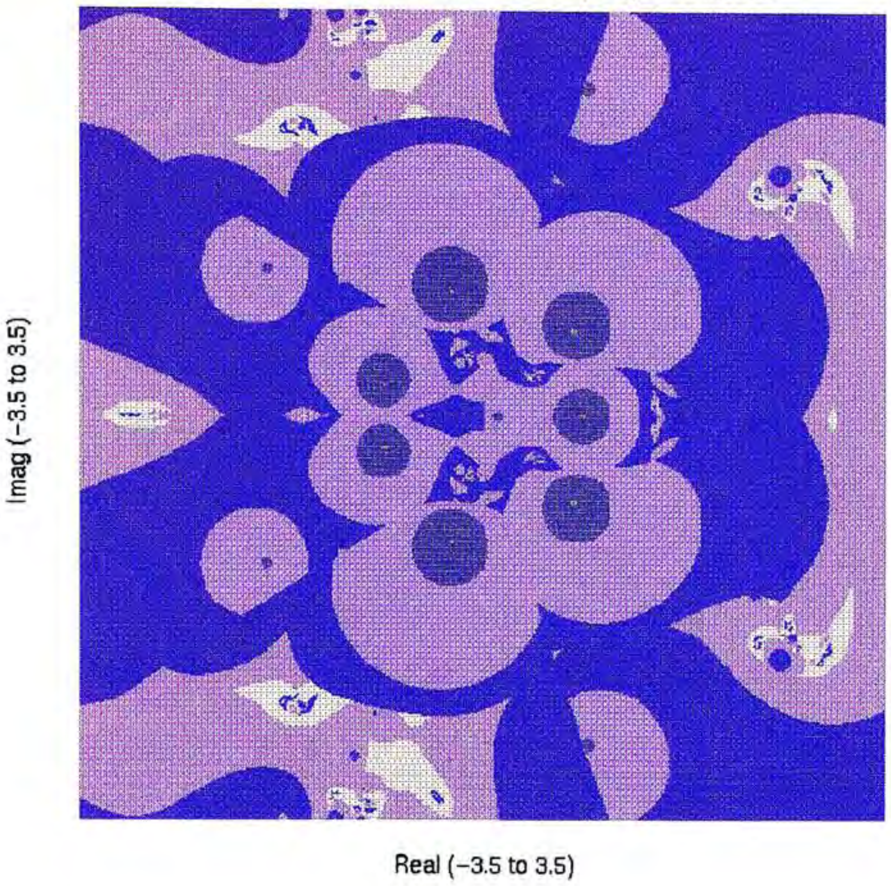
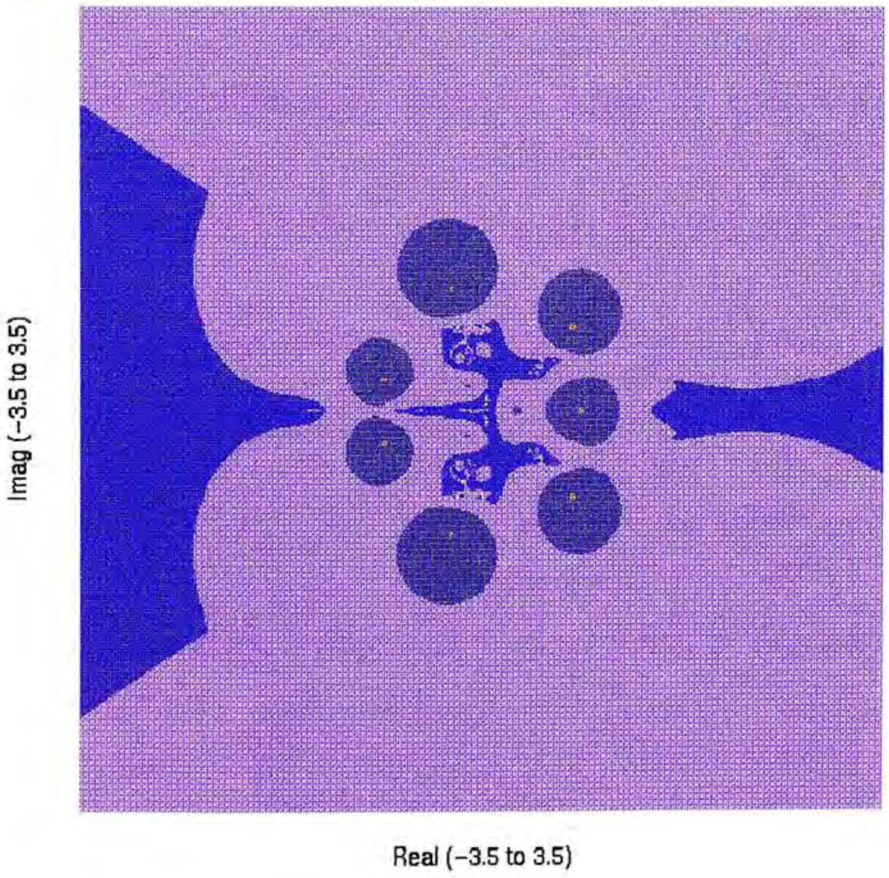


Fig A.5.d Solving z^7 + z^2 - 1 = 0 by CMR method


Imag (-3.5 to 3.5)
Real (-3.5 to 3.5)