THESIS
2
(1995)

MICHIGAN STATE UNIVERSITY LIBRARIES
3 1293 01421 7685

This is to certify that the

dissertation entitled

Quasi-Laguerre's Method and Its Parallel Implementation on Solving Symmetric Tridiagonal Eigenvalue Problems

presented by

Xiulin Zou

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Applied Mathematics

Major professor

Date 12/12/95

MSU is an Affirmative Action/Equal Opportunity Institution

O-12771

LIBRARY
Michigan State
University

PLACE IN RETURN BOX to remove this checkout from your record.
TO AVOID FINES return on or before date due.

DATE DUE	DATE DUE	DATE DUE

MSU is An Affirmative Action/Equal Opportunity Institution

c:\circ\datedue.pm3-p.1

QUASI-LAGUERRE'S METHOD AND ITS PARALLEL IMPLEMENTATION ON SOLVING SYMMETRIC TRIDIAGONAL EIGENVALUE PROBLEMS

By

Xiulin Zou

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Mathematics

1995

ABSTRACT

QUASI-LAGUERRE'S METHOD AND ITS PARALLEL IMPLEMENTATION ON SOLVING SYMMETRIC TRIDIAGONAL EIGENVALUE PROBLEMS

By

Xiulin Zou

First of all, a quasi-Laguerre's iterative method is derived for solving polynomial equations. Unlike the well-known Laquerre's method that requires evaluating the second derivative, the quasi-Laguerre's method only involves the evaluation of the function and its first derivative and still maintains the monotonical convergence property of the Laguerre's method for solving polynomial equations with only real roots. Two different approaches to derive the quasi-Laguerre's method are given and each approach reveals some different features of the method. It is also proven that the order of convergence of the quasi-Laguerre's method is $1 + \sqrt{2}$.

Secondly, a new algorithm using split-merge and the quasi-Laguerre's method with cloud and cluster handler is developed to solve symmetric tridiagonal eigenproblems. The cloud and cluster handler utilizes the multiplicity estimation method developed ealier in this work. Numerical results show that the new algorithm is very competitive in both speed and accuracy.

Finally a parallel version of the new algorithm is designed and implemented. Nu-

merical results on a substantial variety of matrices show our algorithm is the best one among the existing parallel algorithms for symmetric tridiagonal eigenvalue problems.

TO MY PARENTS

For their 70^{th} birthday

Contents

List of Tables

List of Figures

Acknowledgment First of all, I am indebted to my advisor, Prof. Tien-Yien Li, for his support and guidance during my study at Michigan State University. Secondly, I owe my wife, Ying Zhou, for her support and sacrifice as I was so intensively engaged in my research and teaching that she had to take full charge of all the domestic matters while still taking four courses of her own. I would like to thank all the Professors from whom I have gained more insight, more inspiration, or more knowledge about Mathematics and its application. I am very appreciative to Prof. Enbody from the Computer Science Department for giving me the privilege to take several of his courses to gain my knowledge with computer science and parallel computing and for his sponsoring a computer account in the Computer Science Department for me. I also want to thank the Advanced Computer System Lab at Michigan State University for providing the computing facilities for my research. I thank Mr. Min Jin for some valuable discusion, Ms. Wenjiang Qiao, Mr. Paul Gray, etc., for some PVM technical consulting and information exchanging. Last but not least, I thank all the committee members, Prof. Chiu, Prof. Dunninger, Prof. Frazier and Prof. Zhou, for reading my thesis and providing valuable suggestions.

Chapter 1

Introduction

Define

$$L_{\pm}(x) = x - \frac{n}{\frac{p'(x)}{p(x)} \pm \sqrt{(n-1)\left[(n-1)\left(\frac{p'(x)}{p(x)}\right)^2 - n\frac{p''(x)}{p(x)}\right]}}, \tag{1.1}$$

and

$$L(x) = \begin{cases} L_+(x) \text{ if } |L_+(x) - x| \leq |L_-(x) - x|, \\ L_-(x) \text{ otherwise.} \end{cases} \tag{1.2}$$

Then Laguerre's iteration $x_{k+1} = L(x_k)$, applied to a polynomial that has only real roots, converges monotonically and cubically to a root of the polynomial, starting from any initial point [16]. This method has been successfully used to find all the eigenvalues of symmetric tridiagonal matrices [19] with a great speed up. However, this method requires the evaluation of the second derivative, which is normally very expensive. A method that maintains the advantages of Laguerre's iteration and involves only first derivative is sought in this work.

The problem is formulated as follows. Let

$$f(x) = k\prod_{i=1}^{n}(x - r_i) \tag{1.3}$$

be a polynomial of degree n that has only real roots. Assume that the logarithmic derivatives $\frac{f'(x_0)}{f(x_0)} = q_0$ and $\frac{f'(x_1)}{f(x_1)} = q_1$ are known and no root of $f(x)$ lies between x_0 and x_1. Find, based on x_0, q_0, x_1, and q_1, an iterative method that converges monotonically to a root of $f(x)$, starting from x_0 and x_1.

First of all, our iteration formula for this purpose is derived for polynomials of form $f(x) = k(x-r)^m(x-z)^{n-m}$ by solving r in the following system,

$$\frac{m}{x_0 - r} + \frac{n-m}{x_0 - z} = q_0, \tag{1.4}$$

$$\frac{m}{x_1 - r} + \frac{n-m}{x_1 - z} = q_1. \tag{1.5}$$

The formula for r is

$$r = \frac{x_0 + x_1}{2} + \frac{mn - [(n+m)\Delta q + q_0 q_1 \Delta x]\frac{\Delta x}{4}}{-m\frac{(q_0+q_1)}{2} \pm \sqrt{-m(n-m)(q_0 q_1 + n\frac{\Delta q}{\Delta x}) + [q_0 q_1 + n\frac{\Delta q}{\Delta x}]^2 \frac{(\Delta x)^2}{4}}}, \tag{1.6}$$

where $\Delta q = q_1 - q_0$, and $\Delta x = x_1 - x_0$.

We call (1.6) the quasi-Laguerre's iteration formula.

We proceed to derive the formula for a general polynomial of form (1.3) and begin with the following two equations,

$$\sum_{j=1}^{n} \frac{1}{x_0 - r_j} = q_0, \tag{1.7}$$

$$\sum_{j=1}^{n} \frac{1}{x_1 - r_j} = q_1. \tag{1.8}$$

Let

$$\delta_{\pm} = \frac{mn - [(n+m)\Delta q + q_0 q_1 \Delta x]\frac{\Delta x}{4}}{-m\frac{(q_0+q_1)}{2} \pm \sqrt{-m(n-m)(q_0 q_1 + n\frac{\Delta q}{\Delta x}) + [q_0 q_1 + n\frac{\Delta q}{\Delta x}]^2 \frac{(\Delta x)^2}{4}}}, \tag{1.9}$$

and

$$\delta(x_0, x_1, q_0, q_1, m) = \begin{cases} \delta_+ \text{ if } |\delta_+| \le |\delta_-|, \\ \delta_- \text{ if } |\delta_-| < |\delta_+|. \end{cases} \tag{1.10}$$

Define

$$QL(x_0, x_1, q_0, q_1, m) = \frac{x_0 + x_1}{2} + \delta(x_0, x_1, q_0, q_1, m). \tag{1.11}$$

Then, the iteration $x_{k+1} = QL(x_{k-1}, x_k, q_{k-1}, q_k, 1)$, starting from x_0 and x_1, converges monotonically. It is shown that the limit of the iteration is a root of the polynomial. We call this iterative method the quasi-Laguerre's method. The integer m in the formula is called the multiplicity index of the quasi-Laguerre's formula. It is shown that if the multiplicity index of the formula matches the multiplicity of the root, the order of convergence of the iteration is $\sqrt{2} + 1$.

While quasi-Laguerre's iterative method is best suitable for solving polynomial equation with only real roots, it can also be used to solve polynomial that may have complex roots. Though the global convergence property is not guaranteed, we have shown that the order of convergence is still $\sqrt{2}+1$ provided that the multiplicity index matches the multiplicity of the root.

In solving polynomials that have only real roots, the two Laguerre's iteration sequences $x_{k+1}^{+} = x_k^{+} + L_{+}(x_k)$ and $x_{k+1}^{-} = x_k^{-} + L_{-}(x_k)$ both converge to a root of the polynomial, starting from any initial point x_0. That is, the Laguerre's iteration sequence generated by the same sign in formula (1.1) converges to a root of the polynomial, no matter where the iteration starts. However, this feature is not inherited by the quasi-Laguerre's function. A quasi-Laguerre's iteration sequence generated by $x_{k+1}^{+} = \frac{x_{k-1}^{+}+x_k^{+}}{2} + \delta_{+}$, with $m = 1$, marches toward the right hand side of the initial interval $[x_0, x_1]$. If there is no root of the polynomial in this direction, the iteration will wrap around and produce a point on the left hand side of the left most root of the polynomial. When this happens, the iteration can not proceed any more because there are roots of the polynomial between the two iteration points now. Similar situation could occur for quasi-Laguerre's iteration sequence generated using the $-$ sign.

A new algorithm with cloud and cluster handler is designed in Chapter 4, using Split-Merge and quasi-Laguerre's method to solve the symmetric tridiagonal eigenproblems. The new algorithm differs from the one given in [15].

Various numerical results for a substantial variety of matrices presented in Section 4.3 indicate that our algorithm is strongly competitive in both accuracy and speed. A parallel version of our algorithm is implemented on a cluster of workstations using PVM (Parallel Virtual Machine) and numerical results show that our algorithm leads all the existing algorithms on distributed memory parallel platforms.

Chapter 2

Quasi-Laguerre's method

2.1 Laguerre method

Let $p(x)$ be a polynomial of degree n, with all its roots being real. By

$$L_{\pm}(x) = x - \frac{n}{\frac{p'(x)}{p(x)} \pm \sqrt{(n-1)\left[(n-1)\left(\frac{p'(x)}{p(x)}\right)^2 - n\frac{p''(x)}{p(x)}\right]}}, \tag{2.1}$$

the Laguerre's iteration is defined as

$$x_{k+1}^{\pm} = L_{\pm}(x_k).$$

Let x_{k+1} be one of $\{x_{k+1}^-, x_{k+1}^+\}$ which is closer to x_k. Then the iterates $\{x_k\}$ converges at least linearly from any guess x_0 close to a zero z of $p(x)$, and converges cubically when z is a single root.

In general, Laguerre's iteration is defined via

$$L_{m\pm}(x) = x - \frac{n}{\frac{p'(x)}{p(x)} \pm \sqrt{\frac{n-m}{m}\left[(n-1)(\frac{p'(x)}{p(x)})^2 - n\frac{p''(x)}{p(x)}\right]}}. \tag{2.2}$$

This function is called the Laguerre iteration function with multiplicity index m. Let x_{k+1} be one of the $\{L_{m-}(x_k), L_{m+}(x_k)\}$ that is closer to x_k. Then the iterates $\{x_k\}$ converges at least linearly from any guess x_0 close to a zero of $p(x)$ with multiplicity greater than or equal to m, and cubically if x_0 is close to a root that has multiplicity exactly equal to m.

Kahan [16] derived this general Laguerre function in the Riemann sphere. In the real field, Kahan [16] used the non-overshooting strategy to derive the Laguerre's formula, which leads to the monotonical convergence of the method for $m = 1$. For general $m < n$, the method never overshoot more than $m - 1$ roots of the polynomial.

2.2 Quasi-Laguerre method

Laguerre's iteration is an excellent method for finding the roots of polynomials with only real roots because of its global and cubic convergence. It has been successfully applied to solving symmetric tridiagonal eigenvalue problems by Li and Zeng [19]. However Laguerre's iteration involves evaluations of the polynomial itself, its first derivative, and its second derivative. In [7], the so-called Quasi-Laguerre's iteration was established which maintains the global and monotonic convergence of the Laguerre's iteration without evaluating the second derivative. We shall present in the following a different approach to derive the iteration which were obtained independently from, and almost simultaneously as, the work in [7].

The quasi-Laguerre's formula derived from a special polynomial

First of all, we assume the polynomial is of form $p(x) = a(x - z)^m(x - t)^{n-m}$. Suppose $\{x_0, p(x_0), p'(x_0)\}$ and $\{x_1, p(x_1), p'(x_1)\}$ are known. Let $q_0 = \frac{p'(x_0)}{p(x_0)}$ and $q_1 = \frac{p'(x_1)}{p(x_1)}$. Then,

$$\frac{m}{x_0 - z} + \frac{n - m}{x_0 - t} = q_0. \tag{2.3}$$

$$\frac{m}{x_1 - z} + \frac{n - m}{x_1 - t} = q_1. \tag{2.4}$$

Eliminating t from the above two equations yields,

$$\frac{x_1 - x_0}{n - m}[q_0 q_1 - (\frac{mq_0}{x_1 - z} + \frac{mq_1}{x_0 - z}) + \frac{m^2}{(x_1 - z)(x_0 - z)}] = (q_0 - q_1) + m\frac{x_0 - x_1}{(x_1 - z)(x_0 - z)}.$$

Let $\Delta x = x_1 - x_0$, $\Delta q = q_1 - q_0$. It follows that

$$q_0 q_1 - m\frac{q_0 x_0 + q_1 x_1 - (q_0 + q_1)z}{(x_1 - z)(x_0 - z)} + \frac{m^2}{(x_1 - z)(x_0 - z)} = -(n-m)\frac{\Delta q}{\Delta x} - \frac{m(n - m)}{(x_1 - z)(x_0 - z)}.$$

$$q_0q_1 + (n-m)\frac{\Delta q}{\Delta x} = \frac{m(q_0x_0 + q_1x_1) - m(q_0+q_1)z - m^2 - m(n-m)}{(x_1-z)(x_0-z)}.$$

$$q_0q_1 + (n-m)\frac{\Delta q}{\Delta x} = \frac{m(q_0x_0 + q_1x_1 - n) - m(q_0+q_1)z}{(x_1-z)(x_0-z)}.$$

Let

$$\begin{aligned} A &= q_0q_1 + (n-m)\tfrac{\Delta q}{\Delta x}, \\ B &= m(q_0x_0 + q_1x_1 - n), \\ C &= m(q_0+q_1). \end{aligned} \tag{2.5}$$

Then

$$A(x_1-z)(x_0-z) = B - Cz. \tag{2.6}$$

Or,

$$Az^2 + [C - A(x_0+x_1)]z + Ax_0x_1 - B = 0.$$

So,

$$\begin{aligned} z &= \frac{A(x_0+x_1) - C \pm \sqrt{[C-A(x_0+x_1)]^2 - 4A(Ax_0x_1 - B)}}{2A} \\ &= \frac{x_0+x_1}{2} - \frac{1}{2A}\{C \pm \sqrt{[C-A(x_0+x_1)]^2 - 4A(Ax_0x_1 - B)}\} \\ &= \frac{x_0+x_1}{2} - \frac{1}{2A}\{C \pm \sqrt{Q}\}, \end{aligned}$$

where

$$\begin{aligned} Q &= C^2 - 2CA(x_0+x_1) + A^2(x_0+x_1)^2 - 4A^2x_0x_1 + 4AB \\ &= C^2 - 2CA(x_0+x_1) + A^2(x_0-x_1)^2 + 4AB = C^2 + Q_1, \end{aligned}$$

and

$$\begin{aligned} Q_1 &= -2CA(x_0+x_1) + A^2(x_0-x_1)^2 + 4AB \\ &= A[-2C(x_0+x_1) + A(x_0-x_1)^2 + 4B] \\ &= A\{-2m(q_0+q_1)(x_0+x_1) + [q_0q_1(\Delta x)^2 + (n-m)\Delta q\Delta x] \\ &\quad +4m(q_0x_0 + q_1x_1 - n)\} \\ &= A\{-2m[q_0x_0 + q_1x_0 + q_0x_0 + q_1x_1] + q_0q_1(\Delta x)^2 \\ &\quad +(n-m)[q_1x_1 - q_1x_0 - q_0x_1 + q_0x_0] + 4m[q_0x_0 + q_1x_1 - n]\} \end{aligned}$$

$$
\begin{aligned}
&= A\{[-2m+(n-m)+4m][q_0x_0+q_1x_1] \\
&\quad +[-2m-(n-m)][q_0x_1+q_1x_0]-4mn+q_0q_1(\Delta x)^2\} \\
&= A\{(n+m)[q_0x_0q_1x_0-q_0x_1-q_1x_0]-4mn+q_0q_1(\Delta x)^2\} \\
&= A\{(n+m)(q_1-q_0)(x_1-x_0)-4mn+q_0q_1(\Delta x)^2\} \\
&= A\{(n+m)\Delta q\Delta x-4mn+q_0q_1(\Delta x)^2\}.
\end{aligned}
$$

Thus,

$$
\begin{aligned}
Q &= m^2(q_0+q_1)^2+[q_0q_1+(n-m)\frac{\Delta q}{\Delta x}][-4mn+(n+m)\Delta q\Delta x+q_0q_1(\Delta x)^2] \\
&= m^2(q_0+q_1)^2-4mnq_0q_1-4mn(n-m)\frac{\Delta q}{\Delta x}+(n+m)q_0q_1\Delta q\Delta x \\
&\quad +(n^2-m^2)(\Delta q)^2+q_0^2q_1^2(\Delta x)^2+(n-m)q_0q_1\Delta q\Delta x \\
&= m^2(q_0+q_1)^2-4mnq_0q_1-4mn(n-m)\frac{\Delta q}{\Delta x} \\
&\quad +2nq_0q_1\Delta q\Delta x+(n^2-m^2)(\Delta q)^2+q_0^2q_1^2(\Delta x)^2 \\
&= m^2(q_0+q_1)^2-4mnq_0q_1-4mn(n-m)\frac{\Delta q}{\Delta x}+[q_0q_1\Delta x+n\Delta q]^2-m^2(\Delta q)^2 \\
&= m^2[(q_0+q_1)^2-(q_0-q_1)^2]-4mnq_0q_1-4mn(n-m)\frac{\Delta q}{\Delta x}+[q_0q_1\Delta x+n\Delta q]^2 \\
&= -4m(n-m)(q_0q_1+n\frac{\Delta q}{\Delta x})+[q_0q_1\Delta x+n\Delta q]^2
\end{aligned}
$$

Combine the above equations, we have

$$
\begin{aligned}
z &= \frac{x_0+x_1}{2}-\frac{1}{2A}\frac{(C^2-Q)}{(C\mp\sqrt{Q})} \\
&= \frac{x_0+x_1}{2}-\frac{1}{2A}\frac{Q_1}{(C\mp\sqrt{Q})} \\
&= \frac{x_0+x_1}{2}+\frac{1}{2}\frac{4mn-[(n+m)\Delta q+q_0q_1\Delta x]\Delta x}{-m(q_0+q_1)\pm\sqrt{-4m(n-m)(q_0q_1+n\frac{\Delta q}{\Delta x})+[q_0q_1\Delta x+n\Delta q]^2}} \\
&= \frac{x_0+x_1}{2}+\frac{mn-[(n+m)\Delta q+q_0q_1\Delta x]\frac{\Delta x}{4}}{-m\frac{(q_0+q_1)}{2}\pm\sqrt{-m(n-m)(q_0q_1+n\frac{\Delta q}{\Delta x})+[q_0q_1+n\frac{\Delta q}{\Delta x}]^2\frac{(\Delta x)^2}{4}}}. \qquad (2.7)
\end{aligned}
$$

We call (2.7) the Quasi-Laguerre formula with multiplicity index m. Letting $x_1 \to x_0$, the general Laguerre's iteration formula (2.2) with multiplicity index m is obtained.

An alternative form of the quasi-Laguerre's formula

Rewrite (2.6) in the following form

$$A(z-x_1)^2+[C+A(x_1-x_0)](z-x_1)+Cx_1-B=0,$$

where A, B, and C are defined by (2.5), then the solution can be expressed as,

$$z=x_1+\frac{-(C+A\Delta x)\pm\sqrt{R}}{2A}, \tag{2.8}$$

where

$$\begin{aligned}
R &= (C+A\Delta x)^2-4A(Cx_1-B)\\
&= C^2+2AC\Delta x+A^2(\Delta x)^2-4ACx_1+4AB\\
&= C^2-2AC(x_0+x_1)+A^2(\Delta x)^2+4AB\\
&= m^2(q_0+q_1)^2-2\left[q_0q_1+(n-m)\frac{\Delta q}{\Delta x}\right]m(q_0+q_1)(x_0+x_1)\\
&\quad +(q_0q_1+(n-m)\frac{\Delta q}{\Delta x})^2(\Delta x)^2+4\left[q_0q_1+(n-m)\frac{\Delta q}{\Delta x}\right]m(q_0x_0+q_1q_1-n)\\
&= m^2(q_0+q_1)^2+(q_0q_1+(n-m)\frac{\Delta q}{\Delta x})^2(\Delta x)^2\\
&\quad +2\left[q_0q_1+(n-m)\frac{\Delta q}{\Delta x}\right]m(\Delta x\Delta q-2n)\\
&= m^2(q_0+q_1)^2+\left[q_0q_1+n\frac{\Delta q}{\Delta x}\right]^2(\Delta x)^2-2\left[q_0q_1+n\frac{\Delta q}{\Delta x}\right]m\frac{\Delta q}{\Delta x}(\Delta x)^2\\
&\quad +m^2(\Delta q)^2+2m\left[q_0q_1+n\frac{\Delta q}{\Delta x}\right](\Delta x\Delta q-2n)-2m^2(\Delta q)^2+4nm^2\frac{\Delta q}{\Delta x}\\
&= 4m^2\left[q_0q_1+n\frac{\Delta q}{\Delta x}\right]+\left[q_0q_1+n\frac{\Delta q}{\Delta x}\right]^2(\Delta x)^2-4mn\left[q_0q_1+n\frac{\Delta q}{\Delta x}\right]\\
&= \left[q_0q_1+n\frac{\Delta q}{\Delta x}\right]\left(\left[q_0q_1+n\frac{\Delta q}{\Delta x}\right](\Delta x)^2+4m^2-4mn\right),
\end{aligned}$$

and

$$\begin{aligned}
C+A\Delta x &= m(q_0+q_1)+\left(q_0q_1+(n-m)\frac{\Delta q}{\Delta x}\right)\Delta x\\
&= 2mq_0+(q_0q_1+n\frac{\Delta q}{\Delta x}).
\end{aligned}$$

Substituting these into (2.8) yields,

$$z=x_1+\frac{2mq_0+(q_0q_1+n\frac{\Delta q}{\Delta x})\pm\sqrt{\left[q_0q_1+n\frac{\Delta q}{\Delta x}\right]\left(\left[q_0q_1+n\frac{\Delta q}{\Delta x}\right](\Delta x)^2+4m(m-n)\right)}}{2(q_0q_1+(n-m)\frac{\Delta q}{\Delta x})} \tag{2.9}$$

Note that $Cx_1 - B = m(q_0 + q_1)x_1 - m(q_0x_0 + q_1x_1 - n) = nm + mq_0\Delta x$. So,

$$z = x_1 + \frac{2(nm + mq_0\Delta x)}{-\left(2mq_0 + (q_0q_1 + n\frac{\Delta q}{\Delta x})\right) \pm \sqrt{\left[q_0q_1 + n\frac{\Delta q}{\Delta x}\right]\left(\left[q_0q_1 + n\frac{\Delta q}{\Delta x}\right](\Delta x)^2 + 4m(m-n)\right)}} \tag{2.10}$$

Equations (2.7), (2.9) and (2.10) are different formulae for the same root z of the polynomial $p(x) = k(x-z)^m(x-t)^{n-m}$. In the following, we shall use those different forms in different occasions.

Deriving the quasi-Laguerre's formula for general polynomials

Let $p(x)$ be a polynomial of degree n with only real roots $r_1, r_2, \cdots, r_n$. Some of these roots may be equal to each other, i.e., they may be multiple roots. Let x_0, x_1, q_0, q_1 be given such that $x_0 < x_1$, $\frac{p'(x_0)}{p(x_0)} = q_0$, $\frac{p'(x_1)}{p(x_1)} = q_1$ and the interval $[x_0, x_1]$ does not contain any $r_i's$. To summarize, we emphasize that the derivations hereafter will always be under the

Basic Assumptions:

1. The polynomial $p(x)$ has only real roots.

2. There is no root of $p(x)$ between x_0 and x_1.

3. The polynomial $p(x)$ has prescribed logarithmic derivatives at x_0 and x_1.

Write $p(x) = k\prod_{j=1}^{n}(x - r_j)$. We want to estimate all the roots of $p(x)$ under the above conditions. It turns out that the estimation gives rise to the quasi-Laguerre's formula (2.7).

To begin, notice that

$$q_0 = \sum_{j=1}^{n} \frac{1}{x_0 - r_j}, \tag{2.11}$$

$$q_1 = \sum_{j=1}^{n} \frac{1}{x_1 - r_j}. \tag{2.12}$$

Let $Y_j = \frac{x_0+x_1}{2} - r_j, j = 1, 2, \cdots, n$, and $y = \frac{x_1-x_0}{2}$. Then,

$$q_0 = \sum_{j=1}^{n} \frac{1}{Y_j - y}, \tag{2.13}$$

$$q_1 = \sum_{j=1}^{n} \frac{1}{Y_j + y}. \tag{2.14}$$

Adding and then subtracting the above two equations, we have

$$q_{01} = \sum_{j=1}^{n} \frac{Y_j}{Y_j^2 - y^2}, \tag{2.15}$$

$$\frac{-\Delta q}{\Delta x} = \sum_{j=1}^{n} \frac{1}{Y_j^2 - y^2}, \tag{2.16}$$

where $q_{01} = \frac{q_0+q_1}{2}$ and $\Delta q = q_1 - q_0, \Delta x = x_1 - x_0$.

Lemma 2.2.1 *The following inequalities hold*

$$Y_j^2 - y^2 > 0, \text{ for all } j = 1, 2, \cdots, n, \tag{2.17}$$

and

$$\frac{-\Delta q}{\Delta x} > 0. \tag{2.18}$$

Proof. Since $Y_j^2 - y^2 = (Y_j + y)(Y_j - y) = (x_1 - r_j)(x_0 - r_j)$, inequality (2.17) follows. Inequality (2.2.1) follows from inequality (2.17) and equation (2.16) □

Let $Y = \frac{x_0+x_1}{2} - r$, where r is a root of the polynomial $p(x)$ with multiplicity greater than or equal to m. For simplicity and without loss of generality, let $r = r_1 = r_2 = \cdots = r_m = \cdots = r_{m+k}$, where $k \geq 0$. So $Y = Y_1 = Y_2 = \cdots = Y_{m+k}$. Rewrite (2.15) and (2.16) into the following,

$$\sum_{j=m+1}^{n} \frac{Y_j}{Y_j^2 - y^2} = q_{01} - \frac{mY}{Y^2 - y^2}, \tag{2.19}$$

$$\sum_{j=m+1}^{n} \frac{1}{Y_j^2 - y^2} = \frac{-\Delta q}{\Delta x} - \frac{m}{Y^2 - y^2}. \tag{2.20}$$

Note that Y_{m+1}, $\cdots$, Y_{m+k} stay on the left hand side of the above two equations, only m equal terms are combined and moved to the right hand side. It follows from Holder's inequality,

$$\left(\sum_{j=m+1}^{n} \frac{Y_j}{Y_j^2 - y^2}\right)^2 \leq \sum_{j=m+1}^{n} \frac{Y_j^2}{Y_j^2 - y^2} \sum_{j=m+1}^{n} \frac{1}{Y_j^2 - y^2} \tag{2.21}$$

$$= \left[n - m + y^2 \sum_{j=m+1}^{n} \frac{1}{Y_j^2 - y^2}\right] \sum_{j=m+1}^{n} \frac{1}{Y_j^2 - y^2}. \tag{2.22}$$

Remark 2.2.2 *Equality holds in (2.21) if and only if* $Y_{m+1} = Y_{m+2} = \cdots = Y_n$. *So* $p(x)$ *has only two different roots.*

Using (2.19) and (2.20), we have

$$\left[q_{01} - \frac{mY}{Y^2 - y^2}\right]^2 \leq \left[n - m + y^2(-\frac{\Delta q}{\Delta x} - \frac{m}{Y^2 - y^2})\right]\left(-\frac{\Delta q}{\Delta x} - \frac{m}{Y^2 - y^2}\right). \quad (2.23)$$

Or,

$$q_{01}^2 + (n-m)\frac{\Delta q}{\Delta x} - \frac{(\Delta q)^2}{4} - 2mq_{01}\frac{Y}{Y^2 - y^2} + [m^2 + (n-m)m - m\frac{\Delta q \Delta x}{2}]\frac{1}{Y^2 - y^2} \leq 0.$$

Since $Y^2 - y^2 > 0$, so,

$$\left[q_{01}^2 + (n-m)\frac{\Delta q}{\Delta x} - \frac{(\Delta q)^2}{4}\right](Y^2 - y^2) - 2mq_{01}Y + nm - m\frac{\Delta q \Delta x}{2} \leq 0.$$

And hence,

$$\left[nm - m\frac{\Delta q \Delta x}{2} - y^2\left(q_{01}^2 + (n-m)\frac{\Delta q}{\Delta x} - \frac{(\Delta q)^2}{4}\right)\right]\frac{1}{Y^2}$$
$$-2mq_{01}\frac{1}{Y} + q_{01}^2 + (n-m)\frac{\Delta q}{\Delta x} - \frac{(\Delta q)^2}{4} \leq 0. \quad (2.24)$$

Or,

$$\left[nm - m\frac{\Delta q \Delta x}{4} - y^2\left(q_0 q_1 + n\frac{\Delta q}{\Delta x}\right)\right]\frac{1}{Y^2} - 2mq_{01}\frac{1}{Y} + q_0 q_1 + (n-m)\frac{\Delta q}{\Delta x} \leq 0. \quad (2.25)$$

Lemma 2.2.3 *The inequality*

$$q_0 q_1 + n\frac{\Delta q}{\Delta x} \leq 0 \quad (2.26)$$

holds, and equality holds if and only if $p(x) = k(x - r)^n$.

Proof. Applying Holder's inequality, we have

$$\begin{aligned}
& q_0 q_1 + n\frac{\Delta q}{\Delta x} = n\frac{\Delta q}{\Delta x} + q_{01}^2 - \frac{(\Delta q)^2}{4} \\
= & -n\sum_{j=1}^{n}\frac{1}{Y^2 - y^2} + \left(\sum_{j=1}^{n}\frac{Y_j}{Y_j^2 - y^2}\right)^2 - y^2\left(\sum_{j=1}^{n}\frac{1}{Y_j^2 - y^2}\right)^2 \\
\leq & -n\sum_{j=1}^{n}\frac{1}{Y^2 - y^2} + \left(\sum_{j=1}^{n}\frac{Y_j^2}{Y_j^2 - y^2}\right)\left(\sum_{j=1}^{n}\frac{1}{Y^2 - y^2}\right) - y^2\left(\sum_{j=1}^{n}\frac{1}{Y_j^2 - y^2}\right)^2 \\
\leq & -n\sum_{j=1}^{n}\frac{1}{Y^2 - y^2} + \sum_{j=1}^{n}\frac{1}{Y^2 - y^2}\left(\sum_{j=1}^{n}\frac{Y_j^2}{Y_j^2 - y^2} - y^2\sum_{j=1}^{n}\frac{1}{Y_j^2 - y^2}\right) = 0.
\end{aligned}$$

Equality holds if and only if $Y_1 = Y_2 = \cdots = Y_n$. □

Lemma 2.2.4 *The inequality*

$$nm - m\frac{\Delta q \Delta x}{4} - y^2\left(q_0 q_1 + n\frac{\Delta q}{\Delta x}\right) \geq nm \tag{2.27}$$

holds.

Proof. The assertion follows directly from Lemma 2.2.3 and Lemma 2.2.1 □

Lemma 2.2.4 indicates that the leading coefficient of the quadratic inequality (2.24) is positive. Let a, b, c denote the coefficients of inequality (2.24), i.e.,

$$\begin{aligned} a &= nm - m\frac{\Delta q \Delta x}{4} - y^2\left(q_0 q_1 + n\frac{\Delta q}{\Delta x}\right) \\ &= nm - (n+m)\frac{\Delta q \Delta x}{4} - q_0 q_1 \frac{(\Delta x)^2}{4}, \\ b &= -2mq_{01} = -2m\frac{q_0 + q_1}{2}, \\ c &= q_0 q_1 + (n-m)\frac{\Delta q}{\Delta x}. \end{aligned}$$

To solve the quadratic inequality, let's solve the quadratic equation first. Denote the two roots of the quadratic equation (2.25) by z_1 and z_2. Then

$$\begin{aligned} z_{1,2} &= \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{m\frac{q_0+q_1}{2} \pm \sqrt{(\frac{b}{2})^2 - ac}}{a} \\ &= \frac{m\frac{q_0+q_1}{2} \pm \sqrt{m^2(\frac{q_0+q_1}{2})^2 - [nm - (n+m)\frac{\Delta q \Delta x}{4} - q_0 q_1 \frac{\Delta x^2}{4}][(n-m)\frac{\Delta q}{\Delta x} + q_0 q_1]}}{nm - (n+m)\frac{\Delta q \Delta x}{4} - q_0 q_1 \frac{\Delta x^2}{4}}. \end{aligned}$$

Simplifying the radicant, we have

$$\begin{aligned} radicant &= m^2(\frac{q_0+q_1}{2})^2 - nm(n-m)\frac{\Delta q}{\Delta x} - nmq_0 q_1 + (n^2 - m^2)\frac{(\Delta q)^2}{4} \\ &\quad +(n+m)\frac{\Delta q \Delta x}{4} q_0 q_1 + (n-m)q_0 q_1 \frac{\Delta q \Delta x}{4} + (q_0 q_1)^2 \frac{(\Delta x)^2}{4} \\ &= n^2\frac{(\Delta q)^2}{4} + m^2 q_0 q_1 - nm(n-m)\frac{\Delta q}{\Delta x} - nmq_0 q_1 \\ &\quad +2nq_0 q_1 \frac{\Delta q \Delta x}{4} + (q_0 q_1)^2 \frac{(\Delta x)^2}{4} \\ &= (\frac{n\Delta q}{2})^2 - m(n-m)q_0 q_1 - nm(n-m)\frac{\Delta q}{\Delta x} + nq_0 q_1 \Delta x \frac{\Delta q}{2} + (q_0 q_1 \frac{\Delta x}{2})^2 \\ &= -m(n-m)(q_0 q_1 + n\frac{\Delta q}{\Delta x}) + (n\Delta q + q_0 q_1 \Delta x)^2 /4. \end{aligned}$$

So,

$$z_{1,2} = \frac{m\frac{q_0+q_1}{2} \pm \sqrt{-m(n-m)(q_0q_1 + n\frac{\Delta q}{\Delta x}) + (n\Delta q + q_0q_1\Delta x)^2/4}}{nm - (n+m)\frac{\Delta q \Delta x}{4} - q_0q_1\frac{\Delta x^2}{4}}. \tag{2.28}$$

Lemma 2.2.5 *The radicant under the square root in (2.28) is nonnegative.*

Proof. This is an easy consequence of Lemma 2.2.3. □

Lemma 2.2.5 ensures the roots z_1 and z_2 of equation $az^2 + bz + c = 0$ are all real. Without loss of generality, we assume $z_1 \leq z_2$.

Since the leading coefficient of the quadratic inequality (2.25) is greater than zero, we have

$$z_1 \leq \frac{1}{Y} \leq z_2.$$

To achieve an inequality for Y, consider the following three cases.

Case 1: $z_1 < 0 < z_2$.

For this case we have

$$Y \leq \frac{1}{z_1},$$

or

$$Y \geq \frac{1}{z_2}.$$

Since $Y = \frac{x_0+x_1}{2} - r$, so,

$$r \leq \frac{x_0 + x_1}{2} - \frac{1}{z_2},$$

or

$$r \geq \frac{x_0 + x_1}{2} - \frac{1}{z_1}.$$

So $p(x)$ may have roots on either side of the interval $[x_0, x_1]$, and $\frac{x_0+x_1}{2} - \frac{1}{z_1}$ (resp. $\frac{x_0+x_1}{2} - \frac{1}{z_2}$) is the nearest possible root to the right (resp. left) hand side of the interval. See Figure 2.1.

Case 2: $0 < z_1 \leq z_2$.

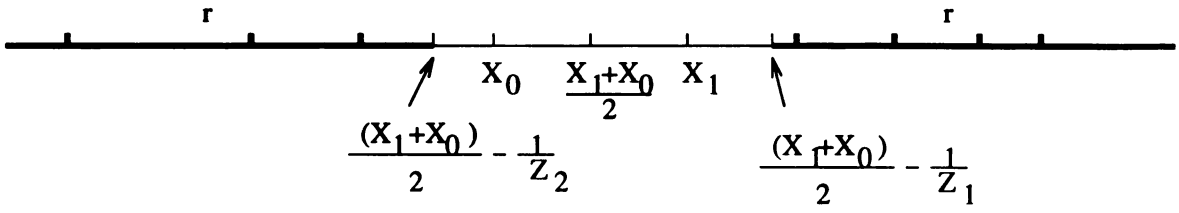

Figure 2.1: Distribution of the roots of the polynomial in case 1

For this case we have

$$\frac{1}{z_2} \leq Y \leq \frac{1}{z_1}.$$

So

$$\frac{x_0 + x_1}{2} - \frac{1}{z_1} \leq r \leq \frac{x_0 + x_1}{2} - \frac{1}{z_2}.$$

That is, all the roots are on the left hand side of the interval $[x_0, x_1]$. The root $\frac{x_0+x_1}{2} - \frac{1}{z_2}$ is the nearest possible to the left hand side of the interval and $\frac{x_0+x_1}{2} - \frac{1}{z_1}$ is the farthest possible root to the left of the interval. See Figure 2.2.

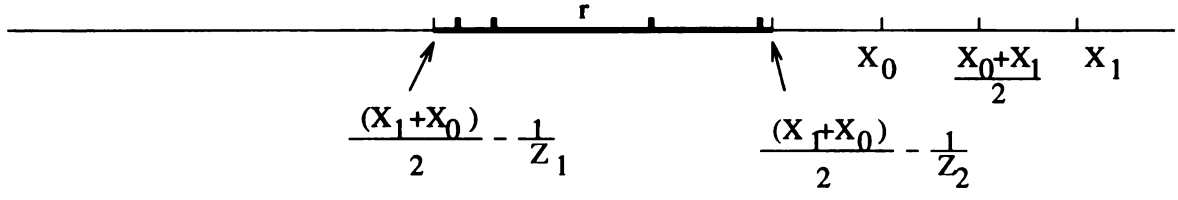

Figure 2.2: Distribution of the roots of the polynomial in case 2

Case 3: $z_1 \leq z_2 < 0$.

This case is similar to case 2 and we also have

$$\frac{1}{z_2} \leq Y \leq \frac{1}{z_1}.$$

It follows that

$$\frac{x_0 + x_1}{2} - \frac{1}{z_1} \leq r \leq \frac{x_0 + x_1}{2} - \frac{1}{z_2}.$$

That is, all the roots are on the right hand side of the interval $[x_0, x_1]$ in this case. The root $\frac{x_0+x_1}{2} - \frac{1}{z_1}$ is the nearest possible to the right hand side of the interval and $\frac{x_0+x_1}{2} - \frac{1}{z_2}$ is the farthest possible root to the right. See Figure 2.3.

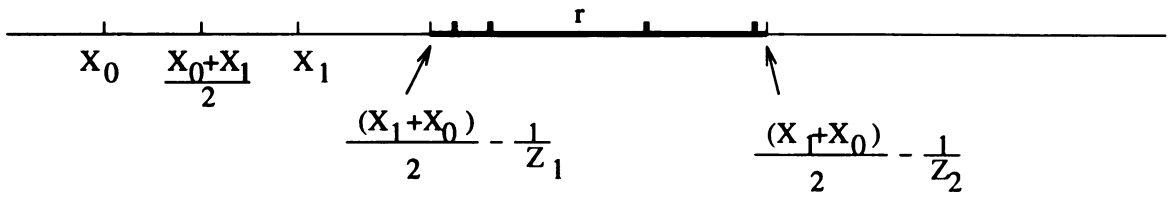

Figure 2.3: Distribution of the roots of the polynomial in case 3

Define

$$\delta_{m\pm} = \frac{nm - (n+m)\frac{\Delta q \Delta x}{4} - q_0 q_1 \frac{\Delta x^2}{4}}{-m\frac{q_0+q_1}{2} \pm \sqrt{-m(n-m)(q_0 q_1 + n\frac{\Delta q}{\Delta x}) + (n\Delta q + q_0 q_1 \Delta x)^2/4}}. \tag{2.29}$$

Note that the set $\{\delta_{m-}, \delta_{m+}\}$ is the same as $\{-\frac{1}{z_1}, -\frac{1}{z_2}\}$.

Now, define

$$QL_{m\pm}(x_0, x_1, q_0, q_1) = \frac{x_0 + x_1}{2} + \delta_{m\pm}. \tag{2.30}$$

This is the same as (2.7). We call $QL_{m\pm}$ the quasi-Laguerre iteration function with multiplicity index m. When there is no ambiguity, we shall write $QL_{m\pm}$ for $QL_{m\pm}(x_0, x_1, q_0, q_1)$. Also for simplicity, if $m = 1$, we write $QL_\pm$ for $QL_{1\pm}$, and $\delta_\pm$ for $\delta_{1\pm}$. In the following theorem, m is taken to be 1.

Theorem 2.2.6 *(1). If δ_- and δ_+ have different signs, then $\delta_- < 0 < \delta_+$. And all the roots of the polynomial $p(x)$ lie outside the interval $[QL_-, QL_+]$.*
(2). If δ_- and δ_+ have the same sign, then $\delta_+ < \delta_-$. And all the roots of the polynomial $p(x)$ lie in the interval $[QL_+, QL_-]$. Furthermore, if $\delta_- < 0$ and $\delta_+ < 0$, then $QL_+ < QL_- < \frac{x_0+x_1}{2}$; if $\delta_- > 0$ and $\delta_+ > 0$, then $\frac{x_0+x_1}{2} < QL_+ < QL_-$.

Proof. We prove the theorem by relating $\delta_\pm$ to $\frac{1}{z_{1,2}}$ and the above three cases. Obviously, any root has multiplicity ≥ 1 and the numerator of $\delta_\pm$(see (2.29)) is positive by Lemma 2.2.4. So only the denominator of $\delta_\pm$ needs to be examined.

If δ_- and δ_+ have different signs and $\delta_+ < 0$, then $-m\frac{q_0+q_1}{2} < 0$, which leads to $\delta_- < 0$, a contradiction. So (1) is proved.

If $\delta_- > 0$ and $\delta_+ > 0$, then

$$-m\frac{q_0 + q_1}{2} > 0$$

and

$$-m\frac{q_0+q_1}{2} > \sqrt{-m(n-m)(q_0q_1+n\frac{\Delta q}{\Delta x})+(n\Delta q+q_0q_1\Delta x)^2/4}.$$

That is, the denominators of δ_- and δ_+ are both positive, and the denominator of δ_- is less than that of δ_+, hence $\delta_- > \delta_+$. Furthermore, we have $\frac{x_0+x_1}{2} < QL_+ < QL_-$.

If $\delta_- < 0$ and $\delta_+ < 0$, then

$$-m\frac{q_0+q_1}{2} < 0$$

and

$$|-m\frac{q_0+q_1}{2}| > \sqrt{-m(n-m)(q_0q_1+n\frac{\Delta q}{\Delta x})+(n\Delta q+q_0q_1\Delta x)^2/4}.$$

So the denominators of δ_- and δ_+ are both negative, and the absolute value of the denominator of δ_- is greater than that of δ_+. Hence, $\delta_- > \delta_+$. Furthermore, we have $QL_+ < QL_- < \frac{x_0+x_1}{2}$. □

The following result can be easily verified.

Corollary 2.2.7

If $\delta_- < 0$ *and* $\delta_+ < 0$, *then* $|\delta_-| < |\delta_+|$.

If $\delta_- > 0$ *and* $\delta_+ > 0$, *then* $|\delta_-| > |\delta_+|$.

Theorem 2.2.8 *Among all the polynomials,* $p(x) = k(x-r)(x-t)^{n-1}$ *is the one that has root closest to the interval* $[x_0, x_1]$.

Proof. It follows from Remark 2.2.2 that $r = QL_\pm$ if only if the polynomial is of the form $p(x) = k(x-r)(x-t)^{n-1}$. The assertion then follows from Theorem 2.2.6. □

We call the polynomial in Theorem 2.2.8 the **optimal** polynomial.

Lemma 2.2.9 *The inequality*

$$|\delta_{m\pm}| > \frac{\Delta x}{2} \tag{2.31}$$

holds.

Proof. The roots of the polynomial must be outside of the interval $[x_0, x_1]$. It is clear that $QL_\pm = \frac{x_0+x_1}{2} + \delta_\pm$ are the roots of the optimal polynomial, so they are outside the interval $[x_0, x_1]$. Therefore $|\delta_{m\pm}| > \frac{\Delta x}{2}$. □

Figure 2.4 illustrates the three cases discussed in Theorem 2.2.6. If one identifies two ends of a straight line as ∞, a uniform circle is obtained. Point A in the figure serves as a reference point so that one can tell where ∞ is on the circle corresponding to the three line cases.

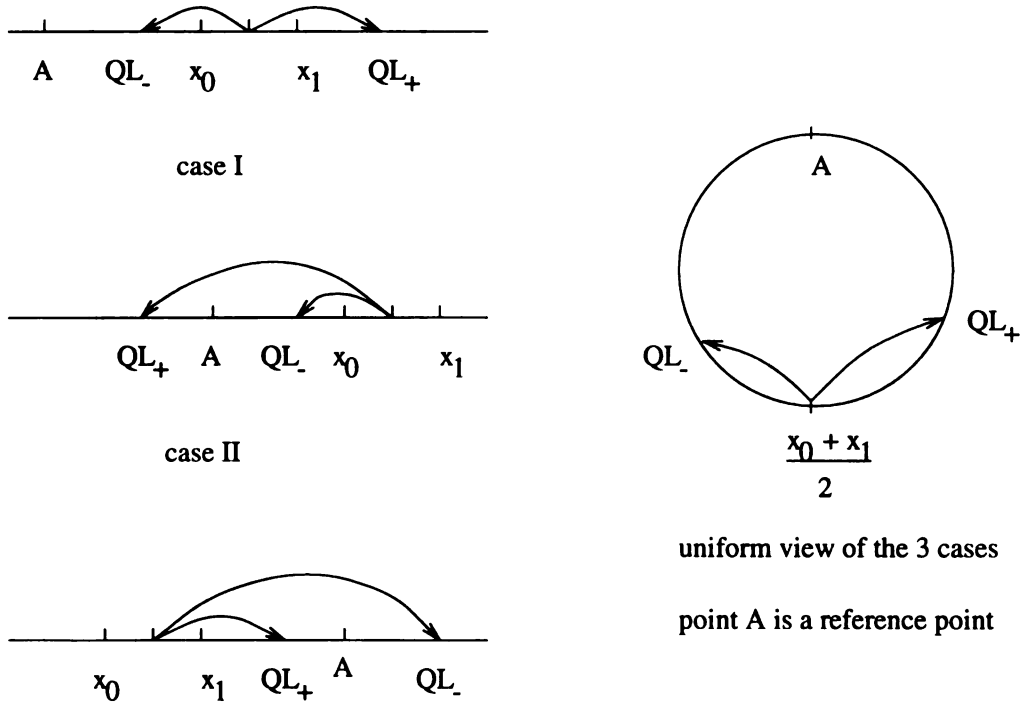

Figure 2.4: Movement of quasi-Laguerre iteration

Proposition 2.2.10 *If* $-q_1 < 0$ *and* $-q_0 < 0$, *then* $|\delta_-| < |\delta_+|$. *If* $-q_1 > 0$ *and* $-q_0 > 0$, *then* $|\delta_-| > |\delta_+|$.

Proof. If $-q_1 < 0$ and $-q_0 < 0$, then the denominator of $|\delta_-|$ is greater than the denominator of $|\delta_+|$. Hence $|\delta_-| < |\delta_+|$. The second case can be achieved similarly. □

For polynomials with only real roots, a monotonically convergent algorithm is obtained and we call it the quasi-Laguerre's algorithm. We describe the algorithm below and deduce some of its properties from the above contents.

Quasi-Laguerre algorithm:

Let $p(x)$ be a polynomial with all real roots. Start with an interval $[x_0, x_1]$ that contains no roots of $p(x)$. Since any root will have multiplicity at least one, we can always use $m = 1$ in the quasi-Laguerre's iteration. If the nearest root of the polynomial is known to be of multiplicity $\geq m$, then m should be used to accelerate the iteration. The following iterative scheme converges monotonically and the iteration sequence never cross the root.

Theorem 2.2.11

1. *Initial step: evaluate* $q_0 = \frac{p'(x_0)}{p(x_0)}, q_1 = \frac{p'(x_1)}{p(x_1)}$.

2. *Choose* m *properly if possible. Otherwise, let* $m = 1$.

3. *Compute* $\delta_\pm$ *according to (2.29). Then, we have*

   (a). *If* $\delta_\pm > 0$, *then* $x_k^+ = \frac{x_{k-2}^+ + x_{k-1}^+}{2} + \delta_+(x_{k-2}^+, x_{k-1}^+, q_{k-2}^+, q_{k-1}^+)$ *converges to a root of* $p(x)$, *where* $x_1^+ = x_1$, $q_1^+ = q_1$, $x_0^+ = x_0$, $q_0^+ = q_0$,

   (b). *If* $\delta_\pm < 0$, *then* $x_k^- = \frac{x_{k-2}^- + x_{k-1}^-}{2} + \delta_-(x_{k-2}^-, x_{k-1}^-, q_{k-2}^-, q_{k-1}^-)$ *converges to a root of* $p(x)$, *where* $x_1^- = x_0$, $q_1^- = q_0$, $x_0^- = x_1$, $q_0^- = q_1$,

   (c). *If* $\delta_- < 0$ *and* $\delta_+ > 0$, *let* $x_1^- = x_0 = x_0^+$, $x_1^+ = x_1 = x_0^-$, $q_1^- = q_0 = q_0^+$, $q_1^+ = q_1 = q_0^-$. *Then,*

   - $x_k^- = \frac{x_{k-2}^- + x_{k-1}^-}{2} + \delta_-(x_{k-2}^-, x_{k-1}^-, q_{k-2}^-, q_{k-1}^-)$ *converges to a root of* $p(x)$, *if* $-q_0 < 0$.
   - $x_k^+ = \frac{x_{k-2}^+ + x_{k-1}^+}{2} + \delta_+(x_{k-2}^+, x_{k-1}^+, q_{k-2}^+, q_{k-1}^+)$ *converges to a root of* $p(x)$, *if* $-q_1 > 0$.

We call the sequence the quasi-Laguerre's iteration sequence.

Proof. It follows from Theorem 2.2.6 and Lemma 2.2.9 that the quasi-Laguerre's iteration sequences converges monotonically. In the following, we will omit superscripts + or −. Assume the limit of the sequence $\{x_k\}_{k=0}^{\infty}$ is r. Let $k \to \infty$ in

$$x_{k+1} = \frac{x_k + x_{k-1}}{2} + \delta(x_k, x_{k-1}, q_k, q_{k-1}),$$

then,

$$\delta(x_k, x_{k-1}, q_k, q_{k-1}) \to 0.$$

That is,

$$\frac{nm - (n+m)\frac{\Delta q_{k-1}\Delta x_{k-1}}{4} - q_{k-1}q_k\frac{(\Delta x_{k-1})^2}{4}}{-m\frac{q_{k-1}+q_k}{2} \pm \sqrt{-m(n-m)(q_{k-1}q_k + n\frac{\Delta q_{k-1}}{\Delta x_{k-1}}) + \left(n\Delta q_{k-1} + q_{k-1}q_k\Delta x_{k-1}\right)^2/4}} \to 0.$$

Lemma 2.2.4 ensures that the numerator of the above fraction is greater than or equal to mn. So,

$$-m\frac{q_{k-1}+q_k}{2} \pm \sqrt{-m(n-m)(q_{k-1}q_k + n\frac{\Delta q_{k-1}}{\Delta x_{k-1}}) + \left(n\Delta q_{k-1} + q_{k-1}q_k\Delta x_{k-1}\right)^2/4} \to \infty.$$

If $p(r) \neq 0$ then the limit should be

$$-m\frac{p'(r)}{p(r)} \pm \sqrt{-m(n-m)((\frac{p'(r)}{p(r)})^2 + n\frac{d}{dx}\frac{p'(r)}{p(r)})} \neq \infty.$$

This contradiction leads to the conclusion that r is a root of $p(x)$. □

Remark 2.2.12 *It follows from Proposition 2.2.10 that if r is a root of $p(x)$ such that $x_k \to r^+$, as $k \to \infty$, i.e., the quasi-Laguerre's iteration sequence is approaching r from the right hand side, then*

$$|\delta_-(x_k, x_{k-1}, q_k, q_{k-1})| < |\delta_+(x_k, x_{k-1}, q_k, q_{k-1})|$$

for k large enough since $-q_k$ and $-q_{k-1}$ would both be negative, see Figure 2.5.

Similarly, if $x_k \to r^-$, as $k \to \infty$, i.e., the quasi-Laguerre's iteration sequence is approaching r from the left hand side, then

$$|\delta_-(x_k, x_{k-1}, q_k, q_{k-1})| > |\delta_+(x_k, x_{k-1}, q_k, q_{k-1})|$$

for k large enough, see Figure 2.5.

Notice that if the quasi-Laguerre's iteration sequence converges, then the sign chosen in the formula is always the one that makes the magnitude of δ smaller than the choice of the other sign. This constitutes an important feature of our algorithm.

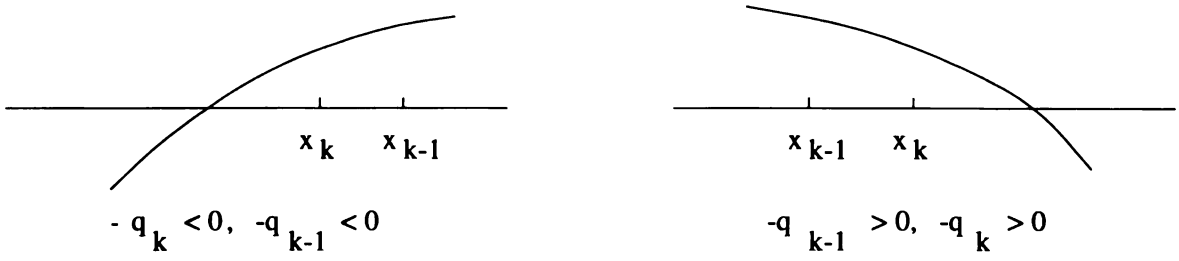

Figure 2.5: sign of $q_k = \frac{f'(x_k)}{f(x_k)}$ and $q_{k-1} = \frac{f'(x_{k-1})}{f(x_{k-1})}$ as the sequence approaches the root

Remark 2.2.13 *Starting from any initial point x_0, two Laguerre's iteration sequences, $x_{k+1}^+ = L_+(x_k^+)$ and $x_{k+1}^- = L_-(x_k^-)$, can be generated and both converge to a (different) root of the polynomial. However, this is not true for the quasi-Laguerre's iteration. The quasi-Laguerre's iteration sequence generated using the + sign, for instance, moves upward (to the right). If there is no root of the polynomial on the right hand side of the initial interval, the iteration sequence will definitely wrap around after some steps of iterations, hence produces a point that is on left hand side of all the roots. When this happens, the iteration usually collapses because there are roots of the polynomial between two consecutive iterates now, hence complex number may be generated with the quasi-Laguerre's function. Numerical experiments have verified this phenomenon. Therefore, the choice of sign in generating quasi-Laguerre's iteration sequence is more delicate. To summarize, if one starts with an interval $[x_0, x_1]$ that contains no root of the polynomial, then the following guidelines can be used to generate iterates: If it is known that there is a root on the left (right, resp.) hand side of the initial interval, then the − sign (+ sign, resp.) is chosen in the quasi-Laguerre's function to generate iteration sequence; If this information can not be obtained, use the criteria in Remark 2.2.12 to generate iteration sequence.*

2.3 From an optimization point of view

A Theorem from optimization

The following result can be found in [22]. Bold faced letters represent point in an Euclidean space E^n of dimension n.

Definition 2.3.1 *Let* $\mathbf{x}^* \in E^n$ *be a point satisfying constraints*

$$\mathbf{h}(\mathbf{x}^*) = 0, \quad \mathbf{g}(\mathbf{x}^*) \leq 0,$$

and let J *be the set of indices* j *for which* $\mathbf{g}_j(x^*) = 0$. *Then* $\mathbf{x}^*$ *is said to be a regular point of the constraints if the gradient vectors* $\nabla \mathbf{h}_i(\mathbf{x}^*), \nabla \mathbf{g}_j(\mathbf{x}^*), 1 \leq i \leq m, j \in J$ *are linearly independent.*

Theorem 2.3.2 (Kuhn-Tucker Conditions) *Let* $\mathbf{x}^*$ *be a solution of the problem*

$$\begin{aligned} &\textit{minimize} \qquad f(\mathbf{x}) \\ &\textit{subject to} \\ &\mathbf{h}(\mathbf{x}) = 0, \qquad (2.32) \\ &\mathbf{g}(\mathbf{x}) \leq \mathbf{0} \qquad (2.33) \end{aligned}$$

and suppose $\mathbf{x}^*$ *is a regular point of the constraints. Then there is a vector* $\lambda \in E^m$ *and a vector* $\mu \in E^p$ *with* $\mu \geq \mathbf{0}$ *such that*

$$\nabla f(x^*) + \lambda^T \nabla \mathbf{h}(x^*) + \mu^T \nabla \mathbf{g}(\mathbf{x}^*) = \mathbf{0} \qquad (2.34)$$

$$\mu^T \mathbf{g}(\mathbf{x}^*) = 0. \qquad (2.35)$$

(2.34) is called the Lagrange equation. (2.35) is called the complimentary condition. Regular point only concerns active constraints.

Now we derive the quasi-Laguerre's formula from the optimization point of view. The basic assumptions on page 9 are still valid in this section. Let's identify the real axis with the unit circle and label the roots, $r_1, r_2, \cdots, r_n$, of $p(x)$ clockwise starting from x_0, see Figure 2.6.

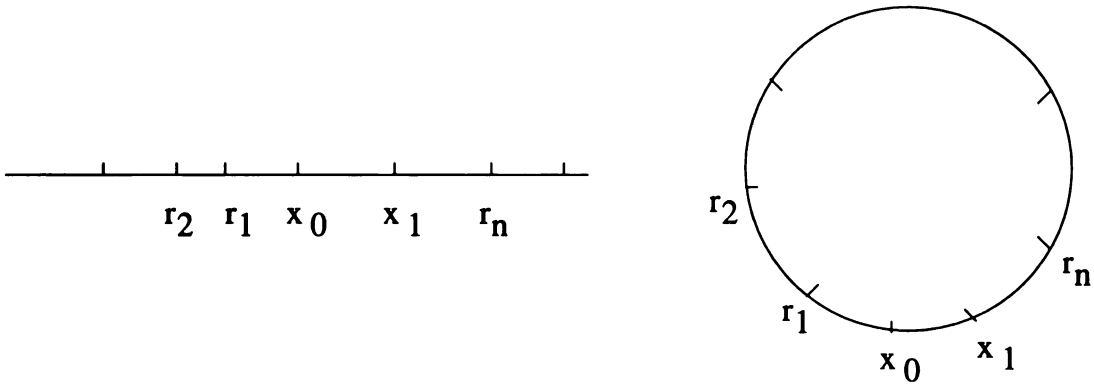

Figure 2.6: Labeling the roots clockwise

Lemma 2.3.3 *If the roots of $p(x)$ is labeled clockwise, then*

$$\frac{1}{r_m - x_0} \geq \frac{1}{r_i - x_0}, \text{ for } 1 \leq i \leq m-1 < n. \tag{2.36}$$

Proof. Consider the following two cases.

Case I: $r_m < x_0$.

Then

$$r_m \leq r_i < x_0.$$

So,

$$r_m - x_0 \leq r_i - x_0.$$

Or,

$$x_0 - r_m \geq x_0 - r_i > 0.$$

Thus,

$$0 < \frac{1}{x_0 - r_m} \leq \frac{1}{x_0 - r_i}.$$

That is,

$$\frac{1}{r_m - x_0} \geq \frac{1}{r_i - x_0}.$$

Case II: $r_m > x_0$.

We consider the following two subcases.

II-1: $r_i < x_0$.

Then

$$\frac{1}{r_m - x_0} > 0 > \frac{1}{r_i - x_0}.$$

II-2: $r_i > x_0$.

Then

$$r_i \geq r_m.$$

So,

$$\frac{1}{r_m - x_0} \geq \frac{1}{r_i - x_0} > 0.$$

□

Therefore, in the clockwise labeling, the farther r_i is to x_0, the larger the quantities $\frac{1}{r_i - x_0}$ is. In the following, we assume $m < n$. Our goal is to look for, based on x_0, x_1, q_0, and q_1, an iteration formula that will not jump over more than $m - 1$ roots of $p(x)$ in the clockwise direction. Such a formula answers the question: *how close to the interval* $[x_0, x_1]$ *is the* m^{th} *root of* $p(x)$. In order to answer this question, we consider the following optimization problem,

$$\begin{aligned}
&\text{minimize} && \frac{1}{r_m - x_0} && (2.37)\\
&\text{subject to} && && \\
& && \sum_{j=1}^{n} \frac{1}{x_0 - r_j} = q_0, && (2.38)\\
& && \sum_{j=1}^{n} \frac{1}{x_1 - r_j} = q_1, && (2.39)\\
& && \frac{1}{r_m - x_0} \geq \frac{1}{r_i - x_0}, \text{ for } i = 1, 2, \cdots, m-1, && (2.40)\\
& && \frac{1}{r_n - x_0} > \frac{1}{r_m - x_0}. && (2.41)
\end{aligned}$$

The constraint (2.41) is to ensure that $p(x)$ has at least two different roots.

Lemma 2.3.4 $(r_1, r_2, \cdots, r_n)$ *is a regular point of the constraints (2.37)-(2.40) if* $r_1 = r_2 = \cdots = r_m$ *and there exists at least one* $j > m$ *such that* $r_j \neq r_m$.

Proof. Write the inequality constraint (2.40) into the following,

$$\frac{1}{x_0 - r_i} - \frac{1}{x_0 - r_m} \geq 0, \text{ for } i = 1, 2, \cdots, m-1,$$

Then the Jacobian matrix of the constraints (2.38)-(2.40) is,

$$\begin{pmatrix} \frac{1}{(x_0-r_1)^2} & \frac{1}{(x_0-r_2)^2} & \cdots & \frac{1}{(x_0-r_{m-1})^2} & \frac{1}{(x_0-r_m)^2} & \frac{1}{(x_0-r_{m+1})^2} & \cdots & \frac{1}{(x_0-r_n)^2} \\ \frac{1}{(x_1-r_1)^2} & \frac{1}{(x_1-r_2)^2} & \cdots & \frac{1}{(x_1-r_{m-1})^2} & \frac{1}{(x_1-r_m)^2} & \frac{1}{(x_1-r_{m+1})} & \cdots & \frac{1}{(x_1-r_n)^2} \\ \frac{1}{(x_0-r_1)^2} & 0 & \cdots & 0 & -\frac{1}{(x_0-r_m)^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{(x_0-r_2)^2} & \cdots & 0 & -\frac{1}{(x_0-r_m)^2} & 0 & \cdots & 0 \\ 0 & 0 & \ddots & \frac{1}{(x_0-r_{m-1})^2} & -\frac{1}{(x_0-r_m)^2} & 0 & \cdots & 0 \end{pmatrix}. \quad (2.42)$$

Multiplying the i^{th} row by -1 and adding the result to the first row, for $i = 3, 4, \cdots, m+1$, and then multiplying the i^{th} row by $-\frac{x_0-r_{i-2}}{x_1-r_{i-2}}$ and adding the result to the second row, for $i = 3, 4, \cdots, m+1$, the above matrix becomes,

$$\begin{pmatrix} 0 & 0 & \cdots & 0 & m\frac{1}{(x_0-r_m)^2} & \frac{1}{(x_0-r_{m+1})^2} & \cdots & \frac{1}{(x_0-r_n)^2} \\ 0 & 0 & \cdots & 0 & \frac{\sum_{i=1}^{m}\frac{(x_0-r_i)^2}{(x_1-r_i)^2}}{(x_0-r_m)^2} & \frac{1}{(x_1-r_{m+1})} & \cdots & \frac{1}{(x_1-r_n)^2} \\ \frac{1}{(x_0-r_1)^2} & 0 & \cdots & 0 & -\frac{1}{(x_0-r_m)^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{(x_0-r_2)^2} & \cdots & 0 & -\frac{1}{(x_0-r_m)^2} & 0 & \cdots & 0 \\ 0 & 0 & \ddots & \frac{1}{(x_0-r_{m-1})^2} & -\frac{1}{(x_0-r_m)^2} & 0 & \cdots & 0 \end{pmatrix}. \quad (2.43)$$

Without loss of generality, we assume $r_n \neq r_m$. Then the submatrix

$$\begin{pmatrix} m\frac{1}{(x_0-r_m)^2} & \frac{1}{(x_0-r_n)^2} \\ \frac{1}{(x_0-r_m)^2}\sum_{i=1}^{m}\frac{(x_0-r_i)^2}{(x_1-r_m)^2} & \frac{1}{(x_1-r_n)} \end{pmatrix} \quad (2.44)$$

is nonsingular since $r_1 = r_2 = \cdots = r_m \neq r_n$. Hence the Jacobian matrix (2.42) is of full rank. □

Theorem 2.3.5 *An optimal solution can be attained only when* $r_1 = r_2 = \cdots = r_m$, *and* $r_{m+1} = r_{m+2} = \cdots = r_n$. *Furthermore* $r_1 = \cdots = r_m = r$ *is given by (2.30).*

Proof. Let $(r_1^*, r_2^*, \cdots, r_n^*)$ be an optimal solution to (2.37)-(2.41) and a regular point of (2.38)-(2.40). According to Kuhn-Tucker Conditions (Theorem 2.3.2), there

exist μ_1, μ_2, s_1, s_2, $\cdots$, s_{m-1} such that

$$\nabla\left(\frac{1}{r_m^*-x_0}+\mu_1(\sum_{j=1}^{n}\frac{1}{x_0-r_j^*}-q_0)+\mu_2(\sum_{j=1}^{n}\frac{1}{x_1-r_j^*}-q_1)\right.$$
$$\left.+\sum_{i=1}^{m-1}s_i(\frac{1}{r_m^*-x_0}-\frac{1}{r_i^*-x_0})\right)=0,$$

and

$$s_i(\frac{1}{r_m^*-x_0}-\frac{1}{r_i^*-x_0})=0,\ \text{for } i=1,2,\cdots,m-1.$$

We have omitted the constraint (2.41) because of the complimentary condition. That is,

$$-\frac{1}{(r_m^*-x_0)^2}+\mu_1\frac{1}{(x_0-r_m^*)^2}+\mu_2\frac{1}{(x_1-r_m^*)^2}-\frac{1}{(r_m^*-x_0)^2}\sum_{i=1}^{m-1}s_i=0, \tag{2.45}$$

$$\mu_1\frac{1}{(x_0-r_j^*)^2}+\mu_2\frac{1}{(x_1-r_j^*)^2}=0,\ \text{for } j=m+1,\cdots,n, \tag{2.46}$$

$$\mu_1\frac{1}{(x_0-r_i^*)^2}+\mu_2\frac{1}{(x_1-r_i^*)^2}+s_i\frac{1}{(r_i^*-x_0)^2}=0,\ \text{for } i=1,2,\cdots,m-1, \tag{2.47}$$

$$s_i(r_m^*-r_i^*)=0,\ \text{for } i=1,2,\cdots,m-1. \tag{2.48}$$

Clearly $\mu_1=0$ would lead to $\mu_2=0$ by (2.46) and vice versa, and this would further lead to $s_i=0$, by (2.47), for all $i=1,2,\cdots,m-1$, a contradiction to (2.45). So none of μ_1,μ_2 is zero. From (2.46) and $\frac{r_j^*-x_0}{r_j^*-x_1}>0$(from the basic assumptions), we have

$$\frac{r_j^*-x_0}{r_j^*-x_1}=\sqrt{\frac{-\mu_1}{\mu_2}},\ \text{for } j=m+1,m+2,\cdots,n.$$

So

$$r_j^*=r_{m+1}^*,\ \text{for } j=m+1,m+2,\cdots,n.$$

Now all r_i^*, for $i=1,2,\cdots,m-1$ must be equal to r_m^*. Otherwise, if $r_1^*\neq r_m^*$ for instance, then $s_1=0$ by (2.40) and the complimentary condition (2.48). Therefore, $\frac{r_1^*-x_0}{r_1^*-x_1}=\sqrt{\frac{-\mu_1}{\mu_2}}$ from (2.47), i.e., $r_1^*=r_n^*$, contradicting to the fact that r_n^* is different from r_m^* (see constraints (2.41)), which implies $r_n^*\neq r_1^*$.

So there are only two different r_i^*'s , $r_1^*=r_2^*=\cdots=r_m^*$, and $r_{m+1}^*=r_{m+2}^*=\cdots=r_n^*$, when the maximum is achieved. We have derived the formula for r when the roots

of the polynomial is of this kind (see (2.7)), and the formula is the quasi-Laguerre's formula(see (2.30)). □

Similarly, if the roots of $p(x)$ is labeled counterclockwisely, starting from the right end of the interval $[x_0, x_1]$, then the m^{th} root is closest to x_1 only when $r_1 = r_2 = \cdots = r_m \neq r_{m+1} = \cdots = r_n$. This again yields the general quasi-Laguerre's iteration formula. The corresponding optimization problem in this case would be maximizing $\frac{1}{r_m - x_1}$ satisfying similar constraints as (2.38)-(2.41). So we have the following properties.

Corollary 2.3.6 *Among all the polynomials that have only real roots (none of the roots is in the interval $[x_0, x_1]$) and satisfy the fundamental constraint (2.38) and (2.39), $p(x) = k(x-r)^m(x-z)^{n-m}$ is the one whose m^{th} root, counting from the interval, clockwise or counterclockwise, is closest to the interval $[x_0, x_1]$.*

Theorem 2.3.7 *$QL_{m\pm}$ computed by (2.30) would never overshoot more than $m-1$ roots of the polynomial $p(x)$, counting from the interval $[x_0, x_1]$, clockwise or counterclockwise.*

Proof. Theorem 2.3.5 showed that the closest m^{th} root to the interval $[x_0, x_1]$, counting from the left (or right) end of the interval, clockwise (or counterclockwise), is given by (2.30). □

Theorem 2.3.8 *Let m be an integer, $1 \leq m \leq n-1$. In the clockwise direction, the larger the m, the farther the QL_{m-} is from x_0. In the counterclockwise direction, the larger the m, the farther the QL_{m+} is from x_1.*

Proof. Let $m < k \leq n-1$ We only prove the case for the clockwise direction. Note QL_{m-} is the closest m^{th} root and QL_{k-} is the closest k^{th} root among all the polynomials with the prescribed logarithmic derivatives, we have $\frac{1}{QL_{k-} - x_0} \geq \frac{1}{QL_{m-} - x_0}$ since $k > m$. □

2.4 For polynomials with complex roots

The quasi-Laguerre's method can also be applied to solve general polynomials or continuously differentiable functions that may have complex roots. The choice of the sign in (2.30), which we developed for polynomials with only real roots, is determined by how the sequence is approaching the root. Negative sign is chosen in (2.30) if the sequence is approaching the root from the right hand side, while positive sign is chosen otherwise. However, for functions with complex roots, one must choose the sign for which the magnitude of $|\delta|$ is the smaller one.

2.5 Convergence order

In this section we will prove the order of convergence for quasi-Laguerre's method is $1+\sqrt{2}$ if the multiplicity index matches the multiplicity of the root. The proof is in the complex plane, hence covers the real case. Write the quasi-Laguerre's formula (2.30) in the following form

$$x_2 = \frac{x_1 + x_0}{2} + \delta_{\pm}(x_0, x_1, q_0, q_1, m), \tag{2.49}$$

where

$$\delta_{\pm}(x_0, x_1, q_0, q_1, m) = \frac{\frac{-m(q_0+q_1)}{2} \pm \sqrt{-m(n-m)[q_0q_1 + n\frac{\Delta q}{\Delta x}] + [q_0q_1 + n\frac{\Delta q}{\Delta x}]^2 \left(\frac{\Delta x}{2}\right)^2}}{q_0q_1 + (n-m)\frac{\Delta q}{\Delta x}}. \tag{2.50}$$

Recall that the choice of $+$ or $-$ in (2.49) depends on the magnitude of $\delta_{\pm}$. For convenience, we let

$$\delta(x_0, x_1, q_0, q_1, m) = \begin{cases} \delta_+ \text{ if } |\delta_+| \leq |\delta_-|, \\ \delta_- \text{ if } |\delta_-| < |\delta_+|. \end{cases}$$

Theorem 2.5.1 *Let $\{x_k\}_{k=1}^{\infty} \subset \mathbf{C}$ be a quasi-Laguerre's iteration sequence in the complex plane, i.e., $x_{k+1} = \frac{x_{k-1}+x_k}{2} + \delta(x_{k-1}, x_k, q_{k-1}, q_k, m)$. If the sequence $\{x_k\}$ converges and the multiplicity index of the formula matches the multiplicity of the converged root, then the order of convergence is $1 + \sqrt{2}$.*

Proof. Let $r \in \mathbf{C}$ be a root of $p(x)$ such that $\lim_{k\to\infty} x_k = r$. write $p(x) = (x-r)^m\varphi(x)$, where $\varphi(r) \neq 0$. Using (2.49), we only need to show that if $|x_1 - r|$ and $|x_0 - r|$ is small, then

$$|x_2 - r| = O(|x_1 - r|^2|x_0 - r|), \text{if } |x_0 - r| = o(1), \text{ and } |x_0 - r| = o(1). \tag{2.51}$$

Write

$$q_0 = \frac{m}{x_0 - r} + \sigma_0, \tag{2.52}$$

$$q_1 = \frac{m}{x_1 - r} + \sigma_1, \tag{2.53}$$

where $\sigma_i = g(x_i)$, for $i = 0, 1$, and $g(x) = \frac{\varphi'(x)}{\varphi(x)}$ is a continuously differentiable function in a neighborhood of r. Then

$$q_0q_1 = \frac{m^2}{(x_0 - r)(x_1 - r)} + \frac{m\sigma_0}{x_1 - r} + \frac{m\sigma_0}{x_0 - r} + \sigma_0\sigma_1, \tag{2.54}$$

$$\frac{\Delta q}{\Delta x} = -\frac{m}{(x_0 - r)(x_1 - r)} + \frac{\Delta\sigma}{\Delta x}, \tag{2.55}$$

where $\frac{\Delta\sigma}{\Delta x} = \frac{\sigma_1 - \sigma_0}{x_1 - x_0}$. Write

$$q_0q_1 + n\frac{\Delta q}{\Delta x} = \frac{m(m-n)}{(x_1 - r)(x_0 - r)} + S_0, \tag{2.56}$$

where

$$S_0 = \frac{m\sigma_0}{x_1 - r} + \frac{m\sigma_1}{x_0 - r} + \sigma_0\sigma_1 + n\frac{\Delta\sigma}{\Delta x}. \tag{2.57}$$

So,

$$\Delta x(q_0q_1 + n\frac{\Delta q}{\Delta x}) = m(m-n)\left(\frac{1}{x_0 - r} - \frac{1}{x_1 - r}\right) + S_0\Delta x, \tag{2.58}$$

and the radicant in (2.49) can be written as,

$$\begin{aligned}
&-m(n-m)\left[q_0q_1 + n\frac{\Delta q}{\Delta x}\right] + \frac{(\Delta x)^2}{4}\left[q_0q_1 + n\frac{\Delta q}{\Delta x}\right]^2 = \frac{m^2(n-m)^2}{(x_0 - r)(x_1 - r)} \\
&-m(n-m)S_0 + \frac{1}{4}\left[m^2(m-n)^2\left(\frac{1}{x_0 - r} - \frac{1}{x_1 - r}\right)^2\right. \\
&\left.+2m(m-n)\left(\frac{1}{x_0 - r} - \frac{1}{x_1 - r}\right)S_0\Delta x + S_0^2(\Delta x)^2\right] \\
= \;& \frac{m^2(n-m)^2}{4}\left[\frac{1}{x_0 - r} + \frac{1}{x_1 - r}\right]^2 - m(m-n)S_0 \\
&+\frac{1}{2}m(m-n)\left(\frac{1}{x_0 - r} - \frac{1}{x_1 - r}\right)S_0\Delta x + \frac{1}{4}S_0^2(\Delta x)^2
\end{aligned}$$

$$
\begin{aligned}
&= \frac{m^2(n-m)^2}{4}\left[\frac{1}{x_0-r}+\frac{1}{x_1-r}\right]^2+\frac{1}{2}m(n-m)\left(\frac{1}{x_0-r}+\frac{1}{x_1-r}\right)S_0\Delta x \\
&\quad +\frac{1}{4}S_0^2(\Delta x)^2+m(m-n)\frac{\Delta x}{x_0-r}S_0-m(m-n)S_0 \\
&= \left[\frac{m(n-m)}{2}\left(\frac{1}{x_0-r}+\frac{1}{x_1-r}\right)+\frac{1}{2}S_0\Delta x\right]^2+m(m-n)\frac{\Delta x}{x_0-r}S_0-m(n-m)S_0 \\
&= S_1^2-m(n-m)\frac{x_1-r}{x_0-r}S_0, \qquad (2.59)
\end{aligned}
$$

where

$$
S_1=\frac{1}{2}m(n-m)\left(\frac{1}{x_0-r}+\frac{1}{x_1-r}\right)+\frac{1}{2}S_0\Delta x. \qquad (2.60)
$$

Assume

$$
|x_1-r|<\alpha|x_0-r|, \qquad (2.61)
$$

for some $0<\alpha<1$. Since the iteration sequence converges, assumption (2.61) is feasible. Hence,

$$
S_0=O(S_1) \text{ and } S_0=o(S_1^2) \text{ if } |x_1-r|=o(1), |x_0-r|=o(1). \qquad (2.62)
$$

Therefore,

$$
\begin{aligned}
&\sqrt{-m(n-m)\left[q_0q_1+n\frac{\Delta q}{\Delta x}\right]+\frac{(\Delta x)^2}{2}\left[q_0q_1+n\frac{\Delta q}{\Delta x}\right]^2} \\
&= \sqrt{S_1^2\left(1-m(m-n)\frac{x_1-r}{x_0-r}\frac{S_0}{S_1^2}\right)} \\
&= \pm S_1\left(1-\frac{1}{2}m(n-m)\frac{x_1-r}{x_0-r}\frac{S_0}{S_1^2}+o((x_1-r)^2)\right). \qquad (2.63)
\end{aligned}
$$

So,

$$
\frac{-m(q_0+q_1)}{2}\pm\sqrt{-m(n-m)[q_0q_1+n\frac{\Delta q}{\Delta x}]+[q_0q_1+n\frac{\Delta q}{\Delta x}]^2\left(\frac{\Delta x}{2}\right)^2} \qquad (2.64)
$$

$$
=\frac{-m(q_0+q_1)}{2}\pm S_1\left(1-\frac{1}{2}m(n-m)\frac{x_1-r}{x_0-r}\frac{S_0}{S_1^2}+o((x_1-r)^2)\right). \qquad (2.65)
$$

It follows from (2.52), (2.53), (2.65) and (2.60) that the dominant term in (2.65) is

$$
\frac{-m^2\pm m(n-m)}{2}\frac{1}{x_1-r}. \qquad (2.66)
$$

Thus, taking the plus sign yields a smaller magnitude in (2.66). Subtracting r from (2.49), and taking the sign in front of the square root that yields a + sign in (2.65), we have

$$\begin{aligned} x_2 - r &= \frac{x_1 - r + x_0 - r}{2} + \frac{\frac{-m(q_0+q_1)}{2} + S_1 - \frac{m(n-m)}{2}\frac{x_1-r}{x_0-r}\frac{S_0}{S_1} + o(x_1-r)}{q_0q_1 + (n-m)\frac{\Delta q}{\Delta x}} \\ &= \frac{S_2 + S_3}{q_0q_1 + (n-m)\frac{\Delta q}{\Delta x}}, \end{aligned} \tag{2.67}$$

where

$$\begin{aligned} S_2 &= \left(\frac{x_1-r}{2} + \frac{x_0-r}{2}\right)\left(q_0q_1 + (n-m)\frac{\Delta q}{\Delta x}\right) \\ &= \left(\frac{x_1-r}{2} + \frac{x_0-r}{2}\right)\left[\frac{m^2}{(x_0-r)(x_1-r)} + \frac{m\sigma_0}{x_1-r} + \frac{m\sigma_1}{x_0-r}\right. \\ &\quad \left. +\sigma_0\sigma_1 - \frac{m(n-m)}{(x_0-r)(x_1-r)} + (n-m)\frac{\Delta\sigma}{\Delta x}\right] \\ &= \left(\frac{x_1-r}{2} + \frac{x_0-r}{2}\right)\left[\frac{m(2m-n)}{(x_0-r)(x_1-r)} + \frac{m\sigma_0}{x_1-r} + \frac{m\sigma_1}{x_0-r}\right. \\ &\quad \left. +\sigma_0\sigma_1 + (n-m)\frac{\Delta\sigma}{\Delta x}\right] \\ &= \frac{m(2m-n)}{2}\frac{1}{x_0-r} + \frac{m}{2}\sigma_0 + \frac{m}{2}\frac{x_1-r}{x_0-r}\sigma_1 + \frac{m(2m-n)}{2}\frac{1}{x_1-r} \\ &\quad +\frac{m}{2}\frac{x_0-r}{x_1-r}\sigma_0 + \frac{m}{2}\sigma_1 + \frac{x_0-r+x_1-r}{2}\left[\sigma_0\sigma_1 + (n-m)\frac{\Delta\sigma}{\Delta x}\right] \\ &= \frac{m(2m-n)}{2}\left[\frac{1}{x_0-r} + \frac{1}{x_1-r}\right] + \frac{m}{2}(\sigma_0+\sigma_1) + \frac{m}{2}\left(\frac{x_1-r}{x_0-r}\sigma_1 + \frac{x_0-r}{x_1-r}\sigma_0\right) \\ &\quad +\frac{x_0-r+x_1-r}{2}\left[\sigma_0\sigma_1 + (n-m)\frac{\Delta\sigma}{\Delta x}\right] \end{aligned} \tag{2.68}$$

$$\begin{aligned} S_3 &= m\frac{q_0+q_1}{2} + S_1 - \frac{m(n-m)}{2}\frac{x_1-r}{x_0-r}\frac{S_0}{S_1} + o(x_1-r) \\ &= -\frac{m^2}{2}\left(\frac{1}{x_0-r} + \frac{1}{x_1-r}\right) - \frac{m}{2}(\sigma_0+\sigma_1) + \frac{1}{2}m(n-m)\left(\frac{1}{x_0-r} + \frac{1}{x_1-r}\right) \\ &\quad +\frac{1}{2}S_0\Delta x - \frac{m(n-m)}{2}\frac{x_1-r}{x_0-r}\frac{S_0}{S_1} + o(x_1-r) \\ &= -\frac{m(2m-n)}{2}\left[\frac{1}{x_0-r} + \frac{1}{x_1-r}\right] - \frac{m}{2}(\sigma_0+\sigma_1) \\ &\quad +\frac{1}{2}S_0\Delta x - \frac{m(n-m)}{2}\frac{x_1-r}{x_0-r}\frac{S_0}{S_1} + o(x_1-r) \end{aligned} \tag{2.69}$$

$$S_2 + S_3 = \frac{m}{2}\left(\frac{x_1-r}{x_0-r}\sigma_1 + \frac{x_0-r}{x_1-r}\sigma_0\right) + \frac{x_0-r+x_1-r}{2}\left[\sigma_0\sigma_1 + (n-m)\frac{\Delta\sigma}{\Delta x}\right]$$

$$
\begin{aligned}
&+\frac{1}{2}S_0\Delta x - \frac{m(n-m)}{2}\frac{x_1-r}{x_0-r}\frac{S_0}{S_1} + o(x_1-r)\\
=\;& \frac{m}{2}\left(\frac{x_1-r}{x_0-r}\sigma_1 + \frac{x_0-r}{x_1-r}\sigma_0\right) + \frac{x_0-r+x_1-r}{2}\left[\sigma_0\sigma_1 + (n-m)\frac{\Delta\sigma}{\Delta x}\right]\\
&+\frac{1}{2}\left(\frac{m\Delta x}{x_1-r}\sigma_0 + \frac{m\Delta x}{x_0-r}\sigma_1 + \sigma_0\sigma_1\Delta x + n\Delta\sigma\right)\\
&-\frac{m(n-m)}{2}\frac{x_1-r}{x_0-r}\frac{S_0}{S_1} + o(x_1-r)\\
=\;& \frac{m}{2}\sigma_0 + \frac{m}{2}\frac{2x_1-x_0-r}{x_0-r}\sigma_1\\
&+\left(\frac{x_0-r+x_1-r}{2} + \frac{\Delta x}{2}\right)\left(\sigma_0\sigma_1 + (n-m)\frac{\Delta\sigma}{\Delta x}\right)\\
&+\frac{m}{2}\Delta\sigma - \frac{m(n-m)}{2}\frac{x_1-r}{x_0-r}\frac{S_0}{S_1} + o(x_1-r)\\
=\;& \frac{m}{2}\sigma_0 + \frac{m}{2}\left[\frac{2x_1-2r}{x_0-r}\sigma_1 - \sigma_1 + \sigma_1 - \sigma_0\right] + (x_1-r)\left[\sigma_0\sigma_1 + (n-m)\frac{\Delta\sigma}{\Delta x}\right]\\
&-\frac{m(n-m)}{2}\frac{x_1-r}{x_0-r}\frac{S_0}{S_1} + o(x_1-r)\\
=\;& m\frac{x_1-r}{x_0-r}\sigma_1 - \frac{m(n-m)}{2}\frac{x_1-r}{x_0-r}\frac{S_0}{S_1}\\
&+(x_1-r)\left[\sigma_0\sigma_1 + (n-m)\frac{\Delta\sigma}{\Delta x}\right] + o(x_1-r)\\
=\;& m\frac{x_1-r}{x_0-r}\left[\sigma_1 - \frac{n-m}{2}\frac{S_0}{S_1}\right]\\
&+(x_1-r)\left[\sigma_0\sigma_1 + (n-m)\frac{\Delta\sigma}{\Delta x}\right] + o(x_1-r), \qquad (2.70)
\end{aligned}
$$

where

$$
\sigma_1 - \frac{n-m}{2}\frac{S_0}{S_1} = \sigma_1 - \frac{n-m}{2}\frac{\frac{m\sigma_0}{x_1-r} + \frac{m\sigma_1}{x_0-r} + \sigma_0\sigma_1 + n\frac{\Delta\sigma}{\Delta x}}{\frac{1}{2}m(n-m)\left(\frac{1}{x_0-r} + \frac{1}{x_1-r}\right) + \frac{1}{2}S_0\Delta x}
$$

$$
\begin{aligned}
&= \frac{\sigma_1 m(n-m)\left(\frac{1}{x_1-r} + \frac{1}{x_0-r}\right) + \sigma_1 S_0\Delta x}{m(n-m)\left(\frac{1}{x_1-r} + \frac{1}{x_0-r}\right) + S_0\Delta x}\\
&\quad - \frac{(n-m)m\left(\frac{\sigma_0}{x_1-r} + \frac{\sigma_1}{x_0-r}\right) - (n-m)(\sigma_0\sigma_1 + n\frac{\Delta\sigma}{\Delta x})}{m(n-m)\left(\frac{1}{x_1-r} + \frac{1}{x_0-r}\right) + S_0\Delta x}\\
&= \frac{n(n-m)\frac{\sigma_1-\sigma_0}{x_1-r} - (n-m)(\sigma_0\sigma_1 - n\frac{\Delta\sigma}{\Delta x}) + \sigma_1 S_0\Delta x}{n(n-m)\left(\frac{1}{x_1-r} + \frac{1}{x_0-r}\right) + S_0\Delta x}\\
&= O(\sigma_1-\sigma_0) + O(x_1-r).
\end{aligned}
$$

So,

$$\begin{aligned} S_2 + S_3 &= m\frac{x_1 - r}{x_0 - r}(O(\sigma_1 - \sigma_0) + O(x_1 - r)) \\ &\quad +(x_1 - r)\left(\sigma_0\sigma_1 + (n-m)\frac{\Delta\sigma}{\Delta x}\right) + o(x_1 - r) \\ &= (x_1 - r)\left[\sigma_0\sigma_1 + (n-m)\frac{\Delta\sigma}{\Delta x}\right. \\ &\quad \left. +\frac{x_1 - x_0}{x_0 - r}\left(O(\frac{\Delta\sigma}{\Delta x}) + O(\frac{x_1 - r}{x_1 - x_0})\right)\right] + o(x_1 - r) \\ &= (x_1 - r)O\left(g^2(r) + (n-m)g'(r) + g'(r) + 1\right) + o(x_1 - r), \quad (2.71) \end{aligned}$$

and,

$$\frac{S_2 + S_3}{q_0 q_1 + (n-m)\frac{\Delta q}{\Delta x}} \approx \frac{(x_1 - r)^2(x_0 - r)O\left(2g^2(r) + 2(n-m)g'(r) + g'(r) + 1\right)}{m(2m-n) + O(x_0 - r)}.$$

That is,

$$x_2 - r \approx O\left(\frac{2g^2(r) + 2(n-m)g'(r) + g'(r) + 1}{m(2m-n)}\right)(x_1 - r)^2(x_0 - r). \quad (2.72)$$

Let the order of convergence be μ, then

$$x_1 - r = O((x_0 - r)^\mu),\ x_2 - r = O((x_1 - r)^\mu).$$

Hence,

$$\mu = 2 + \frac{1}{\mu},$$
$$\mu^2 - 2\mu - 1 = 0,$$

and,

$$\mu = \frac{2 + \sqrt{8}}{2} = 1 + \sqrt{2}.$$

□

Remark 2.5.2 *The following local convergence property of the quasi-Laguerre's iterative method in complex plan is implied by equation (2.72) that: there exits an* $\epsilon > 0$ *such that if* $|x_0 - r| < \epsilon$, $|x_1 - r| < \epsilon$ *and* $|x_1 - r| < \alpha|x_0 - r|$ *for some* $0 < \alpha < 1$, *then the quasi-Laguerre's iteration sequence starting from* x_0, *and* x_1 *converges to* r, *and the convergence order is* $1 + \sqrt{2}$ *if the multiplicity index matches the multiplicity of the root.*

Chapter 3

Estimate multiplicity

3.1 Determining multiplicity of a root

While Quasi-Laguerre's iteration converges super-linearly with convergence order $1+\sqrt{2}$ when the multiplicity index of the method matches the multiplicity of the root, it converges linearly otherwise. It is, therefore, important to estimate the multiplicity of the root in the iteration process.

Let r be a root of the polynomial $p(x)$ with multiplicity m. For an iterative method involving first and second derivatives at one point, the following Lagouandelle's limiting formula [17] can be used to compute the multiplicity of the nearest root

$$m = \lim_{x_k \to r} \frac{p'(x_k)^2}{p'(x_k)^2 - p(x_k)p''(x_k)}. \tag{3.1}$$

In practice, the following formula is used instead,

$$m \approx int\left(\left|\frac{p'(x_k)^2}{p'(x_k)^2 - p(x_k)p''(x_k)}\right|\right), \tag{3.2}$$

where $int(x)$ is the largest integer $\leq x$.

This method requires the evaluation of the second derivative of $p(x)$, therefore it is inappropriate for our iterative method that involves only the first derivative of the polynomial.

Let $x_0, x_1, \cdots, x_k, \cdots$ be an iterative sequence that converges to a root r of $p(x)$

with multiplicity m. Write

$$p(x) = (x-r)^m g(x).$$

Let $q_k = \frac{p'(x_k)}{p(x_k)}$, then

$$q_k = \frac{m}{x_k - r} + \frac{g'(x_k)}{g(x_k)},$$

where $g(x_k) \neq 0$. This gives rise to the following formula for estimating the multiplicity m,

$$m \approx int(|(x_k - x_{k+1})q_k|). \tag{3.3}$$

This formula requires both x_{k-1} and x_k to be close to the root. Otherwise, severe overestimate may occur.

When both q_{k-1} and q_k are known, then a better formula can be designed as follows. Start with the following two equations

$$q_{k-1} = \frac{m}{x_{k-1} - r} + \frac{g'(x_{k-1})}{g(x_{k-1})}, \tag{3.4}$$

$$q_k = \frac{m}{x_k - r} + \frac{g'(x_k)}{g(x_k)}. \tag{3.5}$$

Subtracting (3.4) from (3.5) then dividing the resulting equation by $x_k - x_{k-1}$ yields,

$$\frac{q_k - q_{k-1}}{x_k - x_{k-1}} = \frac{-m}{(x_k - r)(x_{k-1} - r)} \left(1 - \frac{1}{m}(x_{k-1} - r)(x_k - r)(x_k - x_{k-1})h'_k\right), \tag{3.6}$$

where $h'_k = \frac{d}{dx}\left(\frac{g'(\xi_k)}{g(\xi_k)}\right)$ for some ξ_k between x_k and x_{k-1}.

Multiplying (3.4) by (3.5), we obtain

$$q_{k-1}q_k = \frac{m^2}{(x_k - r)(x_{k-1} - r)}(1 + \delta), \tag{3.7}$$

where $\delta = \frac{g'(x_k)}{mg(x_k)}(x_k - r) + \frac{g'(x_{k-1})}{mg(x_{k-1})}(x_{k-1} - r) + \frac{1}{m^2}(x_k - r)(x_{k-1} - r)\frac{g'(x_{k-1})}{g(x_{k-1})}\frac{g'(x_k)}{g(x_k)})$.

Dividing (3.7) by (3.6), we have

$$\frac{q_{k-1}q_k(x_k - x_{k-1})}{q_k - q_{k-1}} = -m\frac{1 + \delta}{1 - \frac{1}{m}(x_{k-1} - r)(x_k - r)(x_k - x_{k-1})h'_k}. \tag{3.8}$$

So,

$$m = -\frac{q_{k-1}q_k(x_k - x_{k-1})}{q_k - q_{k-1}} \frac{1 - \frac{1}{m}(x_{k-1} - r)(x_k - r)(x_k - x_{k-1})h'_k}{1 + \delta}.$$

Hence, the following formula can be used to estimate the multiplicity m,

$$m \approx \frac{q_{k-1}q_k(x_k - x_{k-1})}{q_{k-1} - q_k}. \tag{3.9}$$

This method is tested on the following polynomial of degree 23,

$$p(x) = (x+1)^3(x-1)(x-3)(x-3.0000000999991)(x-3.1000001)^{14}(x-10.5)(x-20.0)^2.$$

Start from $x_1 = 4.5, x_0 = 7.4$, and aim at the root 3.1000001 of multiplicity 14.

If multiplicity index $mul = 1$ is used in our quasi-Laguerre's iteration, it would take 101 iteration steps to converge to the root. If we use (3.9) to estimate the multiplicity when $|(x_{k+1} - x_k)| < 1.0 \times 10^{-3}$, then the computation result shows that after 20 iterations with $mul = 1$, condition $|(x_{20} - x_{19})| < 1.0^{-3}$ is satisfied, and (3.9) is then applied to obtain the true multiplicity 14. When $mul = 14$ is used in the quasi-Laguerre's iteration, it takes only three more iterations to converge to the root 3.1000001.

To approach root 3.1000001 of multiplicity 14, with $mul = 1$ it took 20 iteration steps for the condition $|(x_k - x_{k+1})| < 1.0 \times 10^{-3}$ to be satisfied. If the multiplicity estimation is started earlier, faster convergence can be expected. But, over estimate may occur. The consequence of the over estimation could be the following,

(1) Cause the convergence to march a longer distance toward the target.

(2) Hurt the iteration process by jumping over the root.

In case (2), a back up scheme is necessary to restore the iteration process.

The following back up scheme is used in our experiments,

1. give up the new point,

2. reduce the estimated multiplicity index *mul* by 1 and recompute a new point using the reduced multiplicity index,

3. check overshoot, if still overshoot, reduce *mul* by 1 again to recompute a new point,

4. repeat this process until no overshoot.

A more complicated issue is how to detect overshoot. The following method is useful in many situations. As the sequence is approaching to a root from one side(left or right), the sign of $\frac{p'(x)}{p(x)}$ should stay the same. Therefore, if different signs are obtained at two consecutive iteration points, an overshoot may have occurred. For some specific problems, more reliable tools for detecting overshoot are available. For instance, when our method is used to solve the symmetric tridiagonal eigenvalue problem, the Sturm sequence provides reliable information for detecting whether an overshoot occurs.

The following computation result is obtained using the backup scheme stated above. Estimation of the multiplicity starts when $|x_k - x_{k-1}| < 0.1$, $\frac{f'(x_{k+1})}{f(x_{k+1})} > 0$, $\frac{f'(x_k)}{f(x_k)} > 0$ and $0.1 < \left|\frac{x_{k+1}-x_k}{x_k-x_{k-1}}\right| < 1.0$. The advantage of the multiplicity estimation formula is fully exhibited in this result. Overestimated *mul* values, 17 and 15, are used both successfully once, without overshooting the root. It helped the iteration process advance to a closer position to the root. The total number of iteration is only 9, a 10 times speed up compared to the process using $mul = 1$.

```
starting points          starting logarithmic derivatives
x_0=   7.4000000000000 -p'(x0)/p(x0)=   -3.7424415421671
x_1=   4.5000000000000 -p'(x1)/p(x1)=   -11.868803998501
new points               marching distance          mul-index
x_2=   4.3230028988455  dlt=  -0.17699710115449     mul=  1
x_3=   3.9224125629059  dlt=  -0.40059033593962     mul=  17
x_4=   3.0665236579569  dlt=  -0.85588890494904     mul=  17
x_5=   3.0957747133912  dlt=  -0.82663784951471     mul=  16
x_6=   3.1240282732094  dlt=  -0.79838428969651     mul=  15
x_7=   3.0989338356082  dlt=  -2.5094437601137D-02  mul=  15
x_8=   3.1005358217376  dlt=  -2.3492451471832D-02  mul=  14
x_9=   3.1000001611582  dlt=  -5.3566057939210D-04  mul=  14
x_10=  3.1000001000000  dlt=  -6.1158159848123D-08  mul=  14
```

```
total time for this root =  2.89380E-02sec.  1.71600E-02sec.
total number of iterations=  9
```

The following computation result is obtained when the above method is used to the polynomial.

$$p(x) = (x+1)^3(x-1)(x-3)(x-3.10000009999999)(x-3.1000001)^{14}(x-10.5)(x-20.0)^2.$$

Note roots 3.10000009999999 and 3.1000001000000 are not equal, but extremely close, to each other in the double precision environment. The numerical result shows that 3.1000001000000 is estimated as a root of multiplicity 15, even though it's actual multiplicity is 14.

```
starting points          starting logarithmic derivatives
x_0=   7.4000000000000 -p'(x0)/p(x0)=   -3.7477269546723
x_1=   4.5000000000000 -p'(x1)/p(x1)=   -11.916423052696
new points               marching distance          mul-index
x_2=   4.3239509868566  dlt=  -0.17604901314338     mul=  1
x_3=   3.9247571786955  dlt=  -0.39919380816112     mul=  17
x_4=   3.0720825767638  dlt=  -0.85267460193171     mul=  17
x_5=   3.1012166752925  dlt=  -0.82354050340303     mul=  16
x_6=   3.0999200729812  dlt=  -1.2966023113043D-03  mul=  16
x_7=   3.1000009568750  dlt=  -1.2157184174845D-03  mul=  15
x_8=   3.1000001000000  dlt=  -8.5687498791535D-07  mul=  15
x_9=   3.1000001000000  dlt=  -4.6542100595640D-15  mul=  15
total time for this root =  2.23110E-02sec.  1.36170E-02sec.
total number of iterations=  8
```

Remark 3.1.1 *Stopping criteria for the above numerical results is* $|x_{k+1} - x_k| < \epsilon$ *or* $\frac{f'(x_k)}{f(x_k)} < \epsilon$, *where* ϵ *is the machine precision for double precision numbers.*

Remark 3.1.2 *In this two examples where roots of the polynomials are known, overshooting is easily detected. Genrerally, detecting overshoot is a difficult subject and we*

do not attempt to address this problem here. However, in the application to symmetric tridiagonal eigenvalue problems, the Sturm sequence precisely detects the number of roots jumped.

3.2 A new stopping criteria

Kahan [16] has suggested the following stopping criteria for Laguerre's iteration:

$$|x_{k+1} - x_k|^2 \leq (|x_k - x_{k-1}| - |x_{k+1} - x_k|)\tau, \tag{3.10}$$

where τ denotes the error tolerance. This criteria is based on the following observations. Let

$$r_k = \left|\frac{x_{k+1} - x_k}{x_k - x_{k-1}}\right|.$$

Then as $\{x_k\}_{k=1}^{\infty}$ converges to λ when $k \to \infty$, r_k is normally decreasing. Thus

$$|\lambda - x_{k+1}| = \left|\sum_{i=0}^{\infty}(x_{k+2+i} - x_{k+1+i})\right| \leq \sum_{i=0}^{\infty}|x_{k+2+i} - x_{k+1+i}|$$

$$\leq |x_{k+1} - x_k|\sum_{i=1}^{\infty} r_k^i = \frac{r_k|x_{k+1} - x_k|}{1 - r_k}$$

$$= \frac{|x_{k+1} - x_k|^2}{|x_k - x_{k-1}| - |x_{k+1} - x_k}|.$$

For the quasi-Laguerre iteration, we propose the following stopping criteria,

$$|x_{(k+1)} - x_{(k)}| \times \left|\frac{q_{k+1}}{q_k}\right|^{1+\sqrt{2}} < \tau \tag{3.11}$$

where $q_{k+1} = \frac{p'(x_{k+1})}{p(x_{k+1})}$, $q_{k1} = \frac{p'(x_k)}{p(x_k)}$. This criteria is quite efficient when it is used in solving the symmetric tridiagonal eigenvalue problem. The left hand side of the above inequality is actually a prediction for the distance $|x_{k+2} - \lambda|$. This prediction is based on the rate of convergence of the quasi-Laguerre's iteration. We have shown in Theorem 2.5.1 that

$$|x_{k+2} - \lambda| \approx |x_{k+1} - \lambda|^2|x_k - \lambda| \tag{3.12}$$

$$\approx |x_{k+1} - x_k||x_{k+1} - \lambda||x_k - \lambda|, \tag{3.13}$$

and

$$|x_{k+1} - \lambda| \approx |x_k - \lambda|^{1+\sqrt{2}}. \tag{3.14}$$

It follows that

$$\left|\frac{q_k}{q_{k+1}}\right|^{1+\sqrt{2}} \approx \left|\frac{x_{k+1} - \lambda}{x_k - \lambda}\right|^{1+\sqrt{2}} = |x_{k+1} - \lambda|\frac{|x_{k+1} - \lambda|^{\sqrt{2}}}{|x_k - \lambda|^{1+\sqrt{2}}}$$

$$\approx |x_{k+1} - \lambda|\frac{|x_k - \lambda|^{(1+\sqrt{2})\times\sqrt{2}}}{|x_k - \lambda|^{1+\sqrt{2}}} = |x_{k+1} - \lambda||x_k - \lambda|.$$

So, by (3.13) and (3.14), we have

$$|x_{k+2} - \lambda| \approx |x_{k+1} - x_k|\left|\frac{q_k}{q_{k+1}}\right|^{1+\sqrt{2}}.$$

In practice, power of $1+\sqrt{2}$ should be avoid in floating point computation, so the following inequality is tested for stopping

$$|x_{k+1} - x_k|\left|\frac{q_k}{q_{k+1}}\right|^2 < \tau. \tag{3.15}$$

Chapter 4

Application to symmetric tridiagonal eigenproblem

4.1 Introduction

In this chapter, we shall use the tools developed in previous chapters to approximate all eigenvalues of symmetric tridiagonal matrices.

4.1.1 Evaluation of the logarithmic derivative f'/f of the determinant

Let T be a symmetric tridiagonal matrix of the form

$$T = [\beta_{i-1}, \alpha_i, \beta_i] = \begin{pmatrix} \alpha_1 & \beta_1 & & & & \\ \beta_1 & \alpha_2 & \beta_2 & & 0 & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ & & & & \beta_{n-1} & \alpha_n \end{pmatrix}. \tag{4.1}$$

We may assume, without loss of generality, that T is unreduced; that is, $\beta_j \neq 0,\ j = 1, \cdots, n-1$. For an unreduced T, the characteristic polynomial

$$f(\lambda) \equiv \det(T - \lambda I) \tag{4.2}$$

has only *real* and *simple* zeros ([30], p300). In order to use our quasi-Laguerre iteration developed in previous chapters for finding zeros of $f(\lambda)$, or the eigenvalues of T, it is necessary to evaluate f and f' efficiently with satisfactory accuracy in the first place.

It is well known that the characteristic polynomial $f(\lambda)$ and its derivative with respect to λ can be evaluated by three-term recurrences ([30], p423):

$$\begin{cases} \rho_0 = 1, \quad \rho_1 = \alpha_1 - \lambda \\ \rho_i = (\alpha_i - \lambda)\rho_{i-1} - \beta_{i-1}^2 \rho_{i-2}, \quad i = 2, 3, \ldots, n \end{cases} \tag{4.3}$$

$$\begin{cases} \rho_0' = 0, \quad \rho_1' = -1 \\ \rho_i' = (\alpha_i - \lambda)\rho_{i-1}' - \rho_{i-1} - \beta_{i-1}^2 \rho_{i-2}', \quad i = 2, 3, \ldots, n \end{cases} \tag{4.4}$$

and

$$f(\lambda) = \rho_n, \qquad f'(\lambda) = \rho_n'.$$

However, these recurrences may suffer from a severe underflow-overflow problem and require constant testing and scaling. The following modified recurrence equations [19] is the result of careful investigation of the problem and is more stable than the code presented in [23]. It computes the logarithmic derivative $q(\lambda) = f'(\lambda)/f(\lambda)$, required in our quasi-Laguerre's formula. Let

$$\xi_i = \frac{\rho_i}{\rho_{i-1}}, \qquad \eta_i = -\frac{\rho_i'}{\rho_i},$$

$$\begin{cases} \xi_1 &= \alpha_1 - \lambda, \\ \xi_i &= \alpha_i - \lambda - \dfrac{\beta_{i-1}^2}{\xi_{i-1}}, \quad i = 2, 3, \cdots, n, \end{cases} \tag{4.5}$$

$$\begin{cases} \eta_0 = 0, \quad \eta_1 = \dfrac{1}{\xi_1}, \\ \eta_i = \dfrac{1}{\xi_i}\left[(\alpha_i - \lambda)\eta_{i-1} + 1 - \left(\dfrac{\beta_{i-1}^2}{\xi_{i-1}}\right)\eta_{i-2}\right], \quad i = 2, 3, \cdots, n \end{cases} \tag{4.6}$$

and

$$-\frac{f'(\lambda)}{f(\lambda)} = \eta_n.$$

To prevent the algorithm from breaking down when $\xi_i = 0$ for some $1 \le i \le n$, an extra check [19] is provided:

- If $\xi_1 = 0$ (i.e., $\alpha_1 = \lambda$), set $\xi_1 = \beta_1^2 \varepsilon^2$;

- If $\xi_i = 0$, $i > 1$, set $\xi_i = \dfrac{\beta_{i-1}^2 \epsilon^2}{\xi_{i-1}}$;

where ϵ is the machine precision. A determinant evaluation subroutine DETEVL has been implemented [19] according to the recurrences (4.5) and (4.6). When ξ_i, $i = 1, \cdots, n$ are known, the Sturm sequence is available ([25], p47). Thus, as a by-product, DETEVL also evaluates the number of eigenvalues of T which are less than λ.

Let $\lambda_1 < \lambda_2 < \cdots < \lambda_n$ be the zeros of $f(\lambda)$ and $\hat{\lambda}_1 < \hat{\lambda}_2 < \cdots < \hat{\lambda}_n$ be the zeros of the numerical approximation $\hat{f}(\lambda)$, it was shown in [19] that

$$|fl(\hat{\lambda}_i) - \lambda_i| \leq \frac{5\epsilon}{2} \max_j \{|\beta_j| + |\beta_{j+1}|\} + |\lambda_i|\epsilon. \tag{4.7}$$

4.1.2 The split-merge process

Let

$$\lambda_1 < \lambda_2 < \cdots < \lambda_n$$

be the zeros of f in (4.2). To use our quasi-Laguerre's iteration to approximate any λ_i, $i = 1, 2, \cdots, n$, it is essential to provide a pair of starting points $x^{(0)}$ and $x^{(1)}$, being either $x^{(0)} < x^{(1)} < \lambda_i$ or $\lambda_i < x^{(1)} < x^{(0)}$, with no other λ_j's lying between $x^{(0)}$, $x^{(1)}$ and λ_i. For this purpose, we *split* the matrix T into

$$\hat{T} = \begin{pmatrix} T_0 & 0 \\ 0 & T_1 \end{pmatrix} \tag{4.8}$$

where

$$T_0 = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{k-1} \\ & & \beta_{k-1} & \alpha_k - \beta_k \end{pmatrix}, \quad T_1 = \begin{pmatrix} \alpha_{k+1} - \beta_k & \beta_{k+1} & & \\ \beta_{k+1} & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ & & \beta_{n-1} & \alpha_n \end{pmatrix}. \tag{4.9}$$

Obviously, the eigenvalues of $\hat{T}$ consist of eigenvalues of T_0 and T_1. Without loss of generality, we may assume $\beta_i > 0$, for all $i = 1, 2, \cdots, n-1$, since in (4.3)–(4.6), β_i's always appear in their square form. The following interlacing property for this rank-one tearing is important to our algorithm.

Theorem 4.1.1 *Let* $\lambda_1 < \lambda_2 < \cdots < \lambda_n$ *and* $\hat{\lambda}_1 \leq \hat{\lambda}_2 \leq \cdots \leq \hat{\lambda}_n$ *be eigenvalues of* T *and* $\hat{T}$ *respectively. Then*

$$\hat{\lambda}_1 \leq \lambda_1 \leq \hat{\lambda}_2 \leq \lambda_2 \leq \cdots \leq \hat{\lambda}_n \leq \lambda_n < \hat{\lambda}_{n+1}$$

with the convention $\hat{\lambda}_{n+1} = \hat{\lambda}_n + 2\beta_k$.

Proof. See [11, Theorem 8.6.2, p462]. □

The eigenvalues of $\hat{T}$ will be used critically to approximate the eigenvalues of T by our quasi-Laguerre iteration. We shall call this procedure, splitting T into T_0 and T_1 of $\hat{T}$ and using eigenvalues of $\hat{T}$, consisting of eigenvalues of T_0 and T_1, to approximate eigenvalues of T, the *split-merge* process, similar to Cuppen's divide-and-conquer strategy [4] of course.

From Theorem 4.1.1, we have

$$\hat{\lambda}_i \leq \lambda_i \leq \hat{\lambda}_{i+1}.$$

So, to evaluate certain eigenvalue λ_i, $i = 1, 2, \cdots, n$, of T, we start with the mid-point,

$$p = \frac{\hat{\lambda}_i + \hat{\lambda}_{i+1}}{2}$$

and evaluate $\dfrac{f'(p)}{f(p)}$ by the subroutine DETEVL [19]. Note that the Sturm sequence at p, which decides the position of p relative to λ_i, is a by-product of this evaluation. Based on the information obtained about p, we either use global Newton (see (4.15)) or Newton's method plus bisection adjustment to find the second starting point. To use global Newton's method, we must have $-\frac{f'(p)}{f(p)} > 0$ and $p < \lambda_i$ or $-\frac{f'(p)}{f(p)} < 0$ and $p > \lambda_i$, and there is another point $p0$ such that no eigenvalues lie between $p0$ and p. So if $p < \lambda_i$(determined by Sturm's sequence) and $-\frac{f'(p)}{f(p)} > 0$, then global Newton's method is used to find the second initial point. Since the previous eigenvalue λ_{i-1} has been found, we choose $p0$ to be $\lambda_{i-1} + \epsilon$ in global Newton's method. Otherwise, if $-\frac{f'(p)}{f(p)} < 0$ and $p < \lambda_i$ (or $-\frac{f'(p)}{f(p)} > 0$ and $p > \lambda_i$) then we use bisection method to examine the midpoint $\frac{p+b}{2}$ (or $\frac{a+p}{2}$, respectively); if $-\frac{f'(p)}{f(p)} < 0$ and $p > \lambda_i$ (or $-\frac{f'(p)}{f(p)} > 0$ and $p < \lambda_i$) then we use Newton's method plus bisection adjustment to

examine the point $min(p - \frac{f'(p)}{f(p)}, \frac{p+b}{2})$ (or $max(p - \frac{f'(p)}{f(p)}, \frac{a+p}{2})$, respectively). Repeat this process until two initial points $p0$ and p are found with $\frac{f'(p)}{f(p)} > 0$ if $p < \lambda_i$ (or $\frac{f'(p)}{f(p)} < 0$ if $p > \lambda_i$). The advantage of global Newton's method is that the second initial point can be obtained without extra call to the subroutine DETEVL, which is the most expensive part of the whole algorithm. Once two initial points $p0$ and p are found, the following quasi-Laguerre's iteration formula (see 2.10) with initial points $x_0 = p0$, $x_1 = p$ and initial multiplicity index $m = 1$ is used,

$$x_{m\pm}^{(k+1)} = x_{m\pm}^{(k)} + \frac{n + q(x_{m\pm}^{(k-1)})(x_{m\pm}^{(k)} - x_{m\pm}^{(k-1)})}{-q(x_{m\pm}^{(k-1)}) - (x_{m\pm}^{(k)} - x_{m\pm}^{(k-1)})S \pm \sqrt{S\left[(x_{m\pm}^{(k)} - x_{m\pm}^{(k-1)})^2 S + (m-n)\right]}} \tag{4.10}$$

where

$$S = \frac{1}{2m}\left(q(x_{m\pm}^{(k-1)})q(x_{m\pm}^{(k)}) + n\frac{q(x_{m\pm}^{(k)}) - q(x_{m\pm}^{(k-1)})}{x_{m\pm}^{(k)} - x_{m\pm}^{(k-1)}}\right), \tag{4.11}$$

and

$$q(x_{m\pm}^{(k)}) = \frac{f'(x_{m\pm}^{(k)})}{f(_{m\pm}^{(k)})}. \tag{4.12}$$

Multiplicity index $m > 1$ may be necessary when clusters exist. This will be addressed in Section 4.1.4.

The iteration sequence $\left\{x_{\pm}^{(k)}\right\}_{k=1}^{\infty}$ obtained by (4.10) with an appropriately chosen sign (see Remark 2.2.12 in Chapter 2) converges monotonically to λ_i with ultimate convergence rate $\sqrt{2}+1$ by Theorem 2.5.1.

The eigenvalues of $\hat{T}$ in (4.8) consist of eigenvalues of T_0 and T_1 in (4.9). To find eigenvalues of T_0 and T_1, the split-merge process described above may be applied again. Indeed, the splitting process can be applied to T recursively (See Figure 4.1) until 2×2 and 1×1 matrices are reached.

After T is well split into a tree structure as shown in Figure 4.1, the merging process in the reverse direction from 2×2 and 1×1 matrices can be started. More specifically, let T_σ be split into $T_{\sigma 0}$ and $T_{\sigma 1}$. Let $\hat{\lambda}_1^\sigma, \cdots, \hat{\lambda}_m^\sigma$ be eigenvalues of $\hat{T}_\sigma = \begin{pmatrix} T_{\sigma 0} & 0 \\ 0 & T_{\sigma 1} \end{pmatrix}$ in ascending order. Then the quasi-Laguerre iteration is applied to the

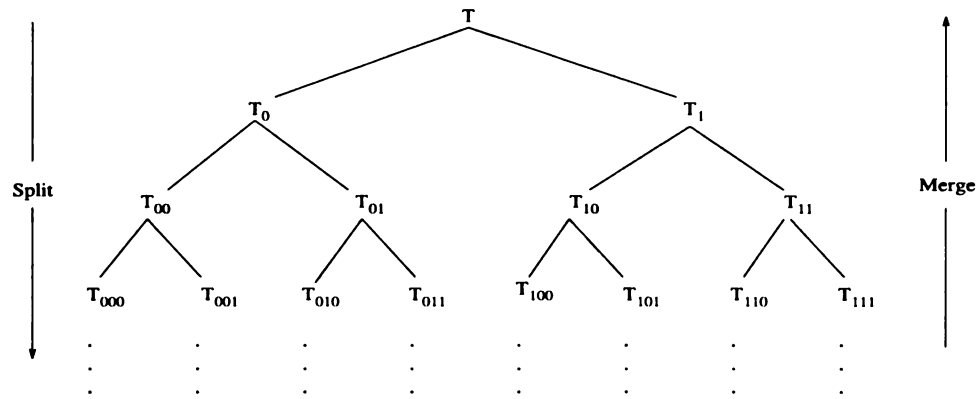

Figure 4.1: Split and merge processes

polynomial equation

$$f_\sigma(\lambda) \equiv \det[T_\sigma - \lambda I] = 0$$

to obtain the corresponding eigenvalue λ_i^σ, $i = 1, 2, \cdots, m$, by the merging process described above. This process continues until T_0 and T_1 are merged into T. That is, in the final step all the eigenvalues of T are obtained by applying the quasi-Laguerre iteration to $f(\lambda) = \det(T - \lambda I)$ from eigenvalues of T_0 and T_1.

4.1.3 Deflation

By Theorem 4.1.1, $\lambda_i \in (\hat{\lambda}_i, \hat{\lambda}_{i+1})$ for each $i = 1, \cdots, n$ with the convention $\hat{\lambda}_{n+1} = \hat{\lambda}_n + 2\beta_k$. If $\hat{\lambda}_{i+1} - \hat{\lambda}_i$ is less than the error tolerance, then either $\hat{\lambda}_i$ or $\hat{\lambda}_{i+1}$ can be accepted as λ_i. In general, if $\hat{T}$ has a cluster of $m + 1$ very close eigenvalues, for instance, $\hat{\lambda}_{j+m} - \hat{\lambda}_j$ is less than the error tolerance for certain $1 \leq j \leq n - m$, then m eigenvalues $\lambda_j, \lambda_{j+1}, \cdots, \lambda_{j+m-1}$ of T can be obtained free of computations. They can be set to any one of $\hat{\lambda}_j, \cdots, \hat{\lambda}_{j+m}$.

4.1.4 Cluster and cloud handler

Since the matrix T in (4.1) is unreduced, its eigenvalues are all simple. Therefore using $m = 1$ in the quasi-Laguerre iterations seems appropriate in all cases to obtain ultimate super-linear convergence with convergence rate $\sqrt{2} + 1$.

However, in some occasions, there may exist a group of eigenvalues of T, say,

$$\lambda_{i+1} < \lambda_{i+2} < \cdots < \lambda_{i+r}$$

which are relatively close to each other, compared to other eigenvalues. We say the spectrum has a cloud. For example, type-3 matrices (see Section 4.3.1) have two clouds of eigenvalues. On some other occasions, some eigenvalues may even be numerically indistinguishable. For instance, the spectrum of Wilkinson matrix [30] contains eigenvalues mostly in pairs and numerically indistinguishable. We call this type of eigenvalues a cluster. Figure 4.2 illustrates these situations.

Spectrum has two clouds

Spectrum has clusters

Figure 4.2: Cluster and cloud of eigenvalues

Definition 4.1.2 *If m roots of $f(x)$ are numerically indistinguishable, we say the function has a cluster of roots with cluster size m.*

Definition 4.1.3 *If m roots of $f(x)$, being gathered in an interval, are relatively far from the other roots, we say $f(x)$ has a cloud of roots in that interval and the cloud size is m.*

Numerical experiments exhibited slow convergence of quasi-Laguerre's iteration with multiplicity index $mul = 1$ in case of clouds and clusters. Hence a cloud and cluster handler is needed to speed up the convergence. Two examples are listed below to show the slow convergence of the quasi-Laguerre's iteration in these situations.

Example 1. Wilkinson matrix W_{99}^{+}. The eigenvalues of this matrix consist mostly of numbers in pairs that are numerically indistinguishable. In particular, we look at eigenvalue # 23, that is equal to 11.000000000000. The following numerical result is obtained from the quasi-Laguerre's iteration process with multiplicity index equal to 1 through out the whole iteration.

Numerical Result of Quasi-Laguerre with Mul-index 1

starting points

p0 = 11.250000000000 p1 = 11.137888560412

new points		marching distance	marching ratio
p2 =	11.057728240655	-8.0160319756418D-02	0.71500571262537
p3 =	11.023732381883	-3.3995858772538D-02	0.42409834286890
p4 =	11.009560676203	-1.4171705679063D-02	0.41686564748620
p5 =	11.003851776062	-5.7089001419764D-03	0.40283789906888
p6 =	11.001550472961	-2.3013031001457D-03	0.40310796176390
p7 =	11.000624132107	-9.2634085422377D-04	0.40252883427876
p8 =	11.000251228591	-3.7290351638350D-04	0.40255540353554
p9 =	11.000101125531	-1.5010305952183D-04	0.40252519197879
p10=	11.000040705300	-6.0420231328607D-05	0.40252498197625
p11=	11.000016384787	-2.4320513144343D-05	0.40252267509654
p12=	11.000006595237	-9.7895493107211D-06	0.40252231737957
p13=	11.000002654728	-3.9405095313399D-06	0.40252205758077
p14=	11.000001068586	-1.5861417018320D-06	0.40252198077863
p15=	11.000000430129	-6.3845684136714D-07	0.40252194405438
p16=	11.000000173137	-2.5699288044115D-07	0.40252193067718
p17=	11.000000069691	-1.0344526894853D-07	0.40252192500802
p18=	11.000000028052	-4.1638989052076D-08	0.40252192754020
p19=	11.000000011292	-1.6760605649697D-08	0.40252191590758
p20=	11.000000004545	-6.7465114403695D-09	0.40252193634135
p21=	11.000000001830	-2.7156191175802D-09	0.40252197622175
p22=	11.000000000736	-1.0930956358719D-09	0.40252170445976
p23=	11.000000000296	-4.3999528135870D-10	0.40252221939185
p24=	11.000000000119	-1.7710815504780D-10	0.40252285092899
p25=	11.000000000048	-7.1289803041994D-11	0.40252129001486
p26=	11.000000000019	-2.8695184519433D-11	0.40251457143920
p27=	11.000000000008	-1.1550485192110D-11	0.40252346815500

```
p28=  11.000000000003    -4.6498264506008D-12    0.40256546571541
p29=  11.000000000001    -1.8709902983900D-12    0.40237852278298
p30=  11.000000000001    -7.5361704612309D-13    0.40279046169913
p31=  11.000000000000    -3.0360953346465D-13    0.40286977985244
p32=  11.000000000000    -1.2186073771329D-13    0.40137322541445
p33=  11.000000000000    -4.8424972315608D-14    0.39737960908740
p34=  11.000000000000    -1.9584490177560D-14    0.40442955857401
p35=  11.000000000000    -6.8926651173973D-15    0.35194508791937
Total number of iterations is 35.
```

Example 2. Type 3 matrix. As mentioned above, eigenvalues of this type of matrix form two clouds. We examine a 99 by 99 matrix of this type with $a = 100.0$ and $b = 44.0$ (see Section 4.3.1). The two clouds reside on intervals [43.928732599741, 43.999929526104] and [100.000000000000, 100.071267400259] respectively. Each cloud of size m may be considered approximately as a root of multiplicity m when viewed from distance. Therefore the quasi-Laguerre's iteration with multiplicity index 1 would converge linearly to the nearest eigenvalue of the cloud.

The following numerical result is extracted from the result of Quasi-Laguerre algorithm applied to type 3 matrix of size 99x99. Multiplicity index 1 is used through out the entire computation. The slow linear convergence is quite obvious.

```
        Numerical Result of Quasi-Laguerre with Mul-index 1
 starting points
 p0=   101.015872629335 p=   100.841454915614
 new points              marching distance       marching ratio
 p2 = 100.743437006649    -9.8017908965762D-02    0.56197221529112
 p3 = 100.644852081645    -9.8584925003505D-02    1.0057848207917
 p4 = 100.563815029678    -8.1037051967655D-02    0.82200247111588
 p5 = 100.492668650879    -7.1146378798231D-02    0.87794875394317
 p6 = 100.431544760492    -6.1123890387780D-02    0.85912862214851
 p7 = 100.378647685865    -5.2897074626041D-02    0.86540752380866
```

```
p8 = 100.332982237172    -4.5665448693327D-02    0.86328873602484
p9 = 100.293536689733    -3.9445547439459D-02    0.86379415002273
p10= 100.259485766377    -3.4050923355454D-02    0.86323870666811
p11= 100.230106703430    -2.9379062947075D-02    0.86279783488954
p12= 100.204781231369    -2.5325472061190D-02    0.86202450046865
p13= 100.182976233372    -2.1804997997236D-02    0.86099078210872
p14= 100.164233166158    -1.8743067214062D-02    0.85957665377625
p15= 100.148157185727    -1.6075980431046D-02    0.85770275736860
p16= 100.134408159176    -1.3749026550949D-02    0.85525275487378
p17= 100.122692599785    -1.1715559390238D-02    0.85210100852045
p18= 100.112756519967    -9.9360798187423D-03    0.84810972210352
p19= 100.104379002023    -8.3775179436346D-03    0.84314116799185
p20= 100.097366441507    -7.0125605164542D-03    0.83706899389962
p21= 100.091547492042    -5.8189494646869D-03    0.82978955419113
p22= 100.086768828702    -4.7786633399168D-03    0.82122440982119
p23= 100.082891873555    -3.8769551465861D-03    0.81130535273354
p24= 100.079790564254    -3.1013093014782D-03    0.79993427424847
p25= 100.077350095807    -2.4404684469887D-03    0.78691552816919
p26= 100.075466390876    -1.8837049306722D-03    0.77186203042146
p27= 100.074045946273    -1.4204446032456D-03    0.75406958919980
p28= 100.073005718055    -1.0402282177712D-03    0.73232579108987
p29= 100.072272825651    -7.3289240457534D-04    0.70454962868210
p30= 100.071784008039    -4.8881761131248D-04    0.66697049698000
p31= 100.071484884863    -2.9912317652253D-04    0.61193207773218
p32= 100.071328989422    -1.5589544052832D-04    0.52117472922255
p33= 100.071273971357    -5.5018065412373D-05    0.35291644980713
p34= 100.071267449120    -6.5222364951387D-06    0.11854717984453
p35= 100.071267400260    -4.8860813617266D-08    7.4914201062295D-03
p36= 100.071267400259    -3.9956651921029D-13    8.1776476818449D-06
```

However, the super-linear convergence can be obtained in these cases if a correct

multiplicity index is used. Therefore, estimating the multiplicity (i.e. the cloud or cluster size) is essential to the cluster and cloud handler.

In our new algorithm, we used the following formula (see (3.9) in Chapter 3) to estimate the cluster size or cloud size of each root as the iteration evolves,

$$mul = int\left(\frac{q_{m\pm}^{(k+1)} q_{m\pm}^{(k)} \left[x_{m\pm}^{(k)} - x_{m\pm}^{(k+1)}\right]}{q(x_{m\pm}^{(k)}) - q(x_{m\pm}^{(k+1)})}\right). \tag{4.13}$$

Overestimation may occur if $x_i^{(m)}$ is not close to λ_i. There are two features concerning over-estimating of *mul*, one being favorable to the convergence if it does not result in jumping because the bigger the *mul* used in the formula the farther the marching distance of the iteration. The other feature will hurt the convergence if it jumps over the eigenvalue. In the second case, the iteration has to back up, namely, reduce the value of *mul* and recompute the next iteration point. We used a simple, yet efficient, back-up scheme by reducing *mul* to be the number of eigenvalues jumped which can be easily detected as a result of Sturm's sequence. Considering the fact that the jumping resulted from overestimating may produce one initial point for approximating the next eigenvalue, so an overestimate of the number *mul* causes essentially no harm because our algorithm can dynamically reduce *mul*(see Figure 4.4), while an underestimate may result in slow convergence.

The following are the numerical results of the new quasi-Laguerre's method with cloud and cluster handler applied to Examples 1 and 2 above. A substantial speedup in each case is observed. *mul* is the multiplicity index used in the quasi-Laguerre's function, *mlt* is the estimated cloud size or cluster size and *jump* is the number of eigenvalues jumped by the current iteration.

Example 1′

```
        Numerical Results of Quasi-Laguerre + Cluster Handler
starting points
p0 =  11.250000000000 p1 =    11.137888560412
new points            marching distance     marching ratio        mul
p2 =  11.057728240655  -8.0160319756418D-02  0.71500571262537      1
```

```
p3 =  11.023732381883  -3.3995858772538D-02  0.42409834286890      1
p4 =  11.009560676203  -1.4171705679063D-02  0.41686564748620      2
p5 =  11.000004846655  -9.5558295481237D-03  0.67428930324464      2
p6 =  11.000000000001  -4.8466546715077D-06  5.0719350393388D-04   2
p7 =  11.000000000000  -7.0335823226288D-13  1.4512241534305D-07   2
```

Example 2′

```
     Numerical Results of Quasi-Laguerre + Cloud Handler
 starting points
 p0 =   101.015872629335 p1  =   100.841454915614
 new points         marching distance     marching ratio     mul mlt jump
100.743437006649  -9.8017908965762D-02  0.56197221529112     1  51   0
100.644852081645  -9.8584925003505D-02  1.0057848207917     51  51   0
100.029961843314  -0.61489023833137     6.2371629162320     51  51  -27
100.194192778959  -0.45065930268662     4.5712800681300     27  27   0
100.097946953508  -9.6245825450974D-02  0.21356671187569    27  27   0
100.065547713253  -3.2399240254871D-02  0.33663008346658    27  27  -9
100.083910561793  -1.4036391714786D-02  0.14583896651118     9   9   0
100.073214269531  -1.0696292261722D-02  0.76204002275420     9   9   0
100.070280416514  -2.9338530174209D-03  0.27428691602977     9   9  -3
100.072038260612  -1.1760089194859D-03  0.10994547369413     3   3   0
100.071242608246  -7.9565236563468D-04  0.67657000933504     3   3  -1
100.071686253674  -3.5200693760571D-04  0.29932335696874     1   1   0
100.071422241996  -2.6401167823509D-04  0.75001839461135     1   1   0
100.071303605790  -1.1863620589650D-04  0.44935969003182     1   1   0
100.071269600688  -3.4005101598554D-05  0.28663342140443     1   1   0
100.071267404003  -2.1966854942081D-06  6.4598704045676D-02  1   1   0
100.071267400259  -3.7435236860743D-09  1.7041691657475D-03  done
```

Our cloud and cluster handler is more dynamic and more effective than the linear acceleration method used in [19] and [8]. A numerical result is reported in [8] that

showed more than 11 iterations are required to compute the 23^{rd} eigenvalue of the 99 by 99 Wilkinson matrix. The starting points they used are closer to the target than the starting points 11.250000000000 and 11.137888560412 used in our test. Our algorithm needs 7 iterations only.

The following numerical result from the linear acceleration in [8] again shows the advantage of our cluster and cloud handler.

Example 2″

Numerical Results of Quasi-Laguerre + Linear Acceleration

starting points

p0 = 102.000000000000 p = 101.961967587691

new points	marching distance	marching ratio	activity
101.419535791275	-0.19431401692363	1.7915217080245	EVFP
101.174041555357	-0.43980825284116		Sturm
101.174041555357	-0.10914874934945	0.55818469601614	EVFP
100.926995451781	-0.24704610357598		Sturm
100.926995451781	-9.8691686273566D-02	0.55818469601614	EVFP
100.703617752133	-0.22337769964884		Sturm
100.703617752133	-6.7609196631664D-02	0.55818469601614	EVFP
100.550591824733	-0.15302592740005		Sturm
100.550591824733	-5.4640879681183D-02	0.55818469601614	EVFP
100.426918246229	-0.12367357850324		Sturm
100.426918246229	-4.0113011667144D-02	0.55818469601614	EVFP
100.336126887701	-9.0791358527966D-02		Sturm
100.336126887701	-3.1160591475516D-02	0.55818469601614	EVFP
100.265598340604	-7.0528547097709D-02		Sturm
100.265598340604	-2.3378215187691D-02	0.55818469601614	EVFP
100.212684340167	-5.2914000436132D-02		Sturm
100.212684340167	-1.7833578400225D-02	0.55818469601614	EVFP
100.172320010912	-4.0364329255723D-02		Sturm

```
100.172320010912   -1.3373261607786D-02 0.55818469601614     EVFP
100.142051114639   -3.0268896272261D-02                      Sturm
100.142051114639   -1.0001205562557D-02 0.55818469601614     EVFP
100.119414493698   -2.2636620941768D-02                      Sturm
100.119414493698   -7.3347879597213D-03 0.55818469601614     EVFP
100.102813013637   -1.6601480060970D-02                      Sturm
100.102813013637   -5.2628633859943D-03 0.55818469601614     EVFP
100.090901105338   -1.1911908298643D-02                      Sturm
100.090901105338   -3.6378825546402D-03 0.55818469601614     EVFP
100.082667161150   -8.2339441885324D-03                      Sturm
100.082667161150   -2.3913400151639D-03 0.55818469601614     EVFP
100.077254627918   -5.4125332318762D-03                      Sturm
100.077254627918   -1.4615792679392D-03 0.55818469601614     EVFP
100.073946505146   -3.3081227715712D-03                      Sturm
100.073946505146   -7.9939460451005D-04 0.55818469601614     EVFP
100.072137164009   -1.8093411371325D-03                      Sturm
100.072137164009   -3.5153908158226D-04 0.55818469601614     EVFP
100.071341494238   -7.9566977062484D-04                      Sturm
100.071341494238   -5.6772584691737D-05 0.55818469601614     EVFP
100.071212995782   -1.2849845666096D-04                      Sturm
100.071214206718   -1.2728752032842D-04                      Sturm
100.071225469893   -1.1602434544500D-04                      Sturm
100.071253032066   -8.8462172612935D-05                      Sturm
100.071284721654   -5.6772584684950D-05                      Sturm
100.071267733919   -1.6987735058407D-05 3.3419749301337      EVFP
100.071267400306   -3.3361304770473D-07 50.920475608743      EVFP
100.071267400259   -4.6343259817599D-11 7198.7393424155      EVFP
100.071267400259   -4.6343259817599D-11 1.3891321138799D-04  done
```

Note that even though Example 2′ and Example 2″ have different starting points, the first clearly showed faster convergence than the second one. The second took 40

iterations after $p = 100.703617752133$, while the first took only 17 iterations after $p = 100.841454915614$ that is farther from the root $\lambda = 100.071267400259$ than 100.703617752133 is.

4.1.5 Partial spectrum

In some applications, only a partial spectrum may be needed. Our algorithm, like bisection method, inherits the features of the split-merge Laguerre's method [19] for finding partial spectrum specified by orders or by intervals.

From the strong interlacing property given in Theorem 4.1.1, one can easily obtain the following:

Proposition 4.1.4 *If $[a,b]$ contains k eigenvalues of $\hat{T}$, then $[a,b]$ contains at least $k-1$ and at most $k+1$ eigenvalues of T. More precisely, let $\kappa(x)$ be the number of eigenvalues of T which are less than $x \in \mathbf{R}$ and $\hat{\lambda}_{s+1}, \cdots, \hat{\lambda}_{s+k}$ be all the eigenvalues of $\hat{T}$ in $[a,b]$. Then $s-1 \leq \kappa(a) \leq s$ and $s+k-1 \leq \kappa(b) \leq s+k$.*

To find eigenvalues of T in a given interval $[a,b]$, the eigenvalues of $\hat{T}$ in [a,b] are found first, say $\hat{\lambda}_{s+1}, \cdots, \hat{\lambda}_{s+k}$. By evaluating $\kappa(a)$ and $\kappa(b)$, the actual number of eigenvalues of T in $[a,b]$ is $\sigma = \kappa(b) - \kappa(a)$. Hence $\lambda_{\kappa(a)+1}, \cdots, \lambda_{\kappa(a)+\sigma}$ are the eigenvalues of T in $[a,b]$. By Proposition 4.1.4, $s-1 \leq \kappa(a) \leq s$ and $s+k-1 \leq \kappa(b) \leq s+k$, so, σ can either be $k-1$, k, or $k+1$. Thus, at most $k+2$ values are needed to be considered as the first starting points to evaluate these σ eigenvalues of T. Let

$$\hat{\hat{\lambda}}_s = a, \quad \hat{\hat{\lambda}}_{s+1} = \hat{\lambda}_{s+1}, \quad \hat{\hat{\lambda}}_{s+2} = \hat{\lambda}_{s+2}, \quad \cdots, \quad \hat{\hat{\lambda}}_{s+k} = \hat{\lambda}_{s+k}, \quad \hat{\hat{\lambda}}_{s+k+1} = b.$$

Then σ values among them can serve as the first staring points which will lead to all σ eigenvalues of T in $[a,b]$.

To find eigenvalues from the i^{th} to the j^{th} eigenvalues, we use bisection and Sturm sequence to find a, and b such that the interval $[a,b]$ contains all the eigenvalues of interest. Then the above method is used in the split-merge process to find all these eigenvalues.

The capability of finding partial spectrum has direct application in parallel computing. We will use this feature in the parallel implementation of the algorithm (see Chapter 5).

4.1.6 Stopping criteria

The following stopping criterion was suggested by Kahan [16]:

$$|x^{(k+1)} - x^{(k)}|^2 \leq (|x^{(k)} - x^{(k-1)}| - |x^{(k+1)} - x^{(k)}|)\tau \tag{4.14}$$

where τ is the error tolerance, see Section 3.2. This criteria is used in [8].

In addition, other stopping criteria are used in our code. We use the trivial stopping criteria, $|x_{k+1} - x_k| \leq \tau$, or $\left|\frac{f(p)}{f'(p)}\right| < \tau$. Note that $\left|\frac{f(p)}{f'(p)}\right|$ is the Newton iteration step size. We also estimate the magnitude of the distance the next quasi-Laguerre's iteration can march. (3.15) derived in Section 3.2 is used for this estimate.

4.2 Description of the new algorithm

4.2.1 The global Newton's formula

Assume that there is no root of $f(x)$ lying between x_0 and x_1, and the logarithmic derivative of f_1'/f_1 at x_1 is known, then the following formula can be used to obtain a second point that does not cross any root of f,

$$x_2 = x_1 - \frac{1}{\frac{f_1'}{f_1} + \frac{n-1}{x_0 - x_1}}. \tag{4.15}$$

This formula is called the global Newton's iteration formula [9]. In [21] this formula is generalized to treat multiple roots and more properties of this method is described.

4.2.2 Initial points for the quasi-Laguerre iteration

At every stage of the merging process, we evaluate the eigenvalues $\lambda_1 < \cdots < \lambda_m$ of an $m \times m$ sub-matrix, given m initial values $\hat{\lambda}_1 \leq \cdots \leq \hat{\lambda}_m$(obtained from the previous

merging process) and an upper bound $\hat{\lambda}_{m+1}$ that interlace those m eigenvalues:

$$\hat{\lambda}_1 \leq \lambda_1 \leq \hat{\lambda}_2 \leq \lambda_2 \leq \cdots \leq \hat{\lambda}_m \leq \lambda_m \leq \hat{\lambda}_{m+1}$$

(see Theorem 4.1.1). To evaluate an eigenvalue λ_i by the quasi-Laguerre iteration, two initial points, say $x^{(0)}$ and $x^{(1)}$, are required on the same side of λ_i without any other eigenvalues lying between them and $x^{(1)}$ is chosen to be closer to λ_i than $x^{(0)}$. For the i^{th} eigenvalue, we start with finding the mid-point $p = \frac{\hat{\lambda}_i + \hat{\lambda}_{i+1}}{2}$ and computing $\frac{f'(p)}{f(p)}$ and $\kappa(p)$. Then use the global Newton's method or Newton's method plus bisection adjustment to determine the next point.

Several improvements are made in our practical implementation. From our computing experience, the high order of convergence of the quasi-Laguerre iteration occurs only when no critical point of f (i.e. zero of f') lies between $x^{(1)}$ and λ_i. In other words, if $x^{(1)}$ is to the left (resp. right) of λ_i, then it is desirable that $-f'(x^{(1)})/f(x^{(1)})$ is positive (resp. negative), see Figure 2.5. If there is one or more critical point in $[\hat{\lambda}_i, \hat{\lambda}_{i+1}]$, then bisection or one step Newton's iteration is used repeatedly until the above requirement at $x^{(1)}$ is satisfied. Also, if the midpoint seems to be too far from the target, then one of $\hat{\lambda}_i$ and $\hat{\lambda}_{i+1}$ might be the eigenvalue and should be tested for quick exit. Our algorithm is summarized in the algorithm INIPTS in Figure 4.3.

4.2.3 Quasi-Laguerre iteration with cluster and cloud handler

After two initial points are found, the quasi-Laguerre's iteration (4.10) is used with $mul = 1$ to begin with. In the process, the algorithm checks whether slow convergence is encountered by checking the ratio $\frac{x_k - x_{k-1}}{x_{k-1} - x_{k-2}}$. If this ratio is between 0.1 and 1.0, (4.13) is used to estimate the multiplicity *mul* of the root. The estimated *mul* value is used in (4.10) to find a new point. However, an overestimated *mul* used in (4.10) may result in a new point that jumps over the target eigenvalue. If this happens, the iteration must back up and the value of *mul* must be reduced to compute a new point. Hence, the previous two points should be saved for possible future backup before using $mul > 1$ in (4.10) to compute a new point. Overshooting is detected by

Algorithm INIPTS

Input: subscript i, initial end points $\hat{\lambda}_i, \hat{\lambda}_{i+1}$. Let $\lambda_0 = \hat{\lambda}_1$.

Local variables: p0ok, p1ok, p1, fp1

Output: starting points $p0, p$ and $\kappa(p0), \kappa(p), fp0 = -\frac{f'(p0)}{f(p0)}, fp = -\frac{f'(p)}{f(p)}$.

Begin INIPTS

$(a, b) = (\hat{\lambda}_i, \hat{\lambda}_{i+1})$;

(##) $p = \frac{a+b}{2}$;

(#1) Evaluate $fp = -\frac{f'(p)}{f(p)}$, $\kappa(p)$ by DETEVL;

If $\kappa(p) = i - 1$, then

if $p0ok$ and $fp > 0$, then go to (#).

else $p0ok = .true., p0 = p, fp0 = fp$;

if $fp > 0$ and $p - \lambda_{i-1} > 2tol$, use G-Newton and goto (#).

else $a = p$, go to (##);

endif

endif

Else if $\kappa(p) = i$, then $b = p$;

if $P1ok$ and $fp < 0$, then p0=p1,fp0=fp1, go to (#).

else $P1ok = .true., p1 = p, fp1 = fp$;

if $fp > 0$ and $p0ok = .false.$, then

evaluate $fa = -\frac{f'(a)}{f(a)}$ and $\kappa(a)$, set p0ok=.true.;

if $\kappa(a) = i$, then $\lambda_i = a$, goto (#) and exit.

else if $fa < 0$ then goto (##);

else then $p = \min\left\{a + fa, \frac{a+p}{2}\right\}$; goto (#1);

endif

endif

else goto (##);

endif

endif

Else

if $\kappa(p) > i$, then $b = p$, goto (##);

else then $a = p$, goto (##);

Endif

(#)End INIPTS

Figure 0.1: **Algorithm** INIPTS

the Sturm sequence obtained in the evaluation of $-\frac{f'(x_k)}{f(x_k)}$. By comparing $\kappa(x_k)$ and $\kappa(x_{k-1})$, the algorithm detects whether a jump occurs. If jmp eigenvalues are jumped over, the algorithm reduces the mul value to $min(mul-1, jmp)$, the minimum of $mul-1$ and jmp, and recompute a new point. The quasi-Laguerre's method with the above feature is called the quasi-Laguerre's method with cluster and cloud handler. The algorithm is illustrated in Figure 4.4.

4.2.4 Stopping test

Stopping test is done in various places in the algorithm. Figure 4.5 shows when and where to check the stopping criteria.

4.3 Numerical tests

Our algorithm is implemented and tested on SPARC stations and DEC Alpha stations with IEEE floating point standard. The machine precision is $\varepsilon \approx 2.2 \times 10^{-16}$.

4.3.1 Testing matrices

There are 12 types of matrices used for testing our algorithm. In the following description of these matrix types, α_i, $i = 1, \cdots, n$, denote the diagonal entries and β_i, $i = 1, \cdots, n-1$, are the sub-diagonal entries.

Matrices with known eigenvalues

Type 1. Toeplitz matrices $[b, a, b]$. Exact eigenvalues: $\{a + 2b\cos\frac{k\pi}{n+1}\}_{1\le k\le n}$ ([12], Example 7.4, p137).

Type 2. $\alpha_1 = a-b$, $\alpha_i = a$ for $i = 2, \cdots, n-1$, $\alpha_n = a+b$. $\beta_j = b$, $j = 1, \cdots, n-1$. Exact eigenvalues: $\{a + 2b\cos\frac{(2k-1)\pi}{2n}\}_{1\le k\le n}$ ([12], Example 7.6, p138).

Algorithm Q-LAG

Input: $p0$, $fp0$, $\kappa(p0)$, p, $dlt0 = p - p0$, no roots between $p0$ and p.

Local variables: $rat = \frac{dlt1}{dlt0}$--used to test convergence rate,

mul--estimated multiplicity used to speed up the convergence,

$oldmul$--multiplicity used in the last iteration,

$jmp = \kappa(p) - \kappa(p0)$--used to adjust mul and $upmul$,

$upmul$--dynamically adjusted upper bound to control mul.

Uutput: $p1$, $dlt1 = p1 - p$.

Begin Q-LAG

$mul = 1$, $upmul = n - 1$;

(#) Evaluate $fp = -\frac{f'(p)}{f(p)}$, $\kappa(p)$ by DETEVL; STOP-CHECK1;

$jmp = \kappa(p) - \kappa(p0)$;

If $|jmp| \neq 0$, then

back up $p0$, $fp0$, p, fp;

$mul = max(min(|jmp|, oldmul - 1), 1)$, $upmul = mul$;

Endif

$p1 = qlag(n, mul, dlt0, p, \kappa, fp0, fp)$ by formula (2.10); $dlt1 = p1 - p$;

$oldmul = mul$. (for backup)

If $mul \neq 1$, store $p0$, $fp0$, p, fp for possible future backup;

$rat = \frac{dlt1}{dlt0}$;

if $(0.1 < rat < 1.0)$, Then

estimate mul by formula (3.9);

$mul = min(mul, upmul)$;

$mul = max(mul, 1)$;

endif

$dlt0 = dlt1$, $p = p1$, $p0 = p$, $fp0 = fp$; STOP-CHECK2; Goto (#);

End Q-LAG

Figure 4.4: **Algorithm** Q-LAG WITH CLUSTER AND CLOUD HANDLER

Algorithm STOP-CHECK

Inputs: $dlt0$, $p0$, $fp0$, p, fp, tol.

Outputs: EigenFound or ContinueIter.

After Evaluating fp, compute $q = 1/fp$;

Begin STOP-CHECK1

if $(|q| < tol)$, then EigenFound, $eig = p + q$.

else ContinueIter;

endif

End STOP-CHECK1

After Q-Lag iteration, new point $p1$ is obtained;

$dlt = p1 - p$, $fprat = \frac{fp0}{fp}$;

Begin STOP-CHECK2

if $dlt1 < tol$, then EigenFound, $eig = p1$.

else if $fprat^2 * dlt < tol$, then EigenFound, $eig = p1$.

else ContinueIter;

endif

End STOP-CHECK2

Figure 4.5: **Algorithm** STOP

Type 3. $\alpha_i = \begin{cases} a & \text{for odd } i \\ b & \text{for even } i \end{cases}$, $\beta_i = 1$. Exact eigenvalues :

$$\left\{ \frac{a+b \pm \sqrt{(a-b)^2 + 16\cos^2 \frac{k\pi}{n+1}}}{2} \right\}_{1 \le k \le n/2} \quad \text{(add } \{a\} \text{ when } n \text{ is odd)}$$

([12], Example 7.8 and 7.9, p139).

Type 4. $\alpha_i = 0$, $\beta_i = \sqrt{i(n-i)}$. Exact eigenvalues: $\{-n + 2k - 1\}_{1 \le k \le n}$ ([12], Example 7.10, p140).

Type 5. $\alpha_i = -[(2i-1)(n-1) - 2(i-1)^2]$, $\beta_i = i(n-i)$. Exact eigenvalues: $\{-k(k-1)\}_{1 \le k \le n}$ ([12], Example 7.11, p141).

Wilkinson and random matrices

Type 6. Wilkinson matrices W_n^+. $\beta_i = 1$,

$$\alpha_i = \begin{cases} n/2 - i + 1 & \text{for even } n \text{ and } 1 \le i \le n/2 \\ i - n/2 & \text{for even } n \text{ and } n/2 < i \le n \\ (n-1)/2 - i + 1 & \text{for odd } n \text{ and } 1 \le i \le (n+1)/2 \\ i - (n+1)/2 & \text{for odd } n \text{ and } (n+1)/2 < i \le n \end{cases}$$

([30], pp308-309). Most of the eigenvalues are in pairs, consisting of two numerically indistinguishable eigenvalues.

Type 7. Random matrices. α_i's and β_i's are random numbers in $[0, 1]$.

LAPACK testing matrices (generated by the LAPACK test matrix generator [2])

Type 8. Matrices with eigenvalues evenly distributed between its smallest and largest eigenvalues.

Type 9. Matrices with geometrically distributed eigenvalues. Namely, eigenvalues can be written as $\{q^k\}_{1 \le k \le n}$ for some $q \in (0, 1)$.

Type 10. Matrices with an eigenvalue 1 and the remaining eigenvalues in $(-\varepsilon, \varepsilon)$.

Type 11. Matrices with eigenvalues evenly distributed in the interval $(0, 1]$ except one eigenvalue with very small magnitude.

Type 12. Matrices with an eigenvalue 1 and the rest of the eigenvalues are evenly distributed in a small interval $[10^{-12} - \varepsilon, 10^{-12} + \varepsilon]$.

4.3.2 Speed test in evaluating eigenvalues without computing eigenvectors

We compare the performance of the following codes for evaluating eigenvalues of an $n \times n$ matrix:

(1) newQ-LAG: our split-merge algorithm using the quasi-Laguerre iteration with cluster and cloud handler. Storage requirement: $9n$;

(2) oldQ-LAG: The old split-merge algorithm using the quasi-Laguerre iteration by [8]. Storage requirement: $9n$;

(3) B/M: bisection/multi-section subroutine DSTEBZ in LAPACK. Storage requirement: $12n$;

(4) RFQR: root-free QR routine DSTERF in LAPACK, as recommended in LAPACK for evaluating eigenvalues only. Storage requirement: $2n$.

First of all, we compare our speed with the old version of quasi-Laguerre's algorithm [8]. The result is presented in Table 4.6. Matrices of types 8-12 are generated by LAPACK. The order of these matrices generated is limited to 550 by the executable program available. For the first five types of matrices, the new version of quasi-Laguerre's algorithm is faster than the old version. For types 8, 11 and 12, the two version dose not have much difference. For the other cases, the old version is faster.

We then compare our new version of quasi-Laguerre's algorithm with the LAPACK subroutines: DSTEBZ–bisection method and DSTERF–root free QR method. The result of this test is given in Figure 4.7 and Figure 4.8. Matrices of type 10 to type 12 involve tiny eigenvalues or clusters and are used more in stability tests. The speed comparison in this category may not be relevant. In particular, matrices of types 10 and 12 have big dense clusters, and intensive deflations make both Q-LAG methods out score RFQR in speed by a wide margin.

4.3.3 Accuracy test

For those matrices with known eigenvalues (type 1 to type 5), the accuracy of a method can be determined by

$$\text{direct error: } \mathcal{D} = \max_i \frac{|\tilde{\lambda}_i - \lambda_i|}{\|T\|_1}$$

where $\tilde{\lambda}_i$ is the approximation of the exact eigenvalue λ_i of T and $\|\cdot\|_1$ is the l_1 norm.

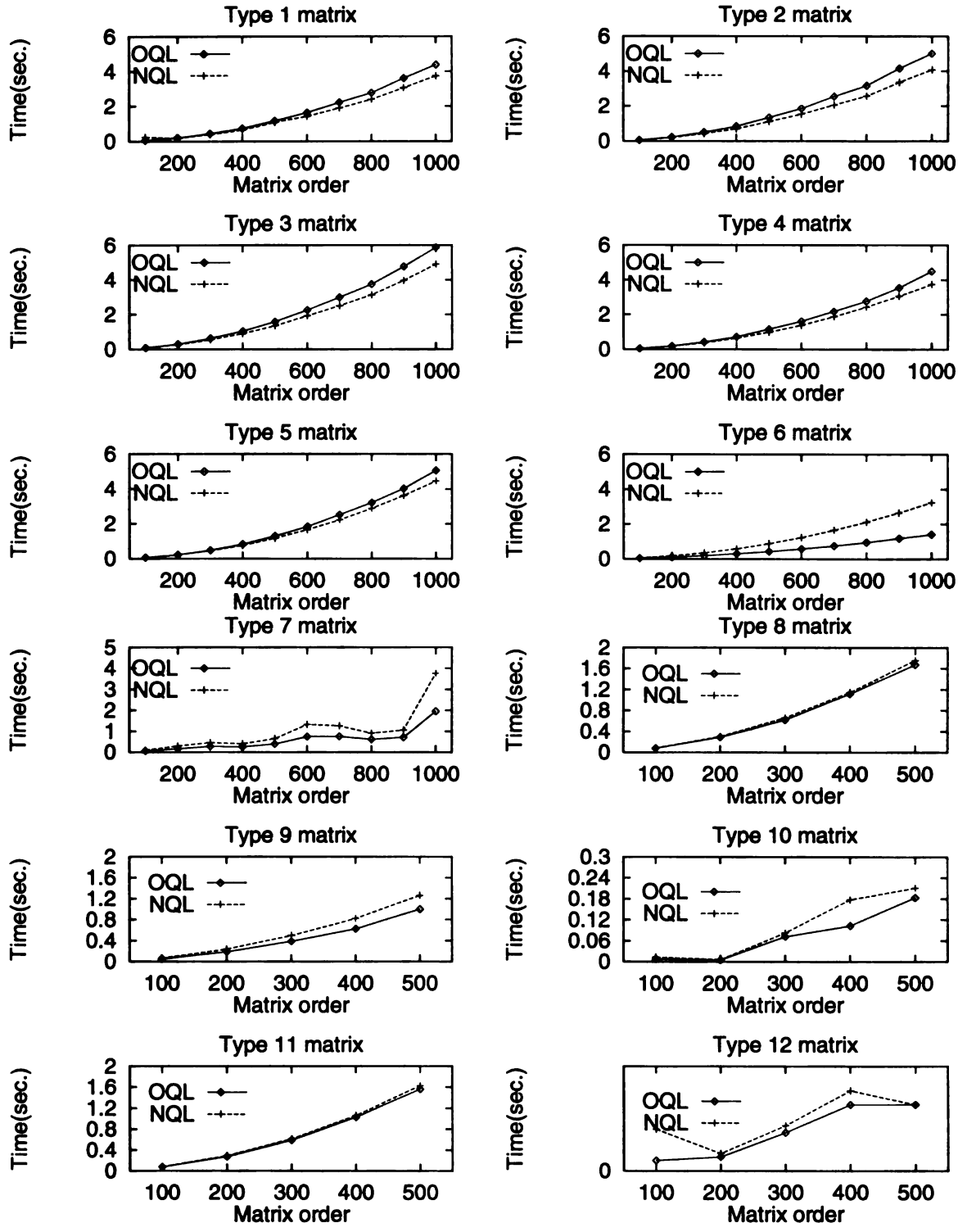

Figure 4.6: Comparison of execution time on Dec Alpha between NQL-the new version of quasi-Laguerre Iteration and OQL-the old version of quasi-Laguerre's method, for finding all eigenvalues without computing eigenvectors.

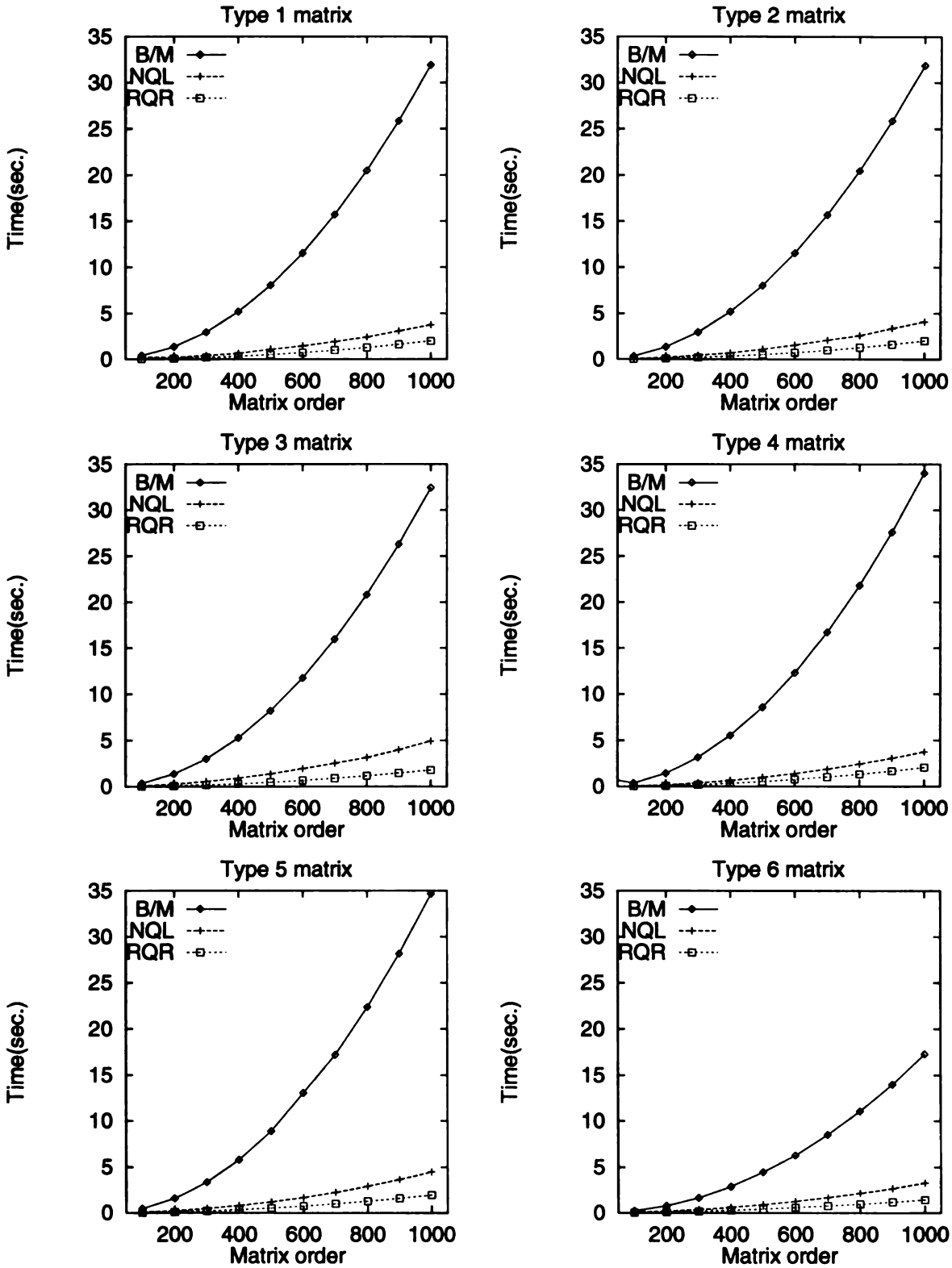

Figure 4.7: Execution time on Dec Alpha for finding all eigenvalues without computing eigenvectors. B/M: DSTEBZ; NQL: the new quasi-Laguerre Iteration; RQR: Root-free-QR–DSTERF.

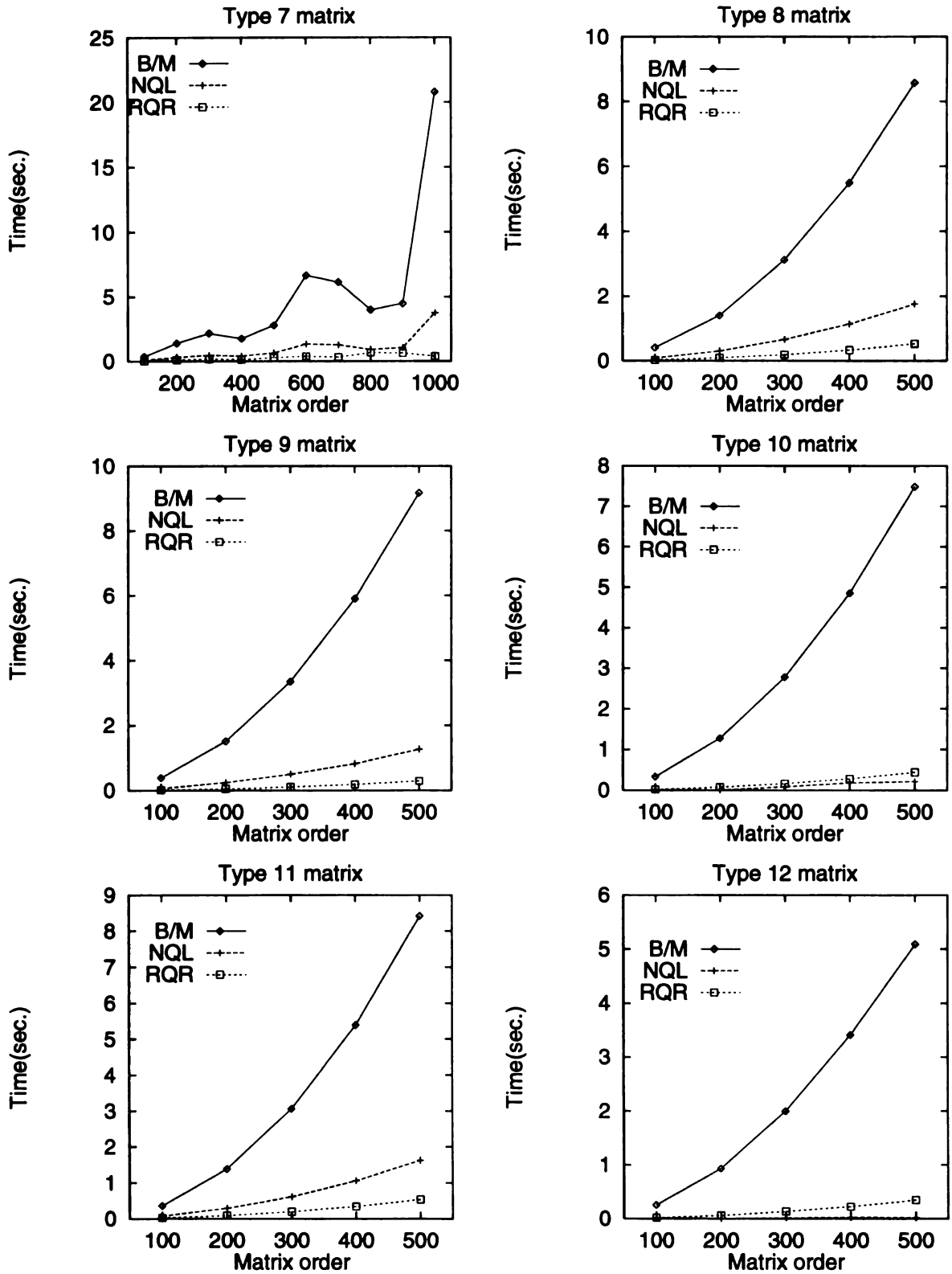

Figure 4.8: Execution time(continued) on Dec Alpha for finding all eigenvalues without computing eigenvectors. B/M: DSTEBZ; NQL: the new quasi-Laguerre Iteration; RQR: Root-free-QR–DSTERF.

First of all, we compare the accuracy of the new Quasi-Laguerre's algorithm with the old version. Results are shown in Table 4.1, which shows the new method is no less accurate than the old one.

We then compare the accuracy of our new algorithm with Bisection/Multi-bisection and Root Free QR method. The results are shown in Figure 4.9. It appears that our algorithm Q-LAG achieves the smallest direct error on all matrices of the first 5 types. The direct error of our algorithm as well as B/M is independent of the matrix size, whereas RFQR seems to have larger error when the matrix size becomes larger. Root free QR is the fastest algorithm, but is the least accurate one, compared to Quasi-Laguerre and Bisection method.

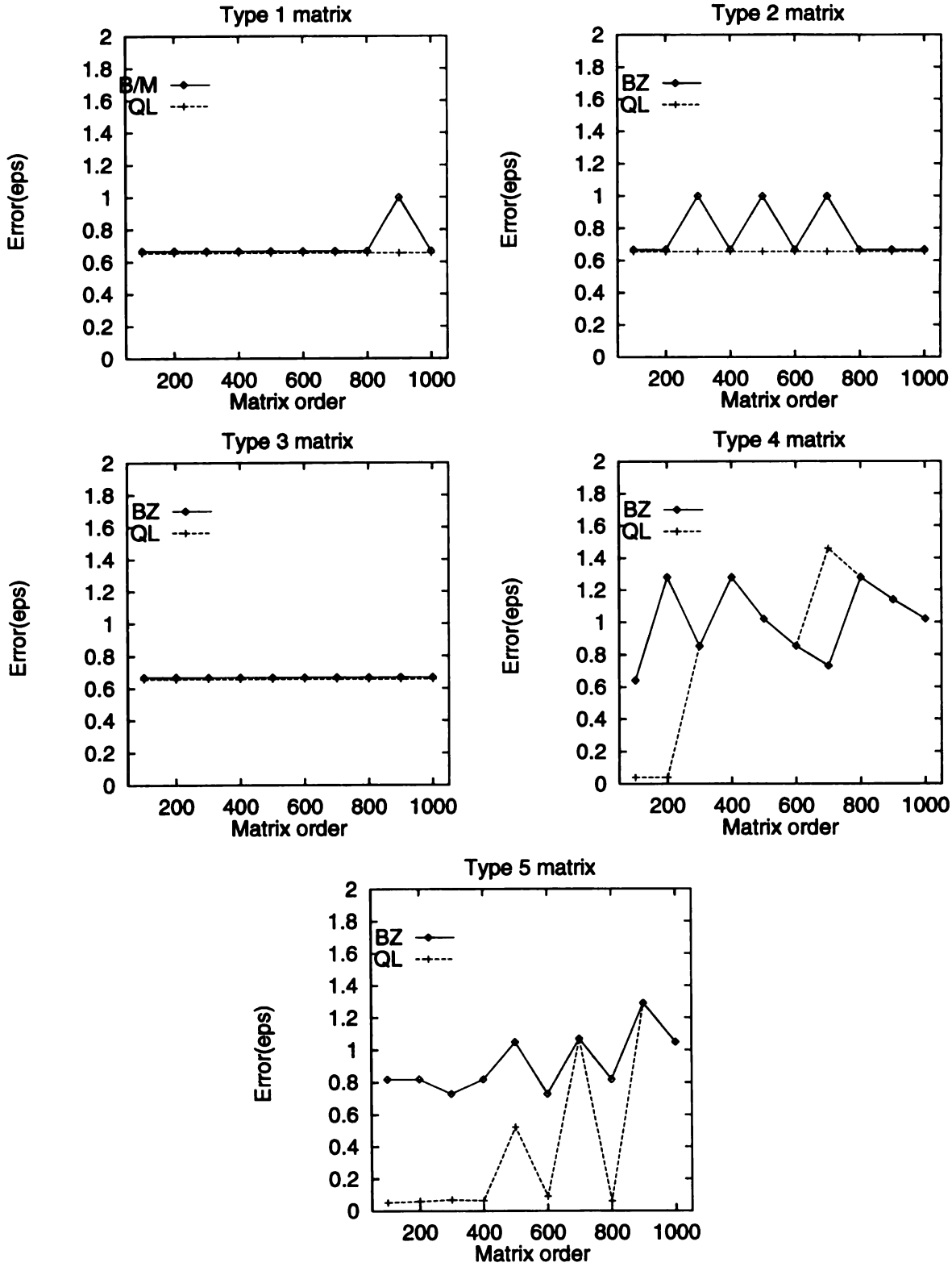

Figure 4.9: Error(on the scale of machine double precision) , on Dec Alpha, for finding all eigenvalues without computing eigenvectors. B/M: DSTEBZ; NQL: the new quasi-Laguerre Iteration; RQR: root free QR-DSTERF.

Order	Version	type 1	type 2	type 3	type 4	type 5
100	new	0.656	0.656	0.656	0.04	0.0512
100	old	0.656	0.656	0.656	0.05	0.102
200	new	0.656	0.656	0.656	0.04	0.0586
200	old	0.656	0.656	0.656	0.04	0.819
300	new	0.656	0.656	0.656	0.853	0.0683
300	old	0.656	0.656	0.656	0.0533	0.728
400	new	0.656	0.656	0.656	1.28	0.0635
400	old	0.656	0.656	0.656	1.28	0.819
500	new	0.656	0.656	0.656	1.02	0.524
500	old	0.656	0.656	0.656	1.02	0.524
600	new	0.656	0.656	0.656	0.853	0.091
600	old	0.656	0.656	0.656	0.853	0.728
700	new	0.656	0.656	0.656	1.46	1.07
700	old	0.656	0.656	0.656	0.731	1.07
800	new	0.656	0.656	0.656	1.28	0.0614
800	old	0.656	0.656	0.656	1.28	1.64
900	new	0.656	0.656	0.656	1.14	1.29
900	old	0.656	0.656	0.656	1.14	1.29
1000	new	0.656	0.656	0.656	1.02	1.05
1000	old	0.656	0.656	0.656	1.02	1.05

Table 4.1: Accuracy comparison between the new version and the old version of the quasi-Laguerre's algorithm. The numbers in the table represents the max-error of $|computedeigs - trueeigs|/(1norm)$, as multiples of machine precision

Chapter 5

Parallel computation of eigenvalues

5.1 Introduction

In the advent of parallel and vector computers, such as Butterfly, Convex, SGI, nCUBE2 etc., and softwares such as PVM [10], it is possible to shorten the time required to finish certain large computing works. Parallel computers often demand parallel algorithm to subdivide the problem into smaller ones so that each CPU(or processor) can handle part of the work simultaneously. As the whole computing is divided into smaller jobs so that all of them can run at the same time, the total amount of computing time can be decreased by a factor of the number of CPUs participated, provided that each job can be executed independently of others. The biggest advantage of our quasi-Laguerre's iteration for solving symmetric tridiagonal eigenproblems is its natural parallelism, in the sense that each eigenvalue can be computed fully independently of the others. In this chapter, we shall present a parallelized version of our algorithm for symmetric tridiagonal eigenvalue problem and present some computation result to show our algorithm is the most efficient and the fastest one among all the existing parallel algorithms for the problem.

There are two types of parallel architectures, one with shared memory and the other with distributed memory. With the parallel software PVM, developed at Oak Ridge National Laboratory, it is possible to hook a bunch of existing UNIX workstations to form a virtually parallel machine. Each workstation has its own memory, so

this virtually parallel machine has distributed memory architecture. Message passing is inevitable for a parallel code to run on this architecture. There are many advantages with this virtually parallel computer:

1. It is less costly since a real parallel computer or vector computer is normally very expensive.

2. It has more memory space since each machine has its own considerably large memory.

3. It is more flexible since the machines can run daily routine jobs when parallel computing is not in demanding.

4. Programs using PVM for a cluster of workstations can be easily ported to some super computers since many super computers support PVM.

5. It is equally suitable for both SIMD and MIMD applications while many other super computer is only better suitable for one of this type of application than the other.

5.2 Issues for parallel algorithm design

To design an efficient parallel algorithm, the following factors are often taken into consideration.

1. **Communication cost**. With a message passing model for distributed memory parallel machine, including PVM machine, sending and receiving data between processors is inevitable. This is called process communication or message passing. Some message passing model requires the sending and receiving happen at the same time, it is called synchronous hand-shaking. With synchronous communication, the computation on the sending processor halts until the matching receive is executed by the receiving processor. A better model allows sending and receiving happen at different time, that is, messages sent are piled up in queue and stored in buffer waiting for picking up by receiver(s) and the sender can continue to process other things

once message is on the way to its destinations. The sending process needs not to wait for the response of the receiver(s). This is called asyncronization. PVM allows asyncrounous communication. The time spent on message passing(including data transferring time, processor waiting time and processor idle time etc.) is called communication cost. An important statistics is the computation to communication ratio: (*time spent computing*)/(*time spent communicating*). One should always maximize this ratio. If the algorithm requires frequent exchange of information among the processors, the communication cost will be very high.

2. **Length of message**. Longer messages take longer time to process, but short messages require a start-up time. For example, if a PVM program wants to send out data, it has to clear the send buffers to prepare them for packing messages. Take a look at the following two pieces of codes, both are sending n integers to n processors (with process IDs specified by tid[ ]).

```
code 1:                          |   code 2:
   ... ...                       |      ... ...
for (i=0; i<n; i++){             |   pvm_initsend(PvmDataRaw);
   pvm_initsend(PvmDataRaw);     |   pvm_pkint(msg, n, 1);
   pvm_pkint(&msg[i], 1, 1);     |   for (i=0; i<n; i++){
   pvm_send(tid[i], tag);        |      pvm_send(tid[i], tag);
}                                |   }
   ... ...                       |       ... ...
```

The second code is obviously more efficient because the first code starts up the sending process n times while the second code starts the initialization only once. The packing in the second code is also more efficient than the one in the first code because it packs the whole array only once.

3. **Message routing**. An effective message routing schedule could also improve the performance of a parallel algorithm. Message routing heavily depends on the architecture of the parallel machine and hardware interconnection. With PVM machines that consist of a bunch of workstations located at different sites with certain

distances and interconnected through Ethernet(typically 30Mbps) or FDDI(typically 100Mbps) fiber cables, message routing is even a more important issue. PVM supports multi-casting that most multiprocessor vendors do not. Multi-casting certainly makes message routing more efficient if one has to send messages to a group of processors. Let's take a look at the following two codes. Both codes are sending, with different ways though, n messages (data) to *n_proc* host machines, specified by the *tid* array.

```
code 3:                              |   code 4:
   ... ...                           |       ... ...
pvm_initsend(PvmDataRaw);            |   pvm_initsend(PvmDataRaw);
pvm_pkint(msg, n_msg, 1);            |   pvm_pkint(msg, n_msg, 1);
pvm_mcast(tid, n_proc, tag);         |   for (i=0; i<n_proc; i++){
   ... ...                           |       pvm_send(tid[i], tag);
                                     |   }
                                     |      ... ...
```

Code 3 is more efficient than code 4 since code 3 sends the messages through the network once while code 2 puts the messages through the network *n_proc* times. Multi-casted data passes the cable (network) only once and is picked up by the receiver when it passes by. Finding out how the PVM machines are wired up could also help reduce the traffic of message passing.

4. **Buffering**. If too many messages are received they must be buffered in some way in some local area. Messages could be lost if we run out of space. With PVM, normally there is no such a problem because each workstation has relatively large memory.

5. **Load balancing**. Section 5.5.1 discuss this issue in more details.

5.3 Determining performance

1. **Timing**. There are different ways to record the run-time of a program. The Fortran subroutine dtime() record the user execution time since the last call to the

subroutine, etime() record the user execution time since the program is started. C function gettimeofday() records the wall time, hence it includes all the computation time, system scheduling time and waiting time etc. In a shared or heterogeneous PVM environment where computer loads, powers or speeds may be different, some processors may be in idle (e.g. waiting for results from other processes) while others are heavily engaged. The performance of an algorithm should be judged by the total time from when the process is first started until when the process is completely finished. This total time includes the machine idle time, waiting time, and certainly computation time. Therefore, using gettimeofday() realistically reflects the performance of an algorithm.

2. **Parallel efficiency**. Parallel efficiency of p processors is defined as

$$E(p) = \frac{T(1)}{pT(p)},$$

where $T(k)$ is the time required to execute the program on k processors. A parallel algorithm with parallel efficiency equal to 1 is a perfectly parallelized algorithm.

5.4 Existed parallel algorithms

There are several parallel algorithms available for computing the eigenvalues of a symmetric tridiagonal matrix. The following is a summary of all the existing algorithms. We will see that our parallel algorithm with quasi-Laguerre's method seems to outperform the others.

1. **Jacobi methods**. This method has been announced to be of historical interest only. Barlow has an introductory description of this algorithm in [3].

2. **QR algorithm**. A parallel QR was first discussed by Sameh and Kuch [28]. Arbenz, Gates, and Sprenger [1] studied a modified version of QR for finding both eigenvalues and eigenvectors. The finding of the eigenvalues was performed redundantly while the calculation of the eigenvectors was done in parallel. In [8], the sequential quasi-Laguerre's method has been compared with the sequential QR for finding both eigenvalues and eigenvectors, and the results showed that quasi-Laguerre is faster than QR on a sequential machine.

3. **Cuppen's Divide-Conquer method.** Ipsen and Jessup [14] reported that parallel bisection is faster than the divide-and-conquer method [4]. Demmel also reported the comparison between these two methods and showed the new version of divide and conquer is only slightly faster than bisection+inverse iteration on a sequential machine for random matrices, but twice as fast as the bisection+inverse for geometrically distributed matrices [5]. Our parallel quasi-Laguerre's method is implemented for distributed memory model, and based on the above information, we do not make the comparison with the method of divide-conquer.

4. **Split-merge Laguerre's method.** A parallel Laguerre's algorithm is reported in [18]. The algorithm parallelized the sequential code along with the split-merge process, that is, after the matrix is split, each processor computes the eigenvalues of the corresponding smaller matrix then computes the eigenvalues of the merged matrix level by level until eigenvalues of the entire matrix are found. With this approach, processors must exchange eigenvalues during each level of merging processes, hence the communication is quite high. [18] reported that when matrix order is small (such as 128), parallel bisection performed better than parallel Laguerre's method. This may attribute to the communication cost of the parallel Laguerre's algorithm. Our parallel approach with the quasi-Laguerre's method requires minimum (nearly zero) communication cost as the parallel bisection does, therefore is always faster than bisection method. Another aspect of the approach in [18] is that the original code was written for nCube supercomputer which requires the number of processors to be power of 2, such as 2, 4, 8, etc.. The code was then ported to PVM environment without implementation modification. Hence the code still requires the number of workstations to be power of 2.

5. **Bisection method.** This is a fully parallelable and fully scalable algorithm. Our algorithm maintains all the good features of this method and our parallel quasi-Laguerre's algorithm outscored parallel bisection method by a great margin.

5.5 The parallel quasi-Laguerre's method

We implemented the parallel quasi-Laguerre's algorithm using PVM with a master and slave program.

The master program divides the spectrum into small chunks, $1 : n\_1$, $(n\_1 + 1) : n\_2$, $(n\_k + 1) : n$, where $i : j$ denotes the eigenvalues from number i to number j. Then the master spawns the slave process to all available machines that form the PVM machine, and sends out information to the slaves. The information sent to the slaves includes the matrix order, the diagonal and off diagonal of the symmetric tridiagonal matrix, starting numbers and ending numbers of eigenvalue chunks, and some other administrative information such as parent process ID and slave process IDs and so on. The master program also serves as an administrator that is freed from computation and also keeps the ability to process other jobs that has to be done sequentially. So we don't spawn slave process onto the machine the master process is running.

The slave program spawned by the master program receives data and 'instructions' sent by the master program and calls split-merge Quasi-Laguerre's subroutine to find the respective eigenvalues of the matrix, then sends the results back to the master program, and wait for another chunk of eigenvalues to compute until an exit instruction is received.

5.5.1 Load balancing

With PVM machine that is composed of a cluster of general purpose, shared workstations interconnected by Ethernet cable or FDDI cable, load balancing is always needed due to the following factors:

1. each workstation may have different computing power and speed,

2. each workstation may have different work loads from other users,

3. initially divided subtasks may require different amount of time to finish.

During the early stage of developing our code, we used uniform(or nearly uniform) subdivision method to divide the whole job into smaller ones. We did not encounter much uneven distribution of computation time among the slave processes until the code is tested in a shared environment and a heterogeneous environment. We found some processors took twice as much time as the others to finish their share in such environment.

Here are two experimental results that exhibit the uneven computation time among the processors due to the shared environment and heterogeneous environment. The first experiment was done on six DEC Alpha workstations(one master and five slaves) in a shared environment, that is, all the processes that are computing the eigenvalues of a matrix have to share CPU with other CPU-intensive processes (mostly from other users). The second experiment was done in a heterogeneous environment, six DEC Alpha-s and two SUN Sparc10s(one Alpha machine served as the master and all other 7 machines served as slaves). The CPU clock speeds for the Dec Alpha workstations (model 3000/400) and Sparc10 workstations used for the experiment are 133MHz and 33MHz, respectively.

In a shared environment, it is somewhat difficult to reproduce an experiment since other user's processes come and go randomly. For comparison purpose, we need to be able to control the environment in order to make the comparison more meaningful. We don't intend to compare apple with cat. So, both experiments were conducted during a reserved time period, that is, all the computers are reserved for the experiment and no other user can get onto the system. In the first experiment, a shared environment is created by creating two CPU-intensive jobs(called dummy processes) on one of the slave machines, then run the parallel quasi-Laguerre program on the PVM machine. One of the slaves has to share CPU with the other two dummy processes that are running on the same host, hence it gets only 1/3 of the CPU access. Both experiments compute all the eigenvalues of a 5000 by 5000 type 4 matrix. The time of each slave, without load balancing, is plotted in Figure 5.1 and Figure 5.2.

Now we have identified the importance of load balancing. The following ideas are implemented in designing the load balancing scheme for our parallel quasi-Laguerre

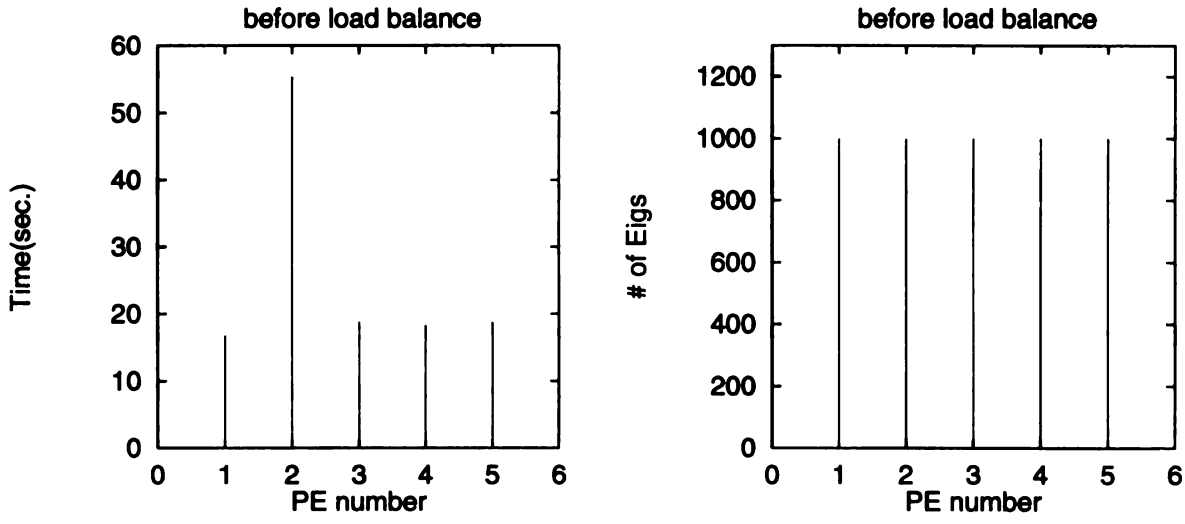

Figure 5.1: Before load balancing in shared environment, other CPU intensive jobs are running on PE # 2 also. Left: Execution time of each Alpha workstation. Right: Number of eigenvalues computed by each Alpha workstation. Matrix size 5000, type 4

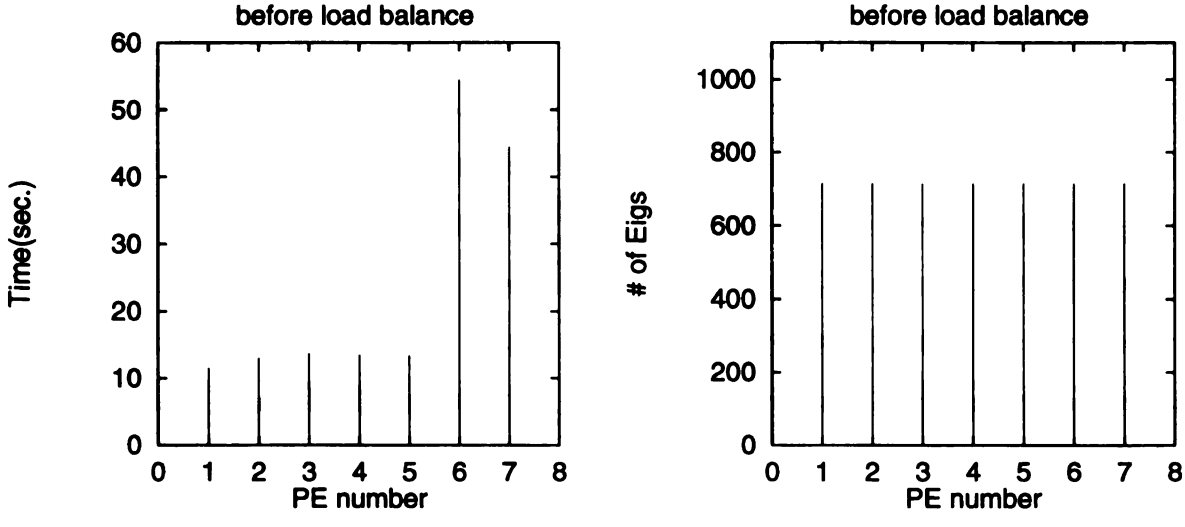

Figure 5.2: Before load balancing in Heterogeneous environment, PE ## 1-5 are Alpha workstations, PE ## 6-7 are SUN Sparc10s. Left: Execution time of each workstation. Right: Number of eigenvalues computed by each workstation. Matrix size 5000, type 4

method.

- Create uneven loads so that earlier distributed load has slightly large chunk size than later distributed loads. This method has an effect of balancing the job as a whole. Also the process which gets the last job won't take too long to finish since the last job is the smallest in chunk size.

- Create more subtasks than the number of available host machines. Hopefully, faster processes can finish more jobs to help the whole situation.

- Spawn more process to each host in hope to gain more CPU favor. This strategy will only be used in emergency, such as, the application needs result as soon possible and must sacrifice other application's needs.

- Reset process priority level to a lower level for courtesy of the actual workstation owners. Lower process priority number means less CPU access, hence less intrusion to the actual workstation owners.

Our algorithm incorporated all of the above features, and the user can control the situation by choosing appropriate parameter values to run the program. However, here we only discuss the first two items, create more and uneven loads, since the other two items are more situation dependent. We used two parameters, n_rounds(number of rounds to distribute the subtasks) and $diff\_size$(chunk size difference between successive processes), to determine how many rounds (each round has n_hosts subtasks) of subtasks to create and how much difference in chunk size between the successive chunks. First of all, the total number of eigenvalues is split into nearly equal chunks(with difference of at most one), then use the value n_rounds to further divide the chunks into smaller ones and use the value of $diff\_size$ to create difference among the chunk sizes. In this way, a job queue is established with chunk sizes in descending order. The rest of the program is just to distribute the chunks from this queue to the slave processes until the queue is empty.

Since the whole job is divided into many small chunks to create more and smaller subtasks, each job takes less time to finish and the processors that finish earlier can

get more subtask to process. As a whole, every host contributes and the hosts that have less load(from other users) or faster CPU speed contribute more to the whole problem. Hence, an overall balanced timing distribution is achieved. The experiment results with this load balancing scheme is plotted in Figures 5.3 and 5.4. It should be noted that time(in seconds) spent on computing is balanced among the participating processes while the number of eigenvalues computed by each host is different.

Although items 3 and 4 above are not discussed in detail here, it is worth to mention that two parameters, n_w(number of salve processes on each host machine) and *prio*(process priority number), are used to determine how many slave processes to spawn on each host machine and whether to honor the actual host owner by reducing our process priority in our code.

Experiments showed that more subtasks create more overheads. In a homogeneous environment, uniform subdivision works slightly better than the nonuniform subdivision method. But in heterogeneous environment or shared platform, this load balancing scheme demonstrates a great advantage.

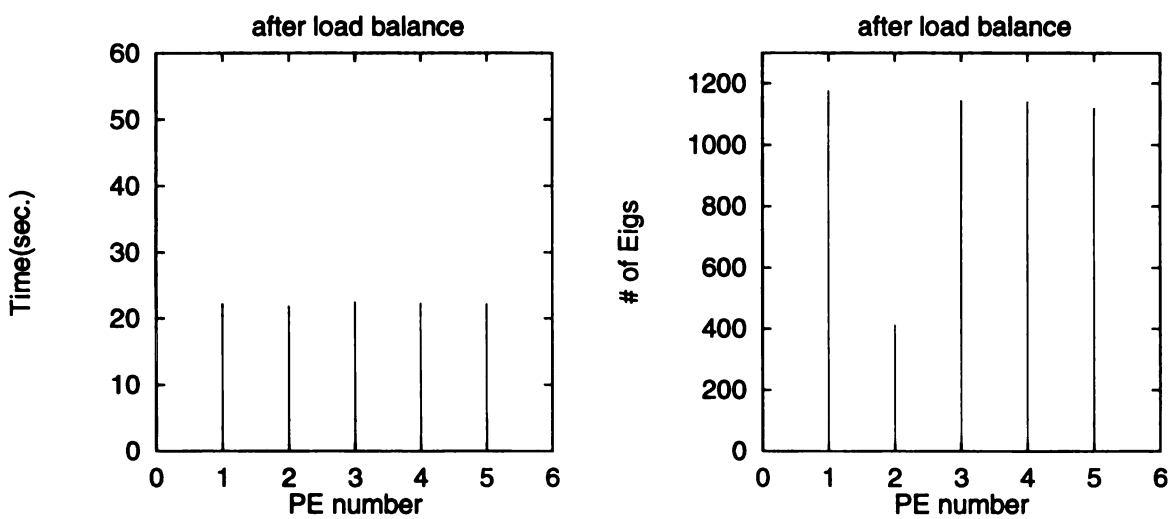

Figure 5.3: After load balancing in shared environment, other jobs are running on PE # 2 also. Left: Execution time of each workstation. Right: Number of eigenvalues computed by each Alpha workstation. Matrix size 5000, type 4

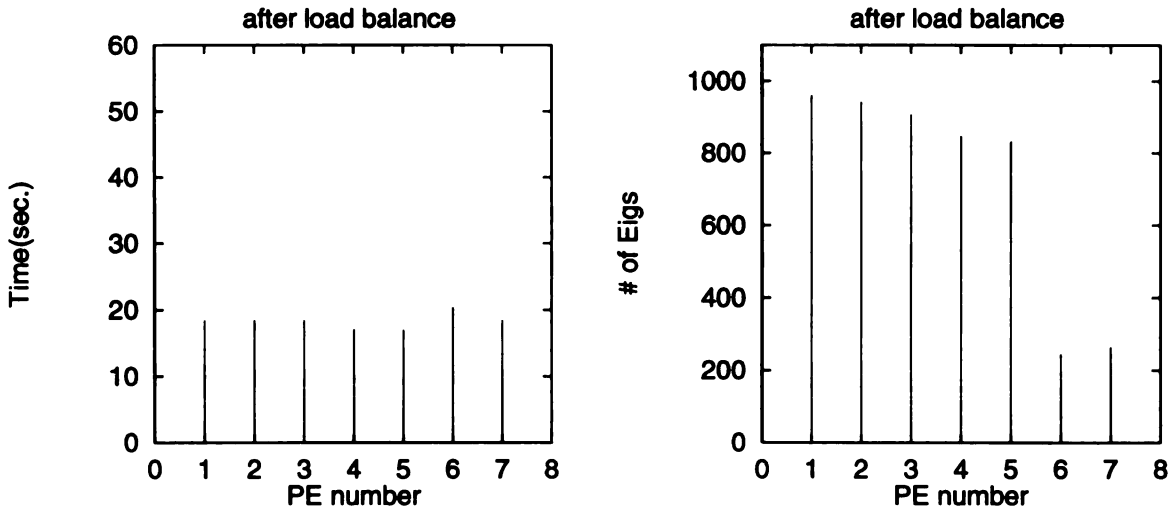

Figure 5.4: After load balancing in Heterogeneous environment, PE ## 1-5 are Alpha workstations, PE ## 6-7 are SUN Sparc10s. Left: Execution time of each workstation. Right: Number of eigenvalues computed by each workstation. Matrix size 5000, type 4

5.5.2 The pseudo-code

The following is the pseudo-code of the master and slave programs. The actual master program is written by C since C handles administration better. The actual slave program is written by Fortran since Fortran is better for scientific computation. On the other hand, the LAPACK routines are written in Fortran.

```
The master program
    ... ...
input diagonal and off-diagonal;
get my process ID -- pid;
detect PVM configuration;
spawn slave process in this config(except me);
get the slave process IDs -- tid[];
create n\_rounds*n\_hosts uneven subtasks, successively differed
       by diff\_size;
multi-cast diagonal and off-diagonal to all slave processes;
```

```
multi-cast slave IDs to all slave processes;
send the first round of subtasks to slaves;
while (received && subtask queue != empty){
  unpack the results;
  send another subtask to the slave process just finished;
}
output all eigenvalues;
    ... ...

The slave program
    ... ...
receive diagonal and sub-diagonal;
receive the first subtask and instructions;
call quasi-Laguerre subroutine to process the first subtask;
send results back to master;
while (received another subtask){
  call quasi-Laguerre subroutine to process the subtask;
}
if (received exit instruction){
  pvm-exit();
}
```

5.6 Performance test

We tested the algorithm on a cluster of DEC ALPHA workstations, a cluster of SUN workstations and the mix of DEC and SUN workstations. We recorded total computation time for each slave and the total time from spawning slave processes until eigenvalues are all received. The result for a type 4 matrix of order 5000 on different number of Alpha workstations is listed in Table 5.1. Husky, Collie, Bulldog, Sheltie and Mongrel are workstations' names. 100, 500, 1000, and so on are the order

of the symmetric tridiagonal matrix whose eigenvalues are computed. The number, in the Collie row and 1000 column for example, is the total contributed time for the workstation Collie to compute its share of the eigenvalues of the corresponding matrix. Max and Min are the maximum and minimum slave time, respectively. The effect of load balancing is reflected by the difference of these two numbers. Large relative difference between these two numbers means unbalanced load. The T5H row lists the whole time, from spawning slave processes till all eigenvalues are received, for 5 slaves to compute all the eigenvalues. The communication to computation ratio, CCR, is calculated by the T5S row and the Max row.

$$CCR = \frac{|T5S - Max|}{Max}.$$

The Collie2 row lists the time to compute all the eigenvalues of the whole matrix by Collie alone. The T1H row listed the whole time from spawning a slave process to Collie until all eigenvalues are received from Collie. The relative PVM overhead, OVHD, is calculated from these two rows,

$$OVHD = \frac{T1H - Collie2}{Collie2}.$$

The PEFY row lists the parallel efficiency of the algorithm. It is calculated by the following formula,

$$PEFY = T1H/5/T5H.$$

Note we did not include the master host in calculating the parallel efficiency. Table 5.1 shows our parallel quasi-Laguerre's algorithm has the following advantages,

1. Communication cost is very low, nearly negligible for large matrices.

2. Parallel efficiency is very high and increases as the order of the matrix increases. This directly reflects the full parallelability and full scalability of our algorithm.

3. The PVM software causes nearly no overhead to the algorithm.

	100	500	1000	5000	10000	20000	40000	50000
Husky	0.04	0.33	0.98	19.28	73.88	286.71	1119.88	1780.84
Collie	0.03	0.29	0.99	19.58	73.88	292.99	1146.79	1818.00
Bulldog	0.02	0.29	0.98	19.22	74.27	288.82	1157.48	1853.14
Sheltie	0.02	0.29	0.99	19.39	74.72	289.76	1166.00	1847.76
Mongrel	0.02	0.29	0.97	19.19	73.76	289.33	1152.39	1829.34
Max	0.04	0.33	0.99	19.58	74.72	292.99	1166.00	1853.14
Min	0.02	0.29	0.97	19.22	73.76	286.71	1119.88	1780.84
T5S	0.79	0.59	1.29	19.9	75.14	293.74	1182.91	1854.66
CCR	1875%	78.8%	26.0%	1.63%	0.56%	0.26%	1.5%	0.1%
Collie2	0.059	0.98	3.74	89.55	358.51	1411.87	5663.02	9014.07
T1H	0.66	1.21	3.98	89.87	358.98	1412.87	5665.25	9016.91
OVHD	1017%	258.4%	6.42%	0.35%	0.13%	0.07%	0.04%	0.03%
PEFY	17%	41%	62%	89%	95%	96%	96%	97%

Table 5.1: Performance test result on DEC ALPHA workstations

5.7 Comparison with parallel bisection and sequential root free QR

We tested type 1 to 7 matrices of order 5000 on six Dec Alphas workstations. Figure 5.5 shows the two configurations PVM machines used for our tests.

We run the parallel program(quasi-Laguerre and bisection) on different number of host machines to compute all the eigenvalues of the seven types of matrices of order 5000. We also run the root free QR program from LAPACK. The total time for each run is recorded and the results are plotted in Figure 5.7. Root free QR method could not take advantage of all available machines. For matrix types 1-5, our parallel quasi-Laguerre's algorithm beats root free QR with three or more machines. For type 6 matrix, our parallel algorithm leads root free QR when there are four or

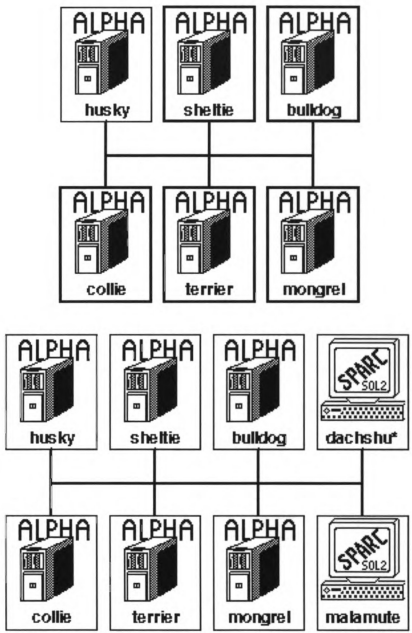

Figure 5.5: Two configuration of the PVM machines. Top: formed by Dec Alpha workstations. Bottom: formed by Dec Alpha and Sun Sparc10 workstations

more machines. For type 7 matrix, quasi-Laguerre wins root free QR with 5 or more machines. In all cases, quasi-Laguerre outperforms bisection method.

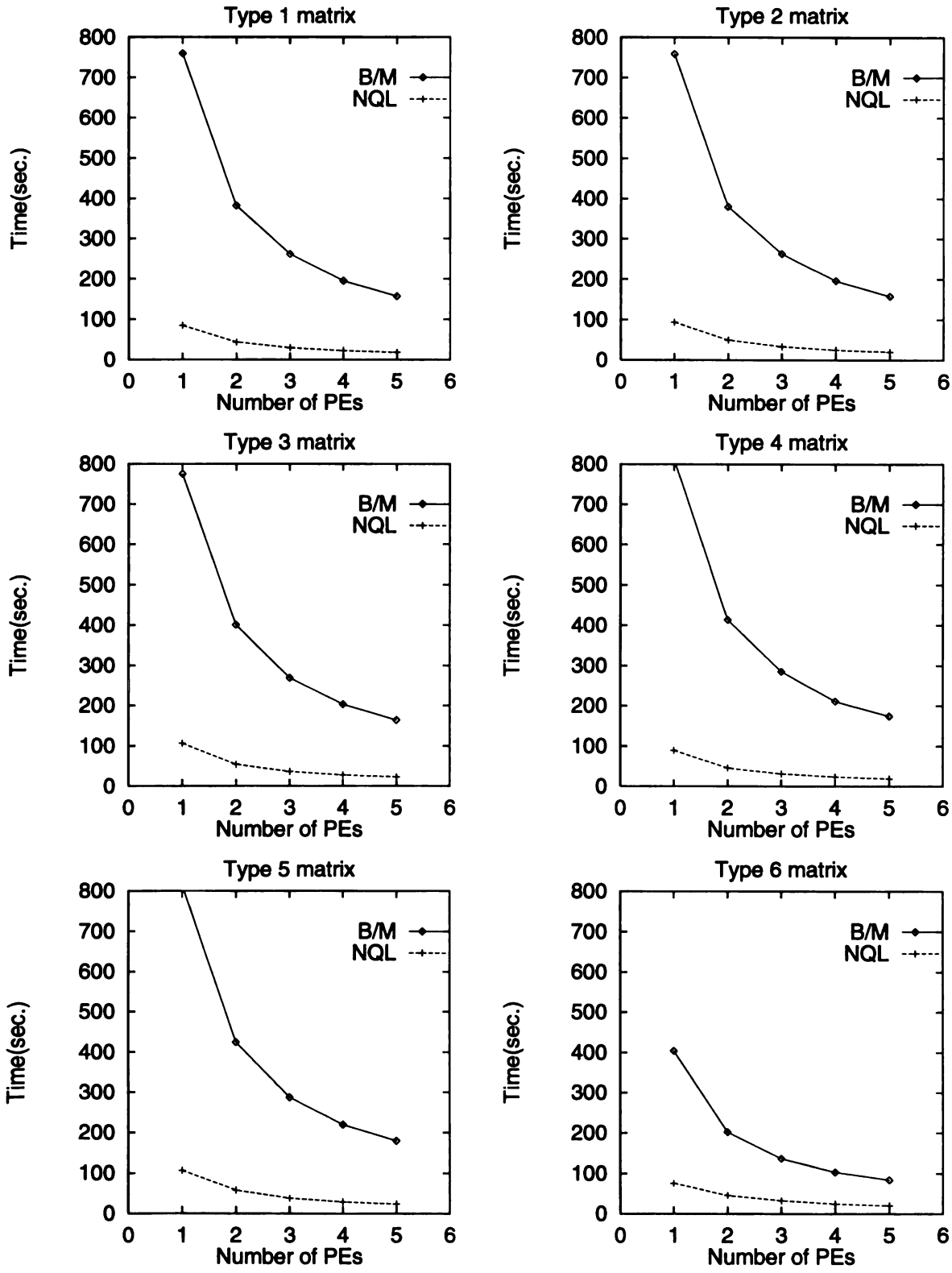

Figure 5.6: Comparison between bisection - B/M, quasi-Laguerre - NQL, and root free QR - RFQR

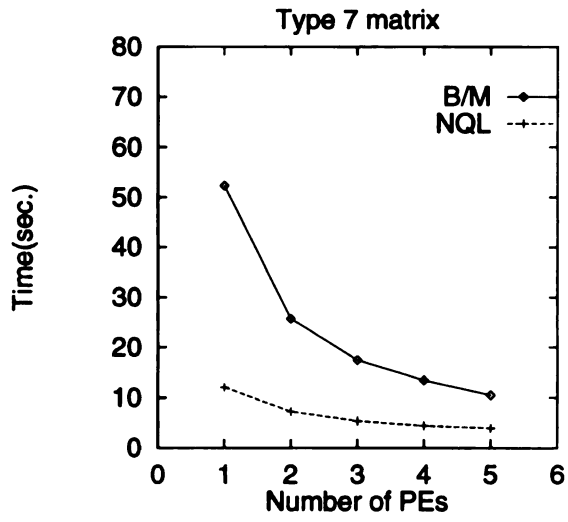

Figure 5.7: Comparison(continued) between bisection - B/M, quasi-Laguerre - NQL, and root free QR - RFQR, on a random matrix of order 5000

Bibliography

[1] P. ARBENZ, K. GATES, AND C. SPRENGER *A Parallel implementation of the symmetric tridiagonal qr algorithm*, Proceedings of the Fourth Symposium on the Frontiers of massively Parallel Computation, IEEE CS Press, 1992.

[2] E. ANDERSON, Z. BAI, C. BISCHOF, J. DEMMEL, J. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, S. OSTROUCHOV, and D. SORENSON, *LAPACK User's Guide*, SIAM, Philadelphia, 1992.

[3] J. L. BARLOW *The Parallel Solution of the Symmetric Eigenvalue Problem* , Large-Scale Matrix Problems and the Numerical Solution of Partial Differential Equations, Advances in Numerical Analysis Vol. III, Oxford, 1994

[4] J. J. M. CUPPEN, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math., 36 (1981), pp. 177–195.

[5] J. DEMMEL, *Designing high performance symmetric eigenvalue software for parallel computers*, http://http:.berkeley.edu/~demmel, Feb 17, 1995.

[6] J. J. DONGARRA AND D. C. SORENSEN, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. 139–154.

[7] Q. DU, M. JIN, T. Y. LI AND Z. ZENG, *Quasi-Laguerre iteration*, preprint, Michingan State University, 1995

[8] Q. DU, M. JIN, T.Y. LI AND Z. ZENG *Quasi-Laguerre iteration in solving symmetric tridiagonal eigenvalue problems*, Preprint, Michigan State University, 1995.

[9] L. V. FOSTER, *Generalizations of Laguerre's method: lower order methods*, preprint.

[10] A. GEIST, A. BEGUELIN, J. DONGARRA, W. JIANG, R. MANCHEK, V. SUNDERAM, *PVM 3 User's Guide and Reference Manual*, September, 1994.

[11] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations, 2nd Ed.*, The Johns Hopkins University Press, Baltimore, MD, 1989.

[12] R. T. GREGORY AND D. L. KARNEY, *A Collection of Matrices for Testing Computational Algorithms*, Robert E. Krieger Publishing Company, Huntington, New York, 1978.

[13] M. GU AND S. C. EISENSTAT, *A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem*, SIAM J. Matrix Anal. Appl., Vol. 16, No. 1 (1995), pp. 172-191.

[14] I. IPSEN & E. JESSUP *Solving the symmetric tridiagonal eigenvalue problem on the hypercube*, SIAM journal of Scientific and Statistical Computing, Vol. 11. pp. 203-229, March 1990.

[15] M. JIN *Quasi-Laguerre's method and application to symmetric tridiagonal eigenvalue problem*, Ph.D thesis, 1995.

[16] W. KAHAN, *Notes On Laguerre's Iteration*, preprint, University of California, Berkeley (1992).

[17] J.L. Lagouanelle. *Sur une méthode de calcul de l'ordre de multiplicité des zéros d'un polynôme.* C. R. Acad. Sci. Paris Sér. A. 262(1966). 626-627.

[18] C. TREFFTZ, C. C. HUANG, P. MCKINLEY, T. Y. LI, AND Z. ZENG, *A scalable eigenvalue solver for symmetric tridiagonal matrices*, to appear, Parallel Comput.

[19] T. Y. Li and Z. Zeng, *Laguerre's iteration in solving the symmetric tridiagonal eigenproblem — revisited*, SIAM J. Sci. Comput., Vol. 15, No. 5 (1994), pp. 1145-1173.

[20] T. Y. Li and Z. Zeng, *Homotopy-determinant algorithm for solving non-symmetric eigenvalue problem* , Mathematics of Computation, Vol. 59, No. 200 (1992), pp. 483-502.

[21] T.Y. Li and X. Zou , *On the global Newton's method and global Secant method*, preprint, Michigan State University, 1995.

[22] D. G. Luenberger *Linear and nonlinear programming*, Reading, Mass. Addison-Wesley, 1984.

[23] B. N. Parlett, *The use of a refined error bound when updating eigenvalues of tridiagonal*, Lin. Alg. & Appls., Vol. 68 (1985), pp. 179-219.

[24] B. N. Parlett, *Orthogonal eigenvectors without Gram-Schmidt*, Dundee Numerical Analysis Conference, 1995.

[25] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[26] M. Petković. *Iterative Methods for Simultaneous Inclusion of Polynomial Zeros. Lecture Notes in Mathematics.* Springer-Verlag 1989.

[27] J. Rutter,*A serial implementation of Cuppen's divide and conquer algorithm for the symmetric eigenvalue problem*, LAPACK lawn 69 (1994).

[28] A. Sameh and D. Kuck, *A parallel QR algorithm for symmetric tridiagonal matrices*, IEEE Transactions on Computers, no. C-26, pp.81-91, 1977.

[29] D. C. Sorensen and P. T. P. Tang, *On the orthogonality of eigenvectors computed by divide-and-conquer techniques*, SIAM. J. Numer. Anal., 28 (1991), pp. 1752 - 1775.

[30] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford, 1965.