QR-like Algorithms for Eigenvalue Problems

David S. Watkins

ABSTRACT. In the year 2000 the dominant method for solving matrix eigenvalue problems is still the QR algorithm. This paper discusses the family of GR algorithms, with emphasis on the QR algorithm. Included are historical remarks, an outline of what GR algorithms are and why they work, and descriptions of the latest, highly parallelizable, versions of the QR algorithm. Now that we know how to parallelize it, the QR algorithm seems likely to retain its dominance for many years to come.

1. Introduction

Since the early 1960's the standard algorithms for calculating the eigenvalues and (optionally) eigenvectors of "small" matrices have been the QR algorithm [**29**] and its variants. This is still the case in the year 2000 and is likely to remain so for many years to come. For us a *small* matrix is one that can be stored in the conventional way in a computer's main memory and whose complete eigenstructure can be calculated in a matter of minutes without exploiting whatever sparsity the matrix may have had. If a matrix is small, we may operate on its entries. In particular, we are willing to perform similarity transformations, which will normally obliterate any sparseness the matrix had to begin with.[1]

If a matrix is not small, we call it *large*. The boundary between small and large matrices is admittedly vague, but there is no question that it has been moving steadily upward since the dawn of the computer era. In the year 2000 the boundary is around $n = 1000$, or perhaps a bit higher.

Eigenvalue problems come in numerous guises. Whatever the form of the problem, the QR algorithm is likely to be useful. For example, for generalized eigenvalue problems $Ax = \lambda Bx$, the method of choice is a variant of the QR algorithm called QZ. Another variant of QR is used to calculate singular value decompositions (SVD) of matrices. The QR algorithm is also important for solving large eigenvalue problems. Most algorithms for computing eigenvalues of large matrices repeatedly generate small auxiliary matrices whose eigensystems need to be computed as a subtask. The most popular algorithms for this subtask are the QR algorithm and its variants.

1991 *Mathematics Subject Classification.* 65F15.

[1]However, we are not averse to seeking to preserve and exploit certain other structures (e.g. symmetry) by choosing our transforming matrices appropriately.

QR Past and Present. In this paper we discuss the family of GR algorithms, which includes the QR algorithm. The subject was born in the early 1950's with H. Rutishauser's quotient-difference algorithm [**44**], [**45**], which he formulated as a method for calculating the poles of a meromorphic function. He then reformulated it in terms of matrix operations and generalized it to the LR algorithm [**46**].[2] The QR algorithm was published by Kublanovskaya [**38**] and Francis [**29**] in 1961. The Francis paper is particularly noteworthy for the refinements it includes. The double-shift implicit QR algorithm laid out there is only a few details removed from codes that are in widespread use today.

And what codes are in use today? By far the most popular tool for matrix computations is Matlab. If you use Matlab to compute your eigenvalues, you will use one of its four QR-based computational kernels. Each of these is just a few refinements removed from codes in the public-domain software packages EISPACK [**2**] and LINPACK [**21**]. In particular, the algorithm for computing eigenvalues of real, nonsymmetric matrices is just the Francis double-shift QR algorithm with some modifications in the shift strategy.

A newer public-domain collection is LAPACK [**26**], which was designed to perform well on vector computers, high-performance work stations, and shared-memory parallel computers. It also has a double-shift implicit QR code, which is used on matrices (or portions of matrices) under 50×50. For larger matrices a multishift QR code is used.

For many years the QR algorithm resisted efforts to parallelize it. The prospects for a massively parallel QR algorithm for distributed memory parallel computers were considered dim. The pessimism was partly dispelled by van de Geijn and Hudson [**48**], who demonstrated the first successful highly parallel QR code. However, their code relies on an unorthodox distribution of the matrix over the processors, which makes it hard to use in conjunction with other codes. Subsequently G. Henry [**34**] wrote a successful parallel QR code that uses a standard data distribution. This is an implicit double-shift code that performs the iterations in pipeline fashion. This code is available in ScaLAPACK [**27**], a collection of matrix computation programs for distributed-memory parallel computers.

On the theoretical side, the first proof of convergence of the LR algorithm (without pivoting or shifts of origin) was provided by Rutishauser [**46**]. His proof was heavily laden with determinants, in the style of the time. Wilkinson [**61**] proved convergence of the unshifted QR algorithm using matrices, not determinants. Wilkinson [**62**], [**41**] also proved global convergence of a shifted QR algorithm on symmetric, tridiagonal matrices. Della Dora [**19**] introduced a family of GR algorithms and proved a general convergence theorem (unshifted case). In [**59**] a more general family of GR algorithms was introduced, and general convergence theorems for shifted GR algorithms were proved.

This Paper's Contents. This paper provides an overview of the family of GR algorithms, with emphasis on the QR case. The properties of the various QR implementations are discussed. We begin by introducing the family of GR algorithms in Section 2. These are iterative methods that move a matrix toward upper-triangular form via similarity transformations. We discuss the convergence

[2]Amazingly the quotient-difference algorithm has had a recent revival. Fernando and Parlett [**28**], [**42**] introduced new versions for finding singular values of bidiagonal matrices and eigenvalues of symmetric, tridiagonal matrices.

of GR algorithms briefly. In Section 3 we show how to implement GR algorithms economically as bulge-chasing procedures on Hessenberg matrices. In Sections 4 and 5 we discuss multishift and pipelined QR algorithms, respectively.

Section 6 discusses the generalized eigenvalue problem $Av = \lambda Bv$ and GZ algorithms, which are generalizations of GR algorithms. Particularly important among the GZ algorithms are the QZ algorithms. These are normally implemented implicitly, as bulge chasing algorithms. However, in Section 7, we discuss a completely different class of explicit QZ algorithms. These attempt to divide and conquer the problem by splitting it apart on each iteration. They are highly parallelizable and may play a significant role in parallel eigensystem computations in the future.

2. GR Algorithms

Let A be an $n \times n$ real or complex matrix whose eigenvalues we seek. GR algorithms [**59**] are iterative methods that begin with a matrix A_0 similar to A (e.g. $A_0 = A$) and produce a sequence (A_m) of similar matrices. All GR algorithms have the following form. Given the iterate A_m, the next iterate A_{m+1} is produced as follows. First a *spectral transformation function* f_m is somehow chosen. At this point the only requirement on f_m is that the matrix $f_m(A_m)$ be well defined. Thus f_m could be a polynomial, rational function, exponential function, or whatever. The next step is to decompose $f_m(A_m)$ into a product

$$f_m(A_m) = G_{m+1}R_{m+1}, \tag{2.1}$$

where G_{m+1} is nonsingular and R_{m+1} is upper triangular. There are lots of ways to do this; the symbol G stands for *general* or *generic*. The final step of the iteration is to use G_{m+1} in a similarity transformation to produce the next iterate:

$$A_{m+1} = G_{m+1}^{-1}A_mG_{m+1}. \tag{2.2}$$

If the f's and G's are chosen well (and perhaps even if they are not), the sequence of similar matrices, all of which have the same eigenvalues, will converge rapidly to a block upper triangular form

$$\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix},$$

thereby splitting the problem into two smaller eigenvalue problems with matrices A_{11} and A_{22}. After $O(n)$ such splittings, the eigenvalue problem has been solved.

Some variants are the RG algorithms, in which the order of factors in (2.1) is reversed, and the GL and LG algorithms, in which lower triangular matrices are used. All of these families have isomorphic convergence theories. In practice some of these variants do come in handy here and there, but we will focus for the most part on the GR case.

A particular GR algorithm is determined by how the spectral transformation functions f_m are chosen and how the transforming matrices G_{m+1} are specified. Let us first discuss choices of G.

If each G_{m+1} is required to be unitary, then the symbol Q is used instead of G, the decomposition becomes $f_m(A_m) = Q_{m+1}R_{m+1}$, and the algorithm is called a QR algorithm. The requirement that Q_{m+1} be unitary implies that the factors in the decomposition are nearly uniquely determined. This is the most popular choice of G. Expositions on the QR algorithm can be found in numerous books [**31**], [**52**], [**61**].

If each G_{m+1} is required to be unit lower triangular, that is, lower triangular with ones on the main diagonal, then the symbol L is used, the decomposition becomes $f_m(A_m) = L_{m+1}R_{m+1}$, and the algorithm is called an LR algorithm. The LR decomposition is unique if it exists, but not every matrix has an LR decomposition. This means that the choice of f_m must be restricted in such a way that $f_m(A_m)$ has an LR decomposition. The algorithm is unstable; difficulties arise when f_m are chosen so that $f_m(A_m)$ is close to a matrix that has no LR decomposition. Stability can be improved markedly by the introduction of pivoting (row and column interchanges). Wilkinson's book [**61**] discusses LR algorithms in detail.

Other examples are the HR [**10**], [**11**] SR [**13**], [**14**], and BR [**30**] algorithms. The H stands for *hyperbolic*, the S for *symplectic*, and the B for *balancing, band-reducing, bulge-chasing* algorithm.

Now let us consider some ways of choosing the functions f_m. We call them spectral transformation functions because it is their job to transform the spectrum of the matrix in order to accelerate convergence. We also refer to f_m as the function that *drives* the mth iteration. The simplest spectral transformation functions are polynomials, and the simplest useful polynomials have degree one. If we take $f(z) = z - \mu$, then we have $f(A) = A - \mu I$. Such a choice gives us a *simple* or *single* GR step with *shift* μ. The quadratic choice $f(z) = (z-\mu)(z-\nu)$ gives a *double* GR step with *shifts* μ and ν. A double step is worth two single steps. The standard QR codes for real matrices (dating back to Francis [**29**]) take double steps with either real μ and ν or complex $\nu = \bar{\mu}$. This keeps the computations real. The multishift QR algorithm [**3**] takes $f(z) = (z-\mu_1)(z-\mu_2)\cdots(z-\mu_p)$, where p can be as big as one pleases in principle. In practice roundoff errors cause problems if p is taken much bigger than six.

A more exotic choice would be a rational function such as

$$f(z) = \frac{(z-\mu)(z-\bar{\mu})}{(z+\mu)(z+\bar{\mu})}.$$

This is the sort of f that is used to drive the Hamiltonian QR algorithm of Byers [**16**], [**17**]. The more general use of rational spectral transformation functions is discussed in [**57**].

An even more exotic choice would be a characteristic function for the unit disk:

$$f(z) = \begin{cases} 1 & \text{if} \quad |z| < 1 \\ 0 & \text{if} \quad |z| > 1 \end{cases} \tag{2.3}$$

This is a simple function to describe, but how does one calculate $f(A)$? For now we just remark that there are good rational approximations. For example, if k is a large integer, the rational function

$$f(z) = \frac{1}{z^k + 1}$$

approximates the characteristic function quite well away from the circle $|z| = 1$.

Factors that affect the convergence rate. The convergence theory of GR algorithms was discussed by Watkins and Elsner [**59**] and summarized in [**55**]. There are two factors affecting the convergence of the algorithm: the choice of f_m and the choice of G_m. Let $\hat{G}_m = G_1G_2\cdots G_m$, the product of the transforming matrices for the first m steps. If the condition numbers $\kappa(\hat{G}_m)$ grow with m, convergence can be degraded or prevented. On the other hand, it is the role of the f_m to promote or accelerate convergence. For starters let's suppose that the same f is used on every iteration. If $\lambda_1, \lambda_2, \ldots, \lambda_n$ are the eigenvalues of A, then $f(\lambda_1), f(\lambda_2), \ldots, f(\lambda_n)$ are the eigenvalues of $f(A)$. Suppose they are numbered so that $|f(\lambda_1)| \geq |f(\lambda_2)| \geq \cdots \geq |f(\lambda_n)|$. Then the ratios

$$\rho_k = |f(\lambda_{k+1})/f(\lambda_k)| \qquad k = 1, \ldots, n-1$$

are what determine the asymptotic convergence rate. These ratios all satisfy $0 \leq \rho_k \leq 1$. The closer to zero they are, the better. The underlying mechanism is subspace iteration [**35, 15, 43, 51, 59**].

Let us consider the effect of the kth ratio ρ_k. Suppose $\rho_k < 1$, and let $\hat{\rho}_k$ be any number satisfying $\rho_k < \hat{\rho}_k < 1$. Partition the iterates A_m into blocks

$$A_m = \begin{bmatrix} A_{11}^{(m)} & A_{12}^{(m)} \\ A_{21}^{(m)} & A_{22}^{(m)} \end{bmatrix},$$

where $A_{11}^{(m)}$ is $k \times k$. Then, under mild assumptions, there exists a constant C such that

$$\| A_{21}^{(m)} \| \leq C\kappa(\hat{G}_m)\hat{\rho}_k^m \qquad \text{for all } m.$$

Thus A_m approaches block upper triangular form if $\kappa(\hat{G}_m)\hat{\rho}_k^m \to 0$.

If there is a bound K such that $\kappa(\hat{G}_m) \leq K$ for all m, then convergence is linear with ratio $\rho_k = |f(\lambda_{k+1})/f(\lambda_k)|$. Even if $\kappa(\hat{G}_m)$ is unbounded, there still can be convergence if the growth is not too fast.

So far we have assumed that f is held fixed. Varying f makes the convergence analysis harder, but (with rare exceptions) it pays off in accelerated convergence. Successful shift strategies are (with rare exceptions) able to choose f_m so that $f_m(A) \to f(A)$, where f is a function such that $\rho_k = 0$ for some k. This yields superlinear convergence. A simple shift strategy that normally yields quadratic convergence is discussed below.

Let us reconsider choices of G in light of the convergence theory. Clearly the objective is to make the transforming matrices as well conditioned as possible. This is true also from the point of view of stability, since the condition numbers $\kappa(\hat{G}_m)$ govern the stability of the algorithm as well. From this viewpoint the QR algorithms are obviously best, as they guarantee $\kappa_2(\hat{Q}_m) = 1$ for all m. No such guarantees exist for any of the other GR algorithms, which explains why the QR algorithms are by far the most popular. In certain special circumstances (e.g. Hamiltonian problems) there exist (non-QR) GR algorithms that are very fast ($O(n)$ work per iteration instead of $O(n^2)$) because they are able to exploit the structure. In those circumstances one may be willing to trade the stability guarantee for speed. But then one must always be alert to the danger of instability. In this paper we will focus mainly on QR algorithms.

We now reconsider choices of f in light of the convergence theory. The simplest and most common choice is the polynomial

$$f(z) = (z - \mu_1)(z - \mu_2) \cdots (z - \mu_p).$$

The best we can do is to take the shifts $\mu_1, \ldots, \mu_p$ to be eigenvalues of A . Then $f(A)$ has p zero eigenvalues, so

$$\frac{f(\lambda_{n-p+1})}{f(\lambda_{n-p})} = 0. \tag{2.4}$$

Such a good ratio implies very rapid convergence. Indeed, after just one iteration we get

$$A_1 = \begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} \\ 0 & A_{22}^{(1)} \end{bmatrix}, \tag{2.5}$$

where $A_{22}^{(1)}$ is $p \times p$ and has $\mu_1, \ldots, \mu_p$ as its eigenvalues.[3]

The catch is that we do not normally have the eigenvalues available to use as shifts. However, after a few iterations we might well have some good approximations, and we can use these as shifts. If all p shifts are excellent approximations to eigenvalues, then the ratio in (2.4) will be close to zero, and convergence to a form like (2.5) will be achieved in a few iterations. Subsequent iterations can be applied to the submatrix A_{11} with a new set of shifts.

Normally new shifts are chosen on each iteration. The most common strategy is to take the shifts (on the mth iteration) to be the eigenvalues of the lower right-hand $p \times p$ submatrix $A_{22}^{(m)}$. In other words, f_m is taken to be the characteristic polynomial of $A_{22}^{(m)}$. Global convergence is not guaranteed, but the local convergence rate is normally quadratic and can even be cubic if the matrices satisfy certain symmetry properties [**59**].

A few words about global convergence are in order. The unitary circulant shift matrix C_n exemplified by the four-by-four case

$$C_4 = \begin{bmatrix} & & & 1 \\ 1 & & & \\ & 1 & & \\ & & 1 & \end{bmatrix}$$

is invariant under QR iterations with zero shifts, as is any unitary matrix. The shift strategy described in the previous paragraph gives zero shifts, as long as $p < n$. Thus the algorithm fails to converge when applied to C_n. Even worse things can happen; in some cases the shifts can wander chaotically [**6**]. The standard cure for these problems is to use *exceptional shifts* (for example, random shifts) if many iterations have passed with no progress. The point of this strategy is to knock the matrix away from any dangerous areas. It is not foolproof [**18**], but it has worked well over the years. Nevertheless, a shift strategy that is provably globally convergent (and converges quadratically on almost all matrices) would be welcome.

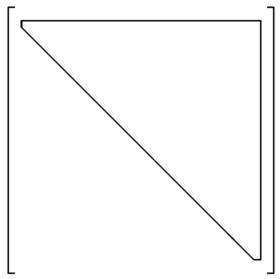
The only class of matrices for which global convergence has been proved is that of Hermitian tridiagonal matrices, provided that the Wilkinson shift strategy is used [**41**]. The Wilkinson strategy takes $p = 1$; the lone shift is the eigenvalue of the 2×2 lower right-hand submatrix that is closer to a_{nn}.

3. Implicit Implementations of GR Algorithms

For most of the choices of f that we have considered, the cost of calculating $f(A)$ is high. For this and other reasons, most implementations of GR algorithms find a way to perform the iterations without calculating $f(A)$ explicitly. Usually the first column of $f(A)$ is all that is needed. This section shows how to do it when f is a polynomial.

[3]This result ignores the effect of roundoff errors. In practice the $(2,1)$ block of (2.5) will not be exactly zero, and usually it will not be small enough to allow a safe deflation of the problem.

If we wish to use an implicit GR algorithm, we must first transform the matrix to a condensed form. The best known such form is upper Hessenberg, but there are others. For example, any Hermitian matrix can be put into tridiagonal form, and so can almost any other square matrix [**61**], although the stability of the transformation comes into question for non-Hermitian matrices. For unitary matrices there are several condensed forms, including the Schur parameter pencil [**1**], [**12**], [**53**] and the double staircase form [**8**], [**53**]. For Hamiltonian matrices there are both Hessenberg-like and tridiagonal-like forms [**40**], [**13**]. Implicit GR algorithms can be built on all of these forms, but for simplicity we will restrict our attention to upper Hessenberg form.



A matrix A is in *upper Hessenberg* form if $a_{ij} = 0$ whenever $i > j + 1$. Every matrix can be transformed stably to upper Hessenberg form by a unitary similarity transformation [**61**], [**31**], [**52**]. There are also various useful nonunitary reductions to Hessenberg form, and these will play a role in what follows. The general plan of all of these reduction algorithms is that they first introduce zeros in the first column, then the second column, then the third column, and so on.

An upper Hessenberg matrix A is in *proper* upper Hessenberg form if $a_{j+1,j} \neq 0$ for $j = 1, \ldots, n-1$. If a matrix is not in proper upper Hessenberg form, we can divide its eigenvalue problem into independent subproblems for which the matrices *are* proper upper Hessenberg.

Suppose A is a proper upper Hessenberg matrix, and we wish to perform an iteration of a multishift GR algorithm:

$$f(A) = GR, \tag{3.1}$$

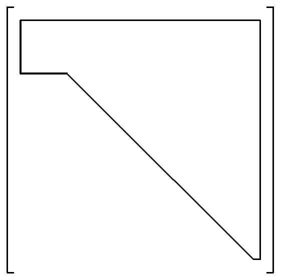
$$\hat{A} = G^{-1}AG, \tag{3.2}$$

where f is a polynomial of degree p: $f(A) = (A - \mu_1 I) \cdots (A - \mu_p I)$. Since we are considering only a single iteration, we have dropped the subscripts to simplify the notation. There is no need to calculate $f(A)$; it suffices to compute the first column, which is

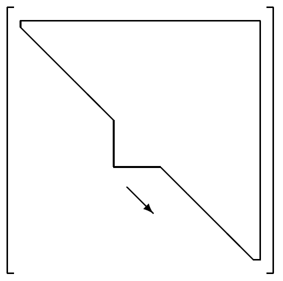
$$x = (A - \mu_1 I) \cdots (A - \mu_p I) e_1.$$

Since A is upper Hessenberg, only the first $p + 1$ entries of x are nonzero, and x can be computed in $O(p^3)$ flops. This is negligible if $p \ll n$.

The implicit GR iteration is set in motion by building a nonsingular matrix $\tilde{G}$ that has its first column proportional to x and looks like an identity matrix except for the $(p+1) \times (p+1)$ submatrix in the upper left-hand corner. There are many ways to do this; for example, $\tilde{G}$ can be a Householder reflector. $\tilde{G}$ is then used to perform a similarity transformation $A \to \tilde{G}^{-1}A\tilde{G}$, which disturbs the upper Hessenberg form; the transformed matrix has a bulge, the size of which is proportional to p, the degree of the iteration.



The rest of the iteration consists of returning the matrix to upper Hessenberg form by any one of the standard reduction algorithms. As the columns are cleared out one by one, new nonzero entries are added to the bottom of the bulge, so the bulge is effectively chased from one end of the matrix to the other.



Hence these algorithms are called *bulge-chasing* algorithms. Once the bulge has been chased off of the bottom of the matrix, the iteration is complete.

Let G denote the product of all of the transforming matrices applied during the iteration, so that the entire similarity transformation is $\hat{A} = G^{-1}AG$. Watkins and Elsner [**58**] showed that no matter what kind of transforming matrices are used, G satisfies $p(A) = GR$ for some upper-triangular R. Thus the procedure just outlined effects a GR iteration (3.1,3.2) implicitly. It follows that the GR convergence theory [**59**] is applicable to all algorithms of this type.

Let us consider some of the possibilities. If $\tilde{G}$ and all of the bulge-chasing transformations are unitary, then G is unitary, so a QR iteration is performed. This is by far the most popular choice. If, on the other hand, all of the transformations are elementary lower triangular (Gaussian elimination) transformations (without pivoting), then G is unit lower triangular, and an LR iteration is performed. For stability one can perform a row interchange to maximize the pivot before each elimination. This is how one implements the LR algorithm with pivoting. Unless the matrix has some special structure that one wishes to preserve (e.g. symmetric, Hamiltonian), there is no reason to insist that all of the transforming matrices be of the same type. Haag and Watkins [**32**] have developed bulge-chasing algorithms that mix unitary and Gaussian elimination transformations.

4. Performance of Multishift QR Algorithms

We now confine our attention to the QR algorithm, although this restriction is by no means necessary. In principle we can perform multishift QR steps of any degree p. What is a good choice of p in practice? Historically the first choice was

$p = 1$, and this is still popular. The most widely used QR codes for real symmetric matrices and for complex non-Hermitian matrices make this choice. Another early choice that is still popular is $p = 2$, which allows the use of complex shifts on real matrices without going outside the real number field. That was Francis's reason for inventing the double-shift algorithm. Descendents of Francis's code are still in widespread use in Matlab, EISPACK, LAPACK, and elsewhere, as we have already mentioned. For many years 1 and 2 were the only choices of p that were used. The structure of certain types of matrices [**17**] causes their eigenvalues to come in sets of four (e.g. λ, $\overline{\lambda}$, $-\lambda$, $-\overline{\lambda}$). For these matrices the choice $p = 4$ is obviously in order. The use of large values of p was first advocated by Bai and Demmel [**3**]. This seemed like an excellent idea. If one gets, say, thirty shifts from the lower right hand 30×30 submatrix and uses them for a QR step of degree $p = 30$, then one has to chase a 30×30 bulge. This is like doing 30 steps at a time, and it entails a lot of arithmetic. Since the computations are quite regular, they can be implemented in level-2 (or possibly level-3) BLAS [**23**], [**22**], thereby enhancing performance on modern vector, cache-based, or parallel computers.

Unfortunately the multishift QR algorithm does not perform well if the degree p is taken too large. This empirical fact is at odds with the convergence theory and came as a complete surprise. Some experiments of Dubrulle [**25**] showed that the problem lies with roundoff errors. If p shifts are chosen, they can be used to perform either one QR iteration of degree p (chasing one big bulge) or $p/2$ iterations of degree two (chasing $p/2$ small bulges). In principle the two procedures should yield the same result. Dubrulle showed that in practice they do not: The code that chases many small bulges converges rapidly as expected, while the one that chases fewer large bulges goes nowhere. The difference is entirely due to roundoff errors.

We were able to shed some light on the problem by identifying the mechanism by which information about the shifts is transmitted through the matrix during a bulge chase [**56**]. The shifts are used only at the very beginning of the iteration, in the computation of the vector x that is used to build the transforming matrix that creates the bulge. The rest of the algorithm consists of chasing the bulge; no further reference to the shifts is made. Yet good shifts are crucial to the rapid convergence of the algorithm. In the case of multishift QR, convergence consists of repeated deflation of (relatively) small blocks off of the bottom of the matrix. The good shifts are supposed to accelerate these deflations. Thus the information about the shifts must somehow be transmitted from the top to the bottom of the matrix during the bulge chase, but how? In [**56**] we demonstrated that the shifts are transmitted as eigenvalues of a certain matrix pencil associated with the bulge. When p is large, the eigenvalues of this *bulge pencil* tend to be ill conditioned, so the shift information is not represented accurately. The shifts are *blurred*, so to speak. The larger p is, the worse is the blurring. When $p = 30$, it is so bad that the shifts are completely lost. The algorithm functions as if random shifts had been applied. From this perspective it is no longer a surprise that multshift QR performs poorly when $p = 30$.

The multishift idea has not been abandoned. The main workhorse in LAPACK [**26**] for solving nonsymmetric eigenvalue problems is a multishift QR code. In principle this code can be operated at any value of p, but $p = 6$ has been chosen for general use. At this value the shift blurring is slight enough that it does not seriously degrade convergence, and a net performance gain is realized through the use of Level 2 BLAS.

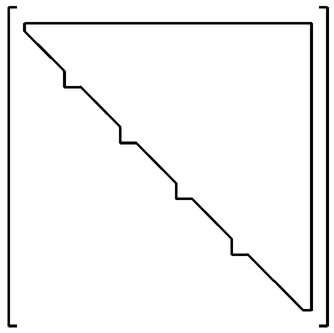


FIGURE 1. Pipelined QR steps

5. Pipelined QR Algorithm

Through Dubrulle's experiments it became clear that one can perform a QR iteration of degree 30, say, by chasing fifteen bulges of degree 2. This works well because the shifts are not blurred at all when $p = 2$. Once we have set one bulge in motion, we can start the next bulge as soon as we please; there is no need to wait for completion of the first bulge chase. Once we have set the second bulge in motion, we can start the third, and so on. In this way we can chase all fifteen (or however many) bulges simultaneously in pipeline fashion.

Imagine a matrix that is really large and is divided up over many processors of a distributed memory parallel computer. If the bulges are spread evenly, as shown in the figure, a good many processors can be kept busy simultaneously.

The idea of pipelining QR steps is not new. For example, it has been considered by Heller and Ipsen [**33**], Stewart [**47**], van de Geijn [**49**], [**50**], and Kaufman [**37**], but the idea did not catch on right away because nobody thought of changing the shift strategy. For bulges of degree two, the standard strategy is to take as shifts the two eigenvalues of the lower right-hand 2×2 submatrix. The entries of this submatrix are among the last to be computed in a QR step, for the bulge is chased from top to bottom. If one wishes to start a new step before the bulge for the current step has reached the bottom of the matrix, one is forced to use old shifts because the new ones are not available yet. If one wants to keep a steady stream of, say, fifteen bulges running in the pipeline, one is obliged to use shifts that are fifteen iterations out of date, so to speak. The use of such "stale" shifts degrades the convergence rate significantly.

But now we are advocating a different strategy [**54**]: Choose some even number p (e.g. 30) and get p shifts by computing the eigenvalues of the lower right-hand $p \times p$ matrix. Now we have enough shifts to chase $p/2$ bulges in pipeline fashion without resorting to out-of-date shifts. This strategy works well. It is used in ScaLAPACK's parallel QR code [**34**] for nonsymmetric eigenvalue problems.

Numerous improvements are possible. For example, the arithmetic could be performed more efficiently if the bulges were chased in (slightly blurred) packets of six instead of two. Another possibility is to chase tight clusters of small bulges, as in recent work of Braman, Byers, and Mathias [**9**]. As a cluster of bulges is chased through a segment of the matrix, the many small transforming matrices generated from the bulge chases can be accumulated in a larger orthogonal matrix, which can then be applied using level 3 BLAS [**22**]. A price is paid for this: the total number of flops per iteration is roughly doubled. The payoffs are that operations implemented in level 3 BLAS are easily parallelized and allow modern cache-based processors to operate at near top speed. Another innovation of [**9**] is the introduction of a more aggressive deflation strategy (and accompanying shift strategy) that allows the algorithm to terminate in fewer iterations. These innovations appear to have a good chance for widespread acceptance in time.

6. Generalized Eigenvalue Problem

Matrix eigenvalue problems frequently present themselves as *generalized eigenvalue problems* involving a matrix pair (A, B), which is also commonly presented as a *matrix pencil* $A - \lambda B$. A nonzero vector v is an *eigenvector* of the matrix pencil with associated *eigenvalue* λ if

$$Av = \lambda Bv.$$

v is an eigenvector with eigenvalue ∞ if $Bv = 0$. The generalized eigenvalue problem reduces to the standard eigenvalue problem in the case $B = I$. In analogy with the standard eigenvalue problem we easily see that λ is a finite eigenvalue of the pencil if and only if $\det(A - \lambda B) = 0$. In contrast with the standard eigenvalue problem, the *characteristic polynomial* $\det(A - \lambda B)$ can have degree less than n. This happens whenever B is a singular matrix. A pencil is *singular* if its characteristic polynomial is identically zero. In this case every λ is an eigenvalue. A pencil that is not singular is called *regular*.

The QZ algorithm of Moler and Stewart [**39**] is a generalization of the QR algorithm that can be used to solve generalized eigenvalue problems for regular pencils. This is just one of a whole family of GZ algorithms [**60**]. A good implementation of a GZ algorithm will perform well, regardless of whether the B matrix is singular or not. However, it is much easier to explain how GZ algorithms work when B is nonsingular, so we shall make that assumption. One iteration of a GZ algorithm transforms a pencil $A - \lambda B$ to a strictly equivalent pencil $\hat{A} - \lambda \hat{B}$ as follows: a spectral transformation function f is chosen, then GR decompositions of $f(AB^{-1})$ and $f(B^{-1}A)$ are computed:

$$f(AB^{-1}) = GR, \qquad f(B^{-1}A) = ZS. \tag{6.1}$$

G and Z are nonsingular, and R and S are upper triangular. The nonsingular matrices G and Z are used to effect the equivalence transformation:

$$\hat{A} = G^{-1}AZ, \qquad \hat{B} = G^{-1}BZ. \tag{6.2}$$

If $B = I$, then we may take $G = Z$ in (6.1), in which case the GZ iteration reduces to a GR iteration.

Recombining the equations (6.2) we see immediately that

$$\hat{A}\hat{B}^{-1} = G^{-1}\left(AB^{-1}\right)G, \quad \text{and} \quad \hat{B}^{-1}\hat{A} = Z^{-1}\left(B^{-1}A\right)Z. \tag{6.3}$$

Equations (6.1) and (6.3) together show that an iteration of the GZ algorithm effects GR iterations on AB^{-1} and $B^{-1}A$ simultaneously. It follows then from the GR convergence theory that if we iterate this process with good choices of spectral transformation functions, both AB^{-1} and $B^{-1}A$ will normally converge rapidly to block upper triangular form. It is shown in [**60**] that the A and B matrices converge individually (at the same rate as AB^{-1} and $B^{-1}A$) to block triangular form

$$\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix} - \lambda \begin{bmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{bmatrix},$$

thus breaking the problem into two smaller problems involving the pencils $A_{11} - \lambda B_{11}$ and $A_{22} - \lambda B_{22}$.

These iterations are expensive unless can find an economical way to perform the equivalence transformation (6.2) without explicitly calculating B^{-1} (which may not exist), much less $f(AB^{-1})$ or $f(B^{-1}A)$. This is done by performing an initial transformation to a condensed form, usually Hessenberg-triangular form. By this we mean that A is made upper Hessenberg and B upper triangular. (Thus AB^{-1} and $B^{-1}A$ are both upper Hessenberg.) Then the GZ step is effected by a process that chases bulges through A and B. The bulges are first formed by a transformation G_1 whose first column is proportional to the first column of $f(AB^{-1})$. This can be computed cheaply if f is a polynomial of degree $p \ll n$, since AB^{-1} is upper Hessenberg. It can be done without explicitly assembling B^{-1}, and it has a reasonable interpretation even if B^{-1} does not exist. Once the bulges have been formed, the rest of the iteration consists of a sequence of transformations that return the pencil to Hessenberg-triangular form by a process that chases the bulges from top to bottom of the matrices. It is similar to the GR bulge-chasing process, but there are extra details. See [**60**], [**31**], or the original Moler-Stewart paper [**39**].

The type of GZ iteration that the bulge chase effects depends on what kinds of transformations are used to do the chasing. For example, if all transformations are unitary, a QZ step results. If Gaussian elimination transformations (with pivoting) are used, an iteration of the LZ algorithm [**36**] results. Other examples are the SZ algorithm for symplectic butterfly pencils [**7**], and the HZ algorithm for pencils of the form $T - \lambda D$, where T is symmetric and D is diagonal with ± 1 entries on the main diagonal. This is a reformulation of the SR algorithm for matrices of the form DT $(= D^{-1}T)$.

Surely the most heavily used GZ code is the one in Matlab. This is a single-shift $(p = 1)$ implicit QZ algorithm that uses complex arithmetic. The original QZ algorithm of Moler and Stewart [**39**] used $p = 2$ for real matrices, following Francis. The QZ codes in LAPACK use either $p = 1$ or $p = 2$, depending on whether the shifts are real or complex.

As far as we know, no parallel QZ code has been written so far. The various approaches that have been tried for QR can also be applied to QZ. For example, one can take $p > 2$ and chase larger bulges [**60**], but this is more difficult to implement than in the QR case. Shift blurring is also a problem if p is too large. The idea of chasing many small bulges in pipeline fashion should work as well for QZ as it does for QR.

Once the QZ algorithm is finished, the pencil will have been reduced to upper triangular form or nearly triangular form. For simplicity let us suppose the form is triangular. Then the eigenvalues are the quotients of the main diagonal entries: $\lambda_i = a_{ii}/b_{ii}$. If $a_{ii} \neq 0$ and $b_{ii} = 0$ for some i, this signifies an infinite eigenvalue. If $a_{ii} = 0$ and $b_{ii} = 0$ for some i, the pencil is singular. In that case the other a_{jj}/b_{jj} signify nothing, as they can take on any values whatsoever [**63**]. Singular pencils have fine structure that can be determined by the staircase algorithm of Van Dooren [**24**]. See also the code GUPTRI of Demmel and Kågström [**20**].

7. Divide-and-Conquer Algorithms

To round out the paper we consider a completely different class of algorithm that has been under development in recent years [**5**, **4**]. They are not usually viewed as GZ algorithms, but that is what they are. They are explicit GZ algorithms; that is, they actual compute $f(AB^{-1})$ and $f(B^{-1}A)$ and their GR decompositions explicitly. They require more computation than a conventional implicit GZ algorithm does, but the computations are of types that can be implemented using level 3 BLAS. They also have a divide-and-conquer aspect. Thus algorithms of this type have a chance of becoming the algorithms of choice for parallel solution of extremely large, dense eigenvalue problems.

Let D be a subset of the complex plane (e.g. a disk) that contains some, say k, of the eigenvalues of the pencil $A - \lambda B$. Ideally $k \approx n/2$. Let $f = \chi_D$, the characteristic function of D. Thus $f(z)$ is 1 if $z \in D$ and 0 otherwise. If we then perform a GZ iteration (6.1,6.2) driven by this f, the resulting pencil normally has the form

$$\hat{A} - \lambda\hat{B} = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix} - \lambda \begin{bmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{bmatrix}, \tag{7.1}$$

where $A_{11} - \lambda B_{11}$ is $k \times k$ and carries the eigenvalues that lie within D. Thus in one (expensive) iteration we divide the problem into two subproblems, which are of about equal size if $k \approx n/2$. A few such divisions suffice to conquer the problem.

It is easy to see why the split occurs. Let $\mathcal{S}_d$ and $\mathcal{S}_r$ be the invariant subspaces of $B^{-1}A$ and AB^{-1}, respectively, associated with the eigenvalues that lie in D. Then $(\mathcal{S}_d, \mathcal{S}_r)$ is a deflating pair for the pencil, i.e. $A\mathcal{S}_d \subseteq \mathcal{S}_r$ and $B\mathcal{S}_d \subseteq \mathcal{S}_r$. Since f is the characteristic function of D, $f(B^{-1}A)$ and $f(AB^{-1})$ are spectral projectors onto $\mathcal{S}_d$ and $\mathcal{S}_r$, respectively. When a decomposition $f(AB^{-1}) = GR$ is performed, the upper-triangular matrix R normally has the form

$$R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix},$$

where R_{11} is $k \times k$ and nonsingular, because $f(AB^{-1})$ has rank k. We can be sure of obtaining an this form if we introduce column pivoting in the GR decomposition: $f(AB^{-1}) = GR\Pi$, where R has the desired form and Π is a permutation matrix. This guarantees that the first k columns of G form a basis for $\mathcal{S}_r$, the range of $f(AB^{-1})$. If we likewise introduce pivoting into the decomposition of $f(B^{-1}A)$, we can guarantee that the first k columns of Z are a basis of $\mathcal{S}_d$. Thus if we replace (6.1) by

$$f(AB^{-1}) = GR\Pi, \qquad f(B^{-1}A) = ZSP, \tag{7.2}$$

where Π and P are suitable permutation matrices, then the transformation (6.2) will result in the form (7.1), because $\mathcal{S}_d$ and $\mathcal{S}_r$ are deflating subspaces.

This type of GZ algorithm yields a deflation on each iteration. In order to implement it, we need to be able to calculate $f(AB^{-1})$ and $f(B^{-1}A)$ for various types of regions D. Various iterative methods have been put forward. The main method discussed in [**4**] can be applied to an arbitrary disk D. The size and location of the disk are determined by a preliminary transformation. Therefore we can take D to be the unit disk without loss of generality. The iterative method described in [**4**] has the effect that if one stops after j iterations, one uses instead of f the rational approximation

$$f_j(z) = \frac{1}{1 + z^{2^j}}.$$

Even for modest values of j this approximation is excellent, except very near the unit circle.

The matrices $f_j(AB^{-1})$ and $f_j(B^{-1}A)$ are computed without ever forming B^{-1}; the algorithm operates directly on A and B. The major operations in the iteration are QR decompositions and matrix-matrix multiplications, which can be done in level 3 BLAS. In the decomposition (7.2) the matrices G and Z are taken to be unitary for stability, so this is actually a QZ algorithm. The algorithm works even if B is singular. See [**4**] for many more details.

Since the iterations that compute $f_j(AB^{-1})$ and $f_j(B^{-1}A)$ are expensive, one prefers not to perform too many of them. Difficulties arise when there is an eigenvalue on or very near the circle that divides D from its complement. The iterations may fail to converge or converge too slowly. The remedy is to move the disk and restart the iterations. Once the projectors and their QR decompositions have been computed, the transformation (6.2) does not deliver exactly the form (7.1). The $(2,1)$ block will not quite be zero in practice, because of roundoff errors and because the projectors have been calculated only approximately. If $\| A_{21} \|$ or $\| B_{21} \|$ is too big, the iteration must be rejected. Again the remedy is to move the disk and try again. Because the iterations are so expensive, one cannot afford to waste too many of them.

An experimental divide-and-conquer code (that uses a different iteration from the one discussed here) is available as a prototype code from ScaLAPACK.

References

[1] G. S. Ammar, W. B. Gragg, and L. Reichel, *On the eigenproblem for orthogonal matrices*, in Proc. 25th IEEE Conference on Decision and Control, Athens, New York, 1986, IEEE, pp. 1063–1066.

[2] B. T. Smith et. al., *Matrix Eigensystem Routines—EISPACK Guide*, Springer-Verlag, New York, Second ed., 1976.

[3] Z. Bai and J. Demmel, *On a block implementation of the Hessenberg multishift QR iteration*, Internat. J. High Speed Comput., 1 (1989), pp. 97–112.

[4] Z. Bai, J. Demmel, and M. Gu, *Inverse free parallel spectral divide and conquer algorithms for nonsymmetric eigenproblems*, Numer. Math., 76 (1997), pp. 279–308.

[5] Z. Bai et. al., *The spectral decomposition of nonsymmetric matrices on distributed memory parallel computers*, Tech. Rep. CS-95-273, University of Tennessee, 1995.

[6] S. Batterson and J. Smillie, *Rayleigh quotient iteration for nonsymmetric matrices*, Math. Comp., 55 (1990), pp. 169–178.

[7] P. Benner, H. F. bender, and D. S. Watkins, *Two connections between the SR and HR eigenvalue algorithms*, Linear Algebra Appl., 272 (1998), pp. 17–32.

[8] B. Bohnhorst, *Ein Lanczos-ähnliches Verfahren zur Lösung des unitären Eigenwertproblems*, PhD thesis, University of Bielefeld, 1993.

[9] K. Braman, R. Byers, and R. Mathias, *The multi-shift QR algorithm: aggressive deflation, maintaining well focused shifts, and level 3 performance.* manuscript, 1999.

[10] M. A. Brebner and J. Grad, *Eigenvalues of $Ax = \lambda Bx$ for real symmetric matrices A and B computed by reduction to pseudosymmetric form and the HR process*, Linear Algebra Appl., 43 (1982), pp. 99–118.

[11] A. Bunse-Gerstner, *An analysis of the HR algorithm for computing the eigenvalues of a matrix*, Linear Algebra Appl., 35 (1981), pp. 155–173.

[12] A. Bunse-Gerstner and L. Elsner, *Schur parameter pencils for the solution of the unitary eigenproblem*, Linear Algebra Appl., 154–156 (1991), pp. 741–778.

[13] A. Bunse Gerstner and V. Mehrmann, *A symplectic QR-like algorithm for the solution of the real algebraic Riccati equation*, IEEE Trans. Auto. Control, AC-31 (1986), pp. 1104–1113.

[14] A. Bunse-Gerstner, V. Mehrmann, and D. S. Watkins, *An SR algorithm for Hamiltonian matrices based on Gaussian elimination*, Methods Oper. Res., 58 (1989), pp. 339–358.

[15] H. J. Buurema, *A geometric proof of convergence for the QR method*, PhD thesis, University of Groningen, Netherlands, 1970.

[16] R. Byers, *Hamiltonian and Symplectic Algorithms for the Algebraic Riccati Equation*, PhD thesis, Cornell University, 1983.

[17] ———, *A Hamiltonian QR algorithm*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 212–229.

[18] D. Day, *How the QR algorithm fails to converge and how to fix it.* Mathematics of Numerical Analysis: Real Number Algorithms, August 1995.

[19] J. Della-Dora, *Numerical linear algorithms and group theory*, Linear Algebra Appl., 10 (1975), pp. 267–283.

[20] J. Demmel and B. K. gström, *GUPTRI.* NETLIB, 1991. http://www.netlib.org/.

[21] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPACK Users' Guide*, SIAM, Philadelphia, 1979.

[22] J. J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling, *A set of level 3 basic linear algebra subprograms*, ACM Trans. Math. Software, 16 (1990), pp. 1–17.

[23] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson, *An extended set of Fortran basic linear algebra subprograms*, ACM Trans. Math. Software, 14 (1988), pp. 1–17.

[24] P. V. Dooren, *The computation of Kronecker's canonical form of a singular pencil*, Linear Algebra Appl., 27 (1979), pp. 103–140.

[25] A. A. Dubrulle, *The multishift QR algorithm—is it worth the trouble?*, Tech. Rep. G320-3558x, IBM Corp., Palo Alto, 1991.

[26] E. A. et. al., *LAPACK Users' Guide*, SIAM, Philadelphia, Second ed., 1995. `http://www.netlib.org/lapack/lug/lapack_lug.html`.

[27] L. S. B. et. al., *ScaLAPACK Users' Guide*, SIAM, Philadelphia, 1997. `http://www.netlib.org/scalapack/slug/scalapack_slug.html`.

[28] K. V. Fernando and B. N. Parlett, *Accurate singular values and differential qd algorithms*, Numer. Math., 67 (1994), pp. 191–229.

[29] J. G. F. Francis, *The QR transformation, parts I and II*, Computer J., 4 (1961), pp. 265–272, 332–345.

[30] G. A. Geist, G. W. Howell, and D. S. Watkins, *The BR eigenvalue algorithm*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 1083–1098.

[31] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Third ed., 1996.

[32] J. B. Haag and D. S. Watkins, *QR-like algorithms for the nonsymmetric eigenvalue problem*, ACM Trans. Math. Software, 19 (1993), pp. 407–418.

[33] D. E. Heller and I. C. F. Ipsen, *Systolic networks for orthogonal decompositions*, SIAM J. Sci. Stat. Comput., 4 (1983), pp. 261–269.

[34] G. Henry, D. Watkins, and J. Dongarra, *A parallel implementation of the nonsymmetric QR algorithm for distributed memory architectures*, Tech. Rep. LAPACK Working Note 121, University of Tennessee, 1997.

[35] A. S. Householder, *The Theory of Matrices in Numerical Analysis*, Dover Books, 1964.

[36] L. C. Kaufman, *The LZ algorithm to solve the generalized eigenvalue problems*, SIAM J. Numer. Anal., 11 (1974), pp. 997–1024.

[37] ———, *A parallel QR algorithm for the symmetric tridiagonal eigenvalue problem*, J. Parallel and Distributed Computing, 23 (1994), pp. 429–434.

[38] V. N. Kublanovskaya, *On some algorithms for the solution of the complete eigenvalue problem*, USSR Comput. Math. and Math. Phys., 3 (1961), pp. 637–657.

[39] C. B. Moler and G. W. Stewart, *An algorithm for generalized matrix eigenvalue problems*, SIAM J. Numer. Anal., 10 (1973), pp. 241–256.

[40] C. Paige and C. Van Loan, *A Schur decomposition for Hamiltonian matrices*, Linear Algebra Appl., 41 (1981), pp. 11–32.

[41] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice Hall, Engelwood Cliffs, New Jersey, 1980. Reprinted by SIAM, 1997.

[42] ———, *The new qd algorithms*, Acta Numerica, (1995), pp. 459–491.

[43] B. N. Parlett and W. G. Poole, *A geometric theory for the QR, LU, and power iterations*, SIAM J. Numer. Anal., 10 (1973), pp. 389–412.

[44] H. Rutishauser, *Der Quotienten-Differenzen-Algorithmus*, Z. angew. Math. Physik, 5 (1954), pp. 233–251.

[45] ———, *Der Quotienten-Differenzen-Algorithmus*, no. 7 in Mitt. Inst. angew. Math. ETH, Birkhäuser, Basel, 1957.

[46] ———, *Solution of eigenvalue problems with the LR-transformation*, Nat. Bur. Standards Appl. Math. Series, 49 (1958), pp. 47–81.

[47] G. W. Stewart, *A parallel implementation of the QR algorithm*, Parallel Comput., 5 (1987), pp. 187–196.

[48] R. van de Geijn and D. G. Hudson, *An efficient parallel implementation of the nonsymmetric QR algorithm*, in Proceedings of the fourth conference on hypercube concurrent computers and applications, 1989.

[49] R. A. van de Geijn, *Implementing the QR Algorithm on an Array of Processors*, PhD thesis, University of Maryland, 1987. Department of Computer Science TR-1897.

[50] ———, *Deferred shifting schemes for parallel QR methods*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 180–194.

[51] D. S. Watkins, *Understanding the QR algorithm*, SIAM Rev., 24 (1982), pp. 427–440.

[52] ———, *Fundamentals of Matrix Computations*, John Wiley and Sons, New York, 1991.

[53] ———, *Some perspectives on the eigenvalue problem*, SIAM Rev., 35 (1993), pp. 430–471.

[54] ———, *Shifting strategies for the parallel QR algorithm*, SIAM J. Sci. Comput., 15 (1994), pp. 953–958.

[55] ———, *QR-like algorithms—an overview of convergence theory and practice*, in The Mathematics of Numerical Analysis, J. Renegar, M. Shub, and S. Smale, eds., vol. 32 of Lectures in Applied Mathematics, American Mathematical Society, 1996.

[56] ———, *The transmission of shifts and shift blurring in the QR algorithm*, Linear Algebra Appl., 241–243 (1996), pp. 877–896.

[57] ———, *Bulge exchanges in algorithms of QR type*, SIAM J. Matrix Anal. Appl., 19 (1998), pp. 1074–1096.

[58] D. S. Watkins and L. Elsner, *Chasing algorithms for the eigenvalue problem*, SIAM J. Matrix Anal. Appl., 12 (1991), pp. 374–384.

[59] ———, *Convergence of algorithms of decomposition type for the eigenvalue problem*, Linear Algebra Appl., 143 (1991), pp. 19–47.

[60] ———, *Theory of decomposition and bulge-chasing algorithms for the generalized eigenvalue problem*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 943–967.

[61] J. H. Wilkinson, *The Algebraic Eigenvalue Problem,*, Clarendon Press, Oxford University, 1965.

[62] ———, *Global convergence of tridiagonal QR algorithm with origin shifts*, Linear Algebra Appl., 1 (1968), pp. 409–420.

[63] ———, *Kronecker's canonical form and the QZ algorithm*, Linear Algebra Appl., 28 (1979), pp. 285–303.

Department of Pure and Applied Mathematics, Washington State University, Pullman, Washington 99164-3113

E-mail address: `watkins@wsu.edu`