5-2011

A Restarted Homotopy Method for the Nonsymmetric Eigenvalue Problem

Brandon Hutchison
University of Arkansas, Fayetteville

A RESTARTED HOMOTOPY METHOD FOR THE NONSYMMETRIC EIGENVALUE PROBLEM

A RESTARTED HOMOTOPY METHOD FOR THE NONSYMMETRIC EIGENVALUE PROBLEM

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Mathematics

By

Brandon Hutchison
Christian Brothers University
Bachelor of Science in Mechanical Engineering, 1997
University of Arkansas
Master of Science in Mathematics, 2008

May 2011
University of Arkansas

Abstract

The eigenvalues and eigenvectors of a Hessenberg matrix H are computed with a combination of homotopy increments and the Arnoldi method. Given a set, Ω, of approximate eigenvalues of H, there exists a unique vector $f = f(H,\Omega) \in \mathbb{R}^n$ where $\lambda(H - e_1 f^t) = \Omega$. A diagonalization of the homotopy $H(t) = H - (1-t)e_1 f^t$ at $t = 0$ provides a prediction of the eigenvalues of $H(t)$ at later times. These predictions define a new Ω that defines a *new homotopy*. The correction for each eigenvalue has an $O(t^2)$ error estimate, enabling variable step size and efficient convergence tests. Computations are done primarily in real arithmetic, and bifurcations are avoided by restarting the homotopy with Arnoldi eigenvalues. Although the method is neither as elegant nor as robust as the QR algorithm, it is about twice as fast in the randomly generated examples considered and is highly parallelizable.

This dissertation is approved for
Recommendation to the
Graduate Council

Dissertation Director

Dr. Mark Arnold

Dissertation Committee

Dr. William Feldman

Dr. Yo'av Rieck

Dissertation Duplication Release

I hereby authorize the University of Arkansas Libraries to duplicate this dissertation when needed for research and/or scholarship.

Agreed _____

Brandon Hutchison

Acknowledgements

I would like to thank my advisor, Dr. Mark Arnold, for his guidance and input throughout the dissertation process. Five years ago, I was toying with the idea of leaving my engineering career to pursue a doctorate degree in mathematics. I scheduled a meeting with Dr. Arnold to learn about the program. His enthusiasm is contagious, and I came out of that meeting knowing I was going to make the career change. I suspect it is no coincidence that two years later he would become my dissertation advisor.

Dedication

I would have never had the option to pursue this work without the love, support, and motivation of my darling wife, Erica. She is by far the single greatest reason I am able to be at this point. This is for her and the bean.

Contents

Notation Conventions

I	The identity matrix
e_j	The jth column of the the identity matrix
e	The vector of ones
$A(i{:}j,k{:}p)$	The submatrix of A consisting of rows i to j and columns k to p
$A_{i,j}$	The (i,j) block of matrix A
A_{i,j_k}	The kth column of the (i,j) block of matrix A
A_i	The (i,i) diagonal block of A or the ith A in a sequence
$a_{i,j}$	The (i,j) element of matrix A or the ith element of vector a_j
y^*	The conjugate transpose of vector y
y^t	The transpose of vector y.

1 Introduction

The fundamental idea behind this method is the iterative improvement of an approximate eigensystem, which we present as a restarted homotopy method. Eigenvalue homotopy methods use the homotopy function

$$M(t) = At + (1-t)B \tag{1.1}$$

to find the eigenvalues of $M(1) = A \in \mathbb{R}^{n\times n}$ where those of $M(0) = B \in \mathbb{R}^{n\times n}$ are known. As the eigenvalues of a matrix are a continuous function of its entries, this homotopy provides eigenvalue paths, which can be followed from $t = 0$ to $t = 1$ where lie the eigenvalues of A.

In the method presented herein, after taking a time step along each eigenvalue homotopy path, the current predictions are used to define a new starting matrix, B_k, which is a rank 1 perturbation of A, and the process repeats as, hopefully, $B_k \to A$. The homotopy step is essentially a Taylor method of order 2 which provides cubic convergence near the solution and useful error estimates. The computations are nearly all in real arithmetic, working for the most part with a block diagonalization of B.

Homotopy methods for the nonsymmetric eigenvalue problem must somehow address the singularities of multiple eigenvalues caused by the bifurcation of a complex pair into two real eigenvalues, or vice-versa. Restarting the homotopy avoids the problem of detecting bifurcation points, but a mechanism must be included to allow for complex-to-real or real-to-complex exchanges. This is achieved through an Arnoldi minimization.

An *upper Hessenberg* matrix, H, or simply *Hessenberg* matrix, is one in which all

elements below the first subdiagonal are zero. That is, $h_{ij} = 0$ for $i > j+1$. A Hessenberg matrix for which no subdiagonal entry is zero is called an *unreduced* Hessenberg matrix. If a Hessenberg matrix is not unreduced and $h_{k+1,k} = 0$ for some k, then its eigenproblem splits into two subproblems: $H(1{:}k, 1{:}k)$ and $H(k+1{:}n, k+1{:}n)$. Because of this and the fact that any matrix $A \in \mathbb{R}^{n\times n}$ is orthogonally similar to a Hessenberg matrix, we will assume we are starting with an unreduced Hessenberg matrix, H.

Restarting the homotopy requires an estimate for the entire spectrum. Thus, the method as presented does not look useful for computing only a portion of the spectrum although some eigenvalues may converge faster than others. Since we are computing the entire spectrum, we desire favorable comparisons to the QR algorithm.

The paper is organized as follows. Section 2 introduces the eigenvalue problem. Section 3 develops the homotopy step and its components. Section 4 motivates the Arnoldi feedback minimization procedure as a mechanism for avoiding bifurcations. The proposed restarted homotopy algorithm is given in Section 5. In 6 we discuss issues with the algorithm as presented, and in 7 we discuss various perspectives on errors. Results comparing our method to the QR algorithm are given in Section 8.

2 Background

Given $A \in \mathbb{C}^{n\times n}$ find a nonzero $x \in \mathbb{C}^n$ and $\lambda \in \mathbb{C}$ such that

$$Ax = \lambda x. \tag{2.1}$$

This is the eigenvalue problem. Here x is known as a *right eigenvector* and λ is the corresponding *eigenvalue*. Similarly, if $y^*A = \lambda y^*$ for a nonzero $y \in \mathbb{C}^n$ then y^* is a *left eigenvector*. From (2.1) we see that (x, λ) is an eigenpair if and only if

$$(\lambda I - A)x = 0 \tag{2.2}$$

has a nontrivial solution. This is the case if and only if $\lambda I - A$ is singular or equivalently

$$det(\lambda I - A) = 0. \tag{2.3}$$

Equation (2.3) is known as the *characteristic equation* of A. The left side of (2.3) is known as the *characteristic polynomial* of A and can be expressed as

$$\lambda^n - c_1\lambda^{n-1} + c_2\lambda^{n-2} \cdots + (-1)^n c_n \tag{2.4}$$

where c_i is the sum of all principal minors of order i for A. A *principal minor* of order i for a matrix is the determinant of a submatrix of size $i \times i$ obtained by removing the rows and columns with indeces from $N = \{k_1, k_2 \ldots, k_{n-i}\} \subset \{1, 2, \ldots, n\}$.

The eigenvalues of a matrix are the roots of its characteristic polynomial. From (2.4) we see that there are exactly n eigenvalues, counting multiplicities. The set of all eigenvalues of A is called the *sprectrum* of A and is denoted $\lambda(A)$. The spectrum of a matrix is invariant

under similarity transformations since

$$\begin{aligned} det(\lambda I - X^{-1}AX) &= det(X^{-1}\lambda IX - X^{-1}AX) \\ &= det(X^{-1}(\lambda I - A)X) \\ &= det(X^{-1})det(\lambda I - A)det(X) \\ &= det(\lambda I - A) \end{aligned}$$

Further, if $A \in \mathbb{R}^{n\times n}$, as will be the focus of this work, the coefficients of (2.4) are all real, and $\lambda(A)$ is closed under complex conjugation. The *algebraic multiplicity* of an eigenvalue is its multiplicity as a root of the characteristic polynomial. An eigenvalue is called *simple* if its algebraic multiplicity is one. The eigenvalue algorithm presented here requires all eigenvalues to be simple for reasons presented later. The *geometric multiplicity* of an eigenvalue, λ, is the dimension of the null space of $\lambda I - A$. This is equivalent to the size of the maximum list of linearly independent eigenvectors associated with λ. The geometric multiplicity is always at least 1 and never greater than the algebraic multiplicity. If the geometric multiplicity equals the algebraic multiplicity for all eigenvalues of A, then A is diagonalizable as it has n linearly independent eigenvectors, $\{x_1, x_2, \ldots, x_n\}$, with $Ax_i = \lambda_i x_i$. If we let $X = \{x_1, x_2, \ldots, x_n\}$ and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$, then $X^{-1}AX = \Lambda$.

Theorem 2.1 *If H is an unreduced Hessenberg matrix, then the geometric multiplicity of all its eigenvalues is 1.*

Proof: Since H is unreduced, the rank of $\lambda I - H$ is at least $n-1$ as its first $n-1$ columns are clearly linearly independent regardless of λ . Therefore the dimension of the null space

of $\lambda I - H$ is at most 1 and is exactly 1 only when λ is an eigenvalue of H. □

Theorem 2.2 *An unreduced Hessenberg matrix, H, is diagonalizable if and only if all of its eigenvalues are simple.*

Proof: ($\Rightarrow$) Since H is diagonalizable it has n linearly independent eigenvectors. By the previous theorem, each of these eigenvectors is associated with an eigenvalue that has geometric multiplicity 1. Thus, there must be n distinct eigenvalues.

($\Leftarrow$) This is clear from a previous discussion. □

2.1 A Brief Overview of the QR Algorithm

A very well known result from Abel and Galois says that, in general, the roots of a polynomial of degree five or greater cannot be determined in a finite number of steps. Therefore, eigenvalue finding algorithms are iterative by nature. There are many eigenvalue algorithms. The properties of the matrix, such as symmetric/nonsymmetric and dense/sparse, often dictate which eigenvalue algorithm should be selected. The preferred algorithm can also depend on whether the full spectrum is desired or just a certain portion. The algorithm presented herein is designed to find the full spectrum of dense, nonsymmetric matrices. The current benchmark for finding the full spectrum of a dense matrix is the QR algorithm with implicit shifts. Later, we will present results from experiments comparing our restarted homotopy method to the QR algorithm with double implicit shifts and with the EIG function in MATLAB®, which uses multiple shifts. Also, we use the QR algorithm to supply the initial eigenvalue estimates needed for the first homotopy iteration. Thus, we will now give a brief discussion of the QR algorithm.

The QR iteration was introduced by Francis in [6]. Beautiful in its simplicity, the algorithm uses QR factorizations to compute the spectrum of A through a series of similarity transformations that under certain restrictions converge toward an upper triangular matrix where the eigenvalues are the diagonal entries. The restrictions are $|\lambda_1| < |\lambda_2| < \ldots < |\lambda_n|$ and the leading principal minors of a matrix of right eigenvectors of A are all nonzero. Refer to [22] for a more detailed discussion about the convergence of the basic QR algorithm.

Algorithm 2.1: Basic QR Algorithm

Input: $A \in \mathbb{R}^{n \times n}$

Output: H an upper triangular matrix similar to A

Put A in Hessenberg form $H = Q^* A Q$

$H_1 = H$

for $k = 1, 2, \ldots$

$Q_k R_k = H_k$

$H_{k+1} = R_k Q_k$

end

It is not obvious from the algorithm that each H_k is similar to H but this is easily dispelled as $H_{k+1} = R_k Q_k = Q_k^* Q_k R_k Q_k = Q_k^* H_k Q_k$. The problem with this algorithm is that it is expensive, $O(n^3)$ per iteration, and slow with only linear convergence. We can improve on the cost by monitoring the subdiagonal elements of each Hessenberg H_k for opportunities to deflate the problem. If $h_{k(i+1,i)} \approx 0$ for some k and i, the eigenproblem for A splits into the two smaller problems: $H_k(1{:}i,1{:}i)$ and $H_k(i{+}1{:}n,i{+}1{:}n)$. A typical criterion for the problem to deflate, such as used in LAPACK [12], is if

$$h_{k(i+1,i)} < \mathbf{u}(|h_{k(i,i)}| + |h_{k(i+1,i+i)}|)$$

with **u** the machine precision.

Francis also showed in [6] the convergence rate can be increased by employing shifts.

Algorithm 2.2: QR Algorithm with Single Shifts

Input: $A \in \mathbb{R}^{n\times n}$

Output: H an upper triangular matrix similar to A

Put A in Hessenberg form $H = Q^*AQ$

$H_1 = H$

for $k = 1, 2, \ldots$

$Q_k R_k = H_k - \rho_k I$ for some $\rho_k \in \mathbb{C}$

$H_{k+1} = R_k Q_k + \rho_k I$

end

If H_k is unreduced and $\rho_k \in \lambda(H_k)$, then $H - \rho_k \mathrm{I}$ has rank $n-1$, and the properties of the QR factorization give $e_n^t R_k = 0$. Thus, the last row of H_{k+1} is $\rho_k e_n^t$, and the problem decouples. If the ρ_k are effectively chosen, then Algorithm 2.2 has qradratic convergence.

When H is real, it quite often has complex eigenvalues. For Algorithm 2.2 to converge in this case, it is necessary for $\rho_k \in \mathbb{C}$ at some point. This then forces $H_{k+1} \in \mathbb{C}^{n\times n}$ and increases the cost of each subsequent iteration. Francis showed in Part II of [6] that complex arithmetic can be avoided by applying what is now referred to as an implicit double shift.

Two subsequent iterations of Algorithm 2.2 produce the steps

$$\begin{aligned} Q_k R_k &= H_k - \rho_k I \\ H_{k+1} &= R_k Q_k + \rho_k I \\ Q_{k+2} R_{k+2} &= H_{k+1} - \rho_{k+1} I \\ H_{k+2} &= R_{k+2} Q_{k+2} + \rho_{k+1} \end{aligned} \tag{2.5}$$

where

$$\begin{aligned} Q_k Q_{k+1} R_{k+1} R_k &= Q_k (H_{k+1} - \rho_{k+1} I) R_k \\ &= Q_k (R_k Q_k + \rho_k I - \rho_{k+1} I) R_k \\ &= (H_k - \rho_k I) Q_k R_k + (\rho_k - \rho_{k+1}) Q_k R_k \\ &= (H_k - \rho_k I)(H_k - \rho_k I) + (\rho_k - \rho_{k+1})(H_k - \rho_k I) \\ &= (H_k - \rho_k I)(H_k - \rho_{k+1} I). \end{aligned}$$

If $H_k \in \mathbb{R}^{n \times n}$ and $\rho_k = \bar{\rho}_{k+1}$ for some k, then

$$\Gamma = (H_k - \rho_k I)(H_k - \rho_{k+1} I) \in \mathbb{R}^{n \times n}$$

and its QR factorization, $\Gamma = Q_\Gamma R_\Gamma$, is necessarily real. Further, $Q_\Gamma = Q_k Q_{k+1}$, $R_\Gamma = R_{k+1} R_k$, and $Q_\Gamma e_1 = \dfrac{\Gamma e_1}{\|\Gamma e_1\|}$. Thus,

$$H_{k+2} = Q_{k+1}^* Q_k^* H_k Q_k Q_{k+1} = Q_\Gamma^* H_k Q_\Gamma$$

and $H_{k+2} \in \mathbb{R}$. If H_{k+2} is calculated via (2.5), rounding errors will almost certainly prevent H_{k+2} from being real when $\rho_k = \bar{\rho}_{k+1}$. This could be dealt with by skipping H_{k+1} and computing the QR factorization of Γ explicity, but this is prohibitively expensive.

The implicit double shift strategy handles these technicalities and performs the transformation $H_{k+2} = Q_\Gamma^* H_k Q_\Gamma$ in $8n^2$ flops as opposed to $O(n^3)$ while bypassing H_{k+1} and complex arithmetic. To illustrate how, we first introduce the *Implicit Q* theorem, a proof of which is available in [7]

Theorem 2.3 *(The Implicit Q Theorem) Let Q and U be orthogonal matrices such that $Q^*AQ = H$ and $U^*AU = \tilde{H}$ are both unreduced Hessenberg. If $q_1 = u_1$ then $q_i = \pm u_i$ and $h_{i+1,i} = \pm\tilde{h}_{i+1,i}$. In the case when H is not unreduced and $h_{k+1,k}$ is its first zero subdiagonal element, then $q_i = \pm u_i$, $h_{i,i-1} = \pm\tilde{h}_{i,i-1}$ for $i = 2,3,\ldots,k$ and $\tilde{h}_{k+1,k} = 0$.*

The idea behind the implicit double shift is to find a transformation U with $Ue_1 = Q_\Gamma e_1$ that can be applied to H_k in place of Q_Γ more efficiently. To this end, let U_1 be a Householder matrix such that $U_1\Gamma e_1$ is a multiple of e_1. As only the first three entries of Γe_1 are nonzero,

$$U_1 = \left[\begin{array}{c|c} \tilde{U}_1 & \\ \hline & I \end{array}\right]$$

where $\tilde{U}_1$ is 3×3. Further $U_1 e_1 = \dfrac{\Gamma e_1}{\|\Gamma e_1\|}$. The transformation $U_1 H_k U_1$ is Hessenberg except for the upper 4×4 submatrix referred to as a bulge. Using Householder matrices $U_2, U_3, \ldots, U_{n-1}$ to return $U_1 H_k U_1$ to Hessenberg form by "chasing the bulge" and letting

$U = U_1 U_2 \cdots U_{n-1}$ gives

$$\begin{aligned}\tilde{H}_{k+2} &= (U_{n-1} \cdots U_2 U_1) H_k (U_1 U_2 \cdots U_{n-1}) \\ &= U^t H_k U\end{aligned}$$

As the first column of U_i is e_1 for $i = 2, 3, \ldots n-1$, we have

$$U e_1 = (U_1 U_2 \cdots U_{n-1}) e_1 = U_1 e_1 = \tilde{Q} e_1.$$

As U and $\tilde{Q}$ have the same first row and each transforms H_k into Hessenberg form, the *Implicit Q* theorem implies U and $\tilde{Q}$ are the same upto signs as are the subdiagonal elements of H_{k+2} and $\tilde{H}_{k+2}$ upto the first zero subdiagonal entry. The transformations U_i only operate on a few of the rows/columns of H_k and can be computed and applied to H_k with only $8n^2$ total flops.

This provides the main mechanism for the QR algorithm with implicit double shifts. An algorithm is given in [7]. This was used to create a routine for supplying the intial eigenvalue estimates needed for the homotopy method. The routine is also used for flop count comparisons.

3 A Homotopy Step

The initial conditions, B, in (1.1) are critical to any homotopy method. Ideally, we would like to choose B close to A in (1.1). Regardless of the criteria for selecting B, its eigenvalues must be known. Some methods, such as that in [14], choose B so that solving its eigenvalue

problem is easier than that of A. We take a different approach. We find a B whose spectrum is a prescribed set. This is done through control theory.

For any unreduced Hessenberg matrix $H \in \mathbb{C}^{n\times n}$, and any set Ω of n complex numbers, there exists a unique $f \in \mathbb{C}^n$ such that $\lambda(H - e_1 f^*) = \Omega$. Furthermore, if H is real and Ω is closed under complex conjugation, then f is real. This is a standard result in control theory, where e_1 is called the *input vector*, f is the *feedback vector*, and Ω is the *closed-loop poles* for (H, e_1, f) (see [10], [23]).

So given an approximation Ω to $\lambda(H)$, we have the following homotopy whose eigenvalues at $t = 0$ are Ω and at $t = 1$ are $\lambda(H)$:

$$\begin{aligned} M(t) &= Ht + (1-t)(H - e_1 f^t) \\ &= H - (1-t)e_1 f^t \end{aligned} \tag{3.1}$$

By computing the feedback, f, we can embed in $\mathbb{R}^{n\times n} \times [0,1]$ the eigenproblem for H as

$$(H - (1-t)e_1 f^t)x(t) = \lambda(t)x(t) \tag{3.2}$$

where $x(t) \in \mathbb{C}^n$ and $\lambda(t) \in \mathbb{C}$ are an eigenpair of M at time t.

Let us define an eigenpath to be the path in $\mathbb{C}$ traversed by $\lambda(t)$, an eigenvalue of $M(t) = H - (1-t)e_1 f^t$, and not the path in $\mathbb{C} \times \mathbb{C}^n$ consisting of both the eigenvalue and eigenvector. The following result says that if $M(t)$ differs from H by a rank one matrix, then the eigenpaths depend only on the eigenvalues of $M(0) = B$. In this sense, all rank one homotopy methods follow the same paths. The statement is not vacuous since for any

collection Ω of n complex numbers, and for almost all $A \in \mathbb{C}^{n\times n}$ and $u \in \mathbb{C}^n$, there exist a unique $v \in \mathbb{C}^n$ such that $\lambda(A+uv^*) = \Omega$. That is, there are many B matrices giving the same $\lambda(M(0))$, but all of them generate the same eigenpaths with the same parameterizations. Thus, we are not concerned that restricting the input vector to e_1 will adversely affect the homotopy paths.

Theorem 3.1 *Let $A \in \mathbb{R}^{n\times n}$, $b, f, u, v \in \mathbb{R}^n$, and $t \in \mathbb{R}$. Define*

$$p(x,t) = det(xI - A - bf^t + tbf^t)$$

and

$$q(x,t) = det(xI - A - uv^t + tuv^t).$$

Then $p(x;0) = q(x;0)$ implies $p = q$.

Proof: Notice that $p(x,1) = q(x,1)$. We will show that each coefficient of $p(x,t)$ and $q(x,t)$ is linear in t. Thus, if $p(x,0) = q(x,0)$, then p and q agree at two points and must be equal. To that end, write

$$p(x;t) = x^n - c_1(t)x^{n-1} + c_2(t)x^{n-2} - \cdots \pm c_n(t),$$

where $c_k(t)$ is the sum of all principal minors of $A + bf^t - tbf^t$ of size k. Likewise, write

$$q(x;t) = x^n + \sum_{k=1}^{n} (-1)^k d_k(t) x^{n-k}$$

where $d_k(t)$ is the sum of all principal minors of $A + uv^t - tuv^t$ of size k.

Let α be a set of $n-k$ integers taken from $\{1,2,\ldots,n\}$ and X_α be the principal submatrix of X indexed by α. Then

$$c_k(t) = \sum_\alpha \det([A + bf^t - tbf^t]_\alpha).$$

Since $[bf^t]_\alpha = b_\alpha f_\alpha^t$, we have

$$\det([A + bf^t - tbf^t]_\alpha) = \det(A_\alpha + b_\alpha f_\alpha^t - tb_\alpha f_\alpha^t).$$

Provided $b_\alpha \neq 0$ we can find an invertible matrix P such that $P^{-1}b_\alpha = e_1$.

Let $\tilde{A}_\alpha = P^{-1}(A_\alpha + b_\alpha f_\alpha^t)P$ and $r_\alpha^t = f_\alpha^t P$. Now expanding the determinant about the first row gives

$$\begin{aligned}
\det(A_\alpha + b_\alpha f_\alpha^t - tb_\alpha f_\alpha^*) &= \det(\tilde{A}_\alpha + te_1 r_\alpha^t) \\
&= \sum_{j=1}^{k} (-1)^{j+1}(\tilde{a}_{1j} + tr_j)\det(\tilde{A}_{1j}) \\
&= \sum_{j=1}^{k} (-1)^{j+1}\tilde{a}_{1j}\det(\tilde{A}_{1j}) + t\sum_{j=1}^{k}(-1)^{j+1}r_j\det(\tilde{A}_{1j}) \\
&\equiv det(A + bf^t)_\alpha + t\sigma_\alpha.
\end{aligned}$$

In the case when $b_\alpha = 0$ we have

$$\det(A_\alpha + b_\alpha f_\alpha^t - tb_\alpha f_\alpha^*) = \det(A_\alpha)$$

In either case, we have $\det([A + bf^t - tbf^t]_\alpha)$ is linear in t for all α, hence, $c_k(t)$ is linear in t. Similarly, $d_k(t)$ is linear in t. $\square$

Let us now get down to the business of following our eigenvalue paths. This will be done through the second order Taylor polynomial approximation

$$\lambda_k(h_k) = \lambda_k(0) + h_k\dot{\lambda}_k(0) + \frac{h_k^2}{2}\ddot{\lambda}_k(0) + O(h_k^3) \tag{3.3}$$

for $h_k \in (0,1]$, $k = 1,2,\ldots,n$.

We may differentiate (3.2) in order to get $\dot{\lambda}_k$. If H is unreduced, then $H-(1-t)e_1f^t$ is unreduced for all t. Recall that an unreduced Hessenberg matrix is diagonalizable if and only if all of its eigenvalues are simple. If all of the eigenvalues of $H-e_1f^t$ are simple, then the eigenvalues of $H-(1-t)e_1f^t$ are also simple in a neigborhood of $t=0$. In this same neighborhood, we can define

$$X(t) = [x_1(t), x_2(t), \ldots, x_n(t)]$$

to be a matrix of right eigenvectors of $H-(1-t)e_1f^t$ and

$$X^{-1}(t) = Y(t) = [y_1(t), y_2(t), \ldots, y_n(t)]^*.$$

Then

$$(H-(1-t)e_1f^t)X(t) = X(t)\Lambda(t),$$

where $\Lambda(t) = diag(\lambda_1(t), \lambda_2(t), \ldots, \lambda_n(t))$. Differentiating (3.2) with respect to t and pre-

multiplying by the left eigenvector y_k^* yields

$$e_1 f^t x_k + (H - (1-t)e_1 f^t)\dot{x}_k = \dot{\lambda}_k x_k + \lambda_k \dot{x}_k \tag{3.4}$$

$$y_k^* e_1 f^t x_k + y_k^* (H - (1-t)e_1 f^t)\dot{x}_k = y_k^* \dot{\lambda}_k x_k + y_k^* \lambda_k \dot{x}_k$$

$$y_k^* e_1 f^t x_k + \lambda_k y_k^* \dot{x}_k = \dot{\lambda}_k + \lambda_k y_k^* \dot{x}_k$$

$$\dot{\lambda}_k = y_k^* e_1 f^t x_k$$

Let $\omega(t) = Y(t)e_1$ and $\xi^t(t) = f^t X(t)$. Then in a neighborhood of t=0 we have

$$\dot{\lambda}_k(t) = \omega_k(t)\xi_k(t). \tag{3.5}$$

In order to get an expression for $\ddot{\lambda}_k$, we first need to obtain an expression for $\dot{x}_k$ as a linear combination of the right eigenvectors. If we write $\dot{x}_k = Xc_k$, then for λ_k simple, we find by using (3.4) and premultiplying by y_i^* for $i \neq k$ that

$$e_1 f^t x_k + (H - (1-t)e_1 f^t)Xc_k = \dot{\lambda}_k x_k + \lambda_k Xc_k$$

$$y_i^* e_1 f^t x_k + y_i^* (H - (1-t)e_1 f^t)Xc_k = y_i^* \dot{\lambda}_k x_k + y_i^* \lambda_k Xc_k$$

$$\omega_i \xi_k + \lambda_i y_i^* Xc_k = \lambda_k y_i^* Xc_k$$

$$\omega_i \xi_k + \lambda_i e_i^t c_k = \lambda_k e_i^t c_k$$

$$\omega_i \xi_k + \lambda_i c_{ik} = \lambda_k c_{ik}$$

which results in

$$c_{ik} = \frac{\omega_i \xi_k}{\lambda_k - \lambda_i}, \quad i \neq k, \quad i = 1, 2, \ldots, n. \tag{3.6}$$

The value of c_{kk} depends on the normalization chosen for x_k, but this is inconsequential. Differentiating (3.2) a second time by taking the derivative of (3.4) and premultiplying by y_k^* gives

$$\begin{aligned}
2e_1 f^t \dot{x}_k + (H-(1-t)e_1 f^t)\ddot{x}_k &= \ddot{\lambda}_k x_k + \dot{\lambda}_k \dot{x}_k + \dot{\lambda}_k \dot{x}_k + \lambda_k \ddot{x}_k \\
2y_k^* e_1 f^t \dot{x}_k + \lambda_k y_k^* \ddot{x}_k &= \ddot{\lambda}_k y_k^* x_k + 2\dot{\lambda}_k y_k^* \dot{x}_k + \lambda_k y_k^* \ddot{x}_k \\
\ddot{\lambda}_k &= 2y_k^* e_1 f^t \dot{x}_k - 2\dot{\lambda}_k y_k^* \dot{x}_k \\
&= 2\omega_k f^t \sum_{i=1}^{n} x_i c_{ik} - 2\omega_k \xi_k y_k^* \sum_{i=1}^{n} x_i c_{ik} \\
&= 2\omega_k (\xi_k c_{kk} + \sum_{i\neq k} \xi_i c_{ik}) - 2\omega_k \xi_k c_{kk} \\
&= 2\omega_k \sum_{i\neq k} \xi_i c_{ik} \\
&= 2\omega_k \sum_{i\neq k} \frac{\xi_i \omega_i \xi_k}{\lambda_k - \lambda_i} \\
&= 2\omega_k \xi_k \sum_{i\neq k} \frac{\omega_i \xi_i}{\lambda_k - \lambda_i}.
\end{aligned}$$

So in a neighborhood of t=0,

$$\ddot{\lambda}_k(t) = 2\dot{\lambda}_k(t) \sum_{i\neq k} \frac{\dot{\lambda}_i(t)}{\lambda_k(t) - \lambda_i(t)}, \tag{3.7}$$

and we see that $\ddot{\lambda}_k$ is available for $O(n)$ operations and that higher order derivatives are possible.

The homotopy step can be summarized as follows. Given a set of pairwise distinct numbers closed under complex conjugation, $\Omega = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$, compute a vector $f \in \mathbb{R}^n$

such that $\lambda(H - e_1 f^t) = \Omega$. Then compute the quantities $\dot{\lambda}_k$ and $\ddot{\lambda}_k$, and define a new approximation to $\lambda(H)$ as in (3.3).

If we view $\ddot{\lambda}$ as an error estimator for the Euler prediction, $\lambda_k(h_k) = \lambda_k(0) + \dot{\lambda}_k h_k$, then we can choose h_k adaptively and independently for each eigenvalue to satisfy some error criterion $\frac{h_k^2}{2}|\ddot{\lambda}| < \tau$. Later, we will use the h_k to detect troublesome eigenpaths, which will be recomputed via an Arnoldi minimization procedure.

3.1 Computing the Feedback

Given an unreduced Hessenberg matrix, H, and a set $\Omega = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ of n numbers closed under complex conjugation, we want to compute $f \in \mathbb{R}^n$ such that $\lambda(H - e_1 f^t) = \Omega$. This is a special case of the *eigenvalue assignment/pole placement problem* addressed in [5], [9], [17], [18], and [20]. More generally, given $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times p}$, and Ω as defined above, the eigenvalue assignment problem is to find $F \in \mathbb{R}^{p\times n}$ such that $\lambda(A - BF) = \Omega$. This problem has a solution for any Ω if and only if the system $\dot{x} = Ax + Bu$ is controllable. This time invariant system (A constant) is controllable if and only if the controllability matrix

$$C = [B, AB, \ldots, A^{n-1}B] \tag{3.8}$$

has rank n (see [10]).

In our case, we take $A = H$ and $B = e_1$. The system (H, e_1) is controllable if and only if $[e_1, He_1, \ldots, H^{n-1}e_1]$ has rank n, which is guaranteed if H is unreduced. Thus, the problem has a unique solution, $F = f^t$. All of the numerically attractive methods for this problem, like that of Miminis and Paige [18], are based on a preliminary reduction of (A, b) to the

controller-Hessenberg form

$$P^t b = \|b\| e_1, \quad P^t A P = H, \tag{3.9}$$

where $P \in \mathbb{R}^{n \times n}$ is orthogonal. Controllablility can be determined (with all of the caveats associated with any rank determination problem) by inspecting the subdiagonal elements of H.

In 1972, Ackermann [1] gave an explicit form for the single-input feedback,

$$f^t = e_n^t C^{-1} \phi(A), \tag{3.10}$$

where $\phi(x) = \prod_{i=1}^{n} (x - \lambda_i)$, λ_i are the desired eigenvalues, and C is the controllability matrix given in (3.8). In the controller-Hessenberg case, C is upper triangular and we have

$$f^t = \alpha e_n^t \phi(H), \tag{3.11}$$

where $\alpha = (\|b\| \prod_{i=1}^{n-1} h_{i+1,i})^{-1}$. Arnold and Datta [2] have shown that a whole class of methods used to compute f that appeared in the 1980's are equivalent to the QR algorithm with shifts from Ω to compute the QR factorization of $\phi(H)$. Such methods are the only ones that have been proven backward stable, and when implemented with implicit double shifts and deflation, require about $5n^3$ flops.

In order to keep our flop count down, we will use a different type of method, due to Datta [5]. Unfortunately, the method is not stable, for the recursion is susceptible to digit cancellation. However, an inexpensive estimate of the backward error is available as the

Algorithm 3.1: Recursion for the Single-Input Feedback [5]

Input: $H \in \mathbb{R}^{n\times n}$ unreduced Hessenberg

$\Omega = \{\lambda_1, \ldots, \lambda_n\}$ closed under complex conjugation

Output: $L \in \mathbb{C}^{n\times n}$, $\|\hat{l}_i\| \in \mathbb{R}$, and $f \in \mathbb{R}^n$ such that $\lambda(H - e_1 f^t) = \Omega$

$l_1 = e_n$

for $i = 1, 2, \ldots, n-1$

$\hat{l}_{i+1} = (H^t - \lambda_i I) l_i$

$l_{i+1} = \dfrac{\hat{l}_{i+1}}{\|\hat{l}_{i+1}\|}$

end

$L = [l_1, l_2, \ldots, l_n]$

$f = \dfrac{(H^t - \lambda_n I) l_n}{l_{1,n}}$

recursion unfolds, so the method is reliable in the sense that it knows when things have gone bad as shown by Arnold in [3]. For our purposes, the beauty of Datta's method comes in its ease and efficiency. It requires only $O(\frac{1}{3}n^3)$ flops. Furthermore, it provides at no extra cost a bidiagonalization of our closed loop matrix $H - e_1 f^t$. Define G_c as the bidiagonal matrix

$$G_c = \begin{bmatrix} \lambda_1 & \|\hat{l}_2\| & & & \\ & \lambda_2 & \|\hat{l}_3\| & & \\ & & \lambda_3 & \ddots & \\ & & & \ddots & \|\hat{l}_n\| \\ & & & & \lambda_n \end{bmatrix}.$$

where the $\|\hat{l}_i\|$ are output from Algorithm 3.1. Then

$$L^t(H - e_1 f^t) = G_c L^t \tag{3.12}$$

with L output from Algorithm 3.1. The fact that H is unreduced Hessenberg ensures that L is lower right triangular and nonsingular. This will be of great value later as we calculate the left and right eigenvectors of $H - e_1 f^t$ that will be needed to follow homotopy paths.

Since H is unreduced, $H - e_1 f^t$ is also unreduced, and diagonalizing $H - e_1 f^t$ requires pairwise distinct λ_j. In that case, if we define $V = [v_{ij}]$ to be unit upper triangular with

$$v_{ij} = \prod_{k=i+1}^{j} \frac{\|\hat{l}_k\|}{(\lambda_i - \lambda_k)} \tag{3.13}$$

for $i < j$, then it is not difficult to show that V diagonalizes G_c as

$$VG_cV^{-1} = \Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n).$$

This method not only provides the feedback, f, but also a triangular factorization of a matrix of left eigenvectors $Y = VL^t$. Later, we will be concerned with the quantities f, $\omega = Ye_1$, and $\xi^t = f^t Y^{-1}$, all of which can be computed for about $\frac{1}{3}n^3$ flops.

There are two main difficulties with computing f and Y in this way. One is fundamental to the overall method: a multiple eigenvalue precludes the existence of a well-defined tangent to that eigenpath as V has a singularity if $\lambda_i = \lambda_j$. The other is simply efficiency: complex arithmetic is about 4 times as expensive as real arithmetic. As might be expected, we can fix the latter problem via a 2×2 block diagonalization. This requires tweaking Algorithm 3.1. From here on, let us assume that the current eigenvalue approximations, λ_i, are ordered so complex conjugate pairs appear together at the front of the list. We further define $\mu_i = \text{Re}(\lambda_i)$ for all i, $\nu_i = \text{Im}(\lambda_i)$ for odd i, and $\nu_i = 0$ for even i.

Algorithm 3.2: Real Recursion for the Single-Input Feedback

Input: $H \in \mathbb{R}^{n\times n}$ unreduced Hessenberg

$\Omega = \{\lambda_1, \ldots, \lambda_n\}$ closed under complex conjugation

Output: $L \in \mathbb{R}^{n\times n}$, $\|\hat{l}_i\| \in \mathbb{R}$, and $f \in \mathbb{R}^n$ such that $\lambda(H - e_1 f^t) = \Omega$

Order λ_i so complex conjugate pairs appear together at the front of the list.

$l_0 = 0$

$l_1 = e_n$

$\mu_i = \mathrm{Re}(\lambda_i)$

$$\nu_i = \begin{cases} \mathrm{Im}(\lambda_i) & : \text{odd } i \\ 0 & : \text{even } i \end{cases}$$

for $i = 1, 2, \ldots, n-1$

$$\hat{l}_{i+1} = (H^t - \mu_i I) l_i + \frac{{\nu_i}^2 \, l_{i-1}}{\|\hat{l}_i\|}$$

$$l_{i+1} = \frac{\hat{l}_{i+1}}{\|\hat{l}_{i+1}\|}$$

end

$L = [l_1, l_2, \ldots, l_n]$

$$f = \frac{(H^t - \mu_n I) l_n + \dfrac{{\nu_n}^2 \, l_{n-1}}{\|\hat{l}_n\|}}{l_{1,n}}$$

The matrix L from Algorithm 3.2 is still lower right triangular and nonsingular, but now it provides a block-bidiagonalization of $H - e_1 f^t$. We have

$$L^t (H - e_1 f^t) = G L^t$$

where

$$G = \begin{bmatrix} G_{1,1} & G_{1,2} & & & \\ & G_{2,2} & G_{2,3} & & \\ & & G_{3,3} & \ddots & \\ & & & \ddots & G_{p-1,p} \\ & & & & G_{p,p} \end{bmatrix},$$

$$G_{i,i} = \begin{bmatrix} \mu_{2i-1} & \|\hat{l}_{2i}\| \\ \dfrac{-\nu_{2i-1}{}^2}{\|\hat{l}_{2i}\|} & \mu_{2i} \end{bmatrix}, \text{ and}$$

$$G_{i,i+1} = \begin{bmatrix} 0 & 0 \\ \|\hat{l}_{2i+1}\| & 0 \end{bmatrix}.$$

Note that $G_{i,i}$ has eigenvalues λ_{2i-1} and λ_{2i}. In the case where n is odd,

$$G_{p,p} = [\mu_n] \in \mathbb{R}^{1\times 1}$$

and

$$G_{p-1,p} = \begin{bmatrix} 0 \\ \|\hat{l}_n\| \end{bmatrix} \in \mathbb{R}^{2\times 1}.$$

3.2 Calculating the Left Eigenvectors

We now will find a factorization for a matrix of left eigenvectors $Y = SVL^t$, where L and V are triangular and S is block diagonal. The first factor, L, is a byproduct of the feedback

calculation in Algorithm 3.2. We get V by block-diagonalizing G. That is, we would like to find a nonsingular matrix V such that

$$VG = DV \tag{3.14}$$

where D is block-diagonal with the same main diagonal blocks as G. That is,

$$D = \text{diag}(G_{1,1}, G_{2,2}, \ldots, G_{p,p}). \tag{3.15}$$

Proposition 3.2 *Define $V = [V_{ij}]$, where V_{ij} is 2×2 (excepting, for odd n, the last row and column). If $VG = DV$ then*

(a) For $j < i \leq \lfloor \frac{n}{2} \rfloor$, $V_{i,j} = 0$.

(b) For $i = j \leq \lfloor \frac{n}{2} \rfloor$ and $\lambda_{2i-1} = \bar{\lambda}_{2i}$, $V_{i,j} = \alpha I + \beta \begin{bmatrix} 0 & \frac{-\|\hat{l}_{2i}\|^2}{v_{2i-1}^2} \\ 1 & 0 \end{bmatrix}$ with $\alpha, \beta \in \mathbb{R}$.

(c) For $i = j \leq \lfloor \frac{n}{2} \rfloor$ and $\lambda_{2i-1}, \lambda_{2i} \in \mathbb{R}$, $V_{i,j} = \alpha I + \beta \begin{bmatrix} \frac{\mu_{2i-1} - \mu_{2i}}{\|\hat{l}_{2i}\|} & 1 \\ 0 & 0 \end{bmatrix}$ with $\alpha, \beta \in \mathbb{R}$.

(d) For $i < j \leq \lfloor \frac{n}{2} \rfloor$, $V_{i,j} = [\frac{1}{\|\hat{l}_{2j}\|}(G_i - \mu_{2j} I)v,\ v]$ where v solves
$\frac{1}{\|\hat{l}_{2j}\|}(G_i - \lambda_{2j-1})(G_i - \lambda_{2j})v = \|\hat{l}_{2j-1}\| V_{i,j-1} e_2$.

(e) For odd n, $e_n^t V = \alpha e_n^t$ for $\alpha \in \mathbb{R}$.

(f) For odd n and $i \leq \lfloor \frac{n}{2} \rfloor$, the 2×1 blocks of the last column of V solve
$$\begin{bmatrix} \mu_{2i-1} - \mu_n & \|\hat{l}_{2i}\| \\ \frac{-v_{2i-1}^2}{\|\hat{l}_{2i}\|} & \mu_{2i} - \mu_n \end{bmatrix} \begin{bmatrix} v_{2i-1,n} \\ v_{2i,n} \end{bmatrix} = \|\hat{l}_n\| \begin{bmatrix} v_{2i-1,n-1} \\ v_{2i,n-1} \end{bmatrix}.$$

Proof: For aesthetics we denote a block on the diagonal with a single subscript (i.e. G_i is the block G_{ii}). To show (a) we proceed by induction on j. For $j = 1$, looking at the (i, j) block of the matrix equation (3.14) and recalling $D_i = G_i$ gives

$$V_{i,1}G_1 = G_iV_{i,1}$$

or

$$G_iV_{i,1} - V_{i,1}G_1 = 0 \tag{3.16}$$

Equation (3.16) is a Sylvester equation and has a unique solution provided G_i and G_1 have no common eigenvalues. This can be arranged provided the multiplicity of real eigenvalues is no more than 2 and that all complex eigenvalues are simple. Recall we are assuming that all eigenvalues are simple; thus, we have a unique solution. Clearly $V_{i,1} = 0$ solves (3.16) and therefore is the solution. Now assuming that (a) holds for $j = p$ then the $(i, p+1)$ block of (3.14) gives

$$G_iV_{i,p+1} - V_{i,p+1}G_{p+1} = 0$$

and we see again that $V_{i,p+1} = 0$ is the unique solution and (a) is shown.

Now for (b) and (c) the (i, i) block of (3.14) is

$$V_iG_i = G_iV_i \tag{3.17}$$

which clearly does not have a unique solution as both $V_i = 0$ and $V_i = I$ are solutions. If we

view (3.17) as

$$\begin{bmatrix} w & x \\ y & z \end{bmatrix} \begin{bmatrix} \mu_{2i-1} & \|\hat{l}_{2i}\| \\ \dfrac{-\nu_{2i-1}^2}{\|\hat{l}_{2i}\|} & \mu_{2i} \end{bmatrix} = \begin{bmatrix} \mu_{2i-1} & \|\hat{l}_{2i}\| \\ \dfrac{-\nu_{2i-1}^2}{\|\hat{l}_{2i}\|} & \mu_{2i} \end{bmatrix} \begin{bmatrix} w & x \\ y & z \end{bmatrix} \tag{3.18}$$

we get the homogeneous system of equations

$$\begin{bmatrix} \|\hat{l}_{2i}\| & (\mu_{2i} - \mu_{2i-1}) & 0 & -\|\hat{l}_{2i}\| \\ \dfrac{\nu_{2i-1}^2}{\|\hat{l}_{2i}\|} & 0 & (\mu_{2i-1} - \mu_{2i}) & \dfrac{-\nu_{2i-1}^2}{\|\hat{l}_{2i}\|} \\ 0 & \dfrac{-\nu_{2i-1}^2}{\|\hat{l}_{2i}\|} & -\|\hat{l}_{2i}\| & 0 \\ 0 & \dfrac{\nu_{2i-1}^2}{\|\hat{l}_{2i}\|} & \|\hat{l}_{2i}\| & 0 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{3.19}$$

If λ_{2i-1} and λ_{2i} are a complex conjugate pair then the coefficient matrix in (3.19) reduces to

$$\begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & \nu_{2i-1}^2 & \|\hat{l}_{2i}\|^2 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.20}$$

from which it is easy to see that

$$V_i = \alpha I + \beta \begin{bmatrix} 0 & \dfrac{-\|\hat{l}_{2i}\|^2}{\nu_{2i-1}^2} \\ 1 & 0 \end{bmatrix} \tag{3.21}$$

for $\alpha, \beta \in \mathbb{R}$. This gives us (b). If, however, λ_{2i-1} and λ_{2i} are real eigenvalues then the

coefficient matrix in (3.19) reduces to

$$\begin{bmatrix} \|\hat{l}_{2i}\| & (\mu_{2i}-\mu_{2i-1}) & 0 & -\|\hat{l}_{2i}\| \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.22}$$

showing that

$$V_i = \alpha I + \beta \begin{bmatrix} \dfrac{\mu_{2i-1}-\mu_{2i}}{\|\hat{l}_{2i}\|} & 1 \\ 0 & 0 \end{bmatrix} \tag{3.23}$$

for $\alpha, \beta \in \mathbb{R}$, thus (c) is proven.

We now develop the solution for V_{ij} with $i < j$. Considering the (i, j) block from matrix equation (3.14) and again recalling that $D_i = G_i$ yields

$$V_{i,j-1}G_{j-1,j} + V_{i,j}G_j = G_i V_{i,j}$$

$$G_i V_{i,j} - V_{i,j}G_j = V_{i,j-1} \begin{bmatrix} 0 & 0 \\ \|\hat{l}_{2j-1}\| & 0 \end{bmatrix}$$

$$G_i V_{i,j} - V_{i,j}G_j = \|\hat{l}_{2j-1}\| V_{i,j-1} e_2 e_1^t \tag{3.24}$$

Again, we see that (3.24) is a Sylvester equation and has a unique solution under our assumption that all eigenvalues are simple. Let V_{i,j_1} and V_{i,j_2} denote the first and second

columns of $V_{i,j}$ respectively, then the second column of (3.24) reveals

$$G_i V_{i,j} e_2 - V_{i,j} G_j e_2 = \|\hat{l}_{2j-1}\| V_{i,j-1} e_2 e_1^t e_2$$

$$G_i V_{i,j_2} - V_{i,j} \begin{bmatrix} \|\hat{l}_{2j}\| \\ \mu_{2j} \end{bmatrix} = 0$$

$$G_i V_{i,j_2} - \|\hat{l}_{2j}\| V_{i,j_1} - \mu_{2j} V_{i,j_2} = 0$$

$$\frac{1}{\|\hat{l}_{2j}\|}(G_i - \mu_{2j} I) V_{i,j_2} = V_{i,j_1}. \tag{3.25}$$

So we now see for $i < j$ that

$$V_{i,j} = \begin{bmatrix} \frac{1}{\|\hat{l}_{2j}\|}(G_i - \mu_{2j} I) V_{i,j_2}, & V_{i,j_2} \end{bmatrix} \tag{3.26}$$

which leaves us only to find V_{i,j_2}. To do this, we now look at the first column of (3.24), which gives

$$G_i V_{i,j} e_1 - V_{i,j} G_j e_1 = \|\hat{l}_{2j-1}\| V_{i,j-1} e_2 e_1^t e_1$$

$$G_i V_{i,j_1} - V_{i,j} \begin{bmatrix} \mu_{2j-1} \\ \frac{-\nu_{2j-1}^2}{\|\hat{l}_{2j}\|} \end{bmatrix} = \|\hat{l}_{2j-1}\| V_{i,j-1} e_2$$

$$G_i V_{i,j_1} - \mu_{2j-1} V_{i,j_1} + \frac{-\nu_{2j-1}^2}{\|\hat{l}_{2j}\|} V_{i,j_2} = \|\hat{l}_{2j-1}\| V_{i,j-1} e_2$$

$$(G_i - \mu_{2j-1} I) V_{i,j_1} + \frac{-\nu_{2j-1}^2}{\|\hat{l}_{2j}\|} V_{i,j_2} = \|\hat{l}_{2j-1}\| V_{i,j-1} e_2$$

$$\frac{1}{\|\hat{l}_{2j}\|} (G_i - \mu_{2j-1} I)((G_i - \mu_{2j} I) V_{i,j_2} + \frac{-\nu_{2j-1}^2}{\|\hat{l}_{2j}\|} V_{i,j_2} = \|\hat{l}_{2j-1}\| V_{i,j-1} e_2$$

$$\frac{1}{\|\hat{l}_{2j}\|} \left[G_i^2 - (\mu_{2j-1} + \mu_{2j}) G_i + (\mu_{2j-1}\mu_{2j} + \nu_{2j-1}^2) I \right] V_{i,j_2} = \|\hat{l}_{2j-1}\| V_{i,j-1} e_2$$

$$\frac{1}{\|\hat{l}_{2j}\|} (G_i - \lambda_{2j-1} I)(G_i - \lambda_{2j} I) V_{i,j_2} = \|\hat{l}_{2j-1}\| V_{i,j-1} e_2 \qquad (3.27)$$

and proves (d).

For (e) we first consider the $(1,n)$ and $(2,n)$ elements (not blocks) of (3.14). This yields the homogenous system of equations

$$\begin{bmatrix} \mu_1 - \mu_n & -\frac{\nu_1^2}{\|\hat{l}_2\|} \\ \|\hat{l}_2\| & (\mu_2 - \mu_n) \end{bmatrix} \begin{bmatrix} v_{n,1} \\ v_{n,2} \end{bmatrix} = 0.$$

There are three scenarios: 1) $\mu_n = \mu_1 = \mu_2$ and $\nu_1 \neq 0$, 2) $\mu_1 = \mu_2 \neq \mu_n$ and $\nu_1 \neq 0$,

and 3) $\mu_1 \neq \mu_n \neq \mu_2$ and $\nu_1 = 0$. In any case, the coefficient matrix can be shown to be nonsingular. Thus, $v_{n,1} = v_{n,2} = 0$. Proceeding sequentially along the 1×2 blocks on the last row of V will produce a similar set of equations with the same result and gives us $v_{n,1} = v_{n,2} = \cdots = v_{n,n-1} = 0$. The (n,n) element of (3.14) gives the equation

$$v_{nn}\mu_n = \mu_n v_{nn}$$

where any real v_{nn} is a solution and we have (e).

Finally, for (f) consider the $(2i-1,n)$ and $(2i,n)$ elements (not blocks) in (3.14). This produces the two equations

$$\begin{aligned} \|\hat{l}_n\| v_{2i-1,n-1} + \mu_n v_{2i-1,n} &= \mu_{2i-1} v_{2i-1,n} + \|\hat{l}_{2i}\| v_{2i,n} \\ \|\hat{l}_n\| v_{2i,n-1} + \mu_n v_{2i,n} &= \frac{-\nu_{2i-1}^2}{\|\hat{l}_{2i}\|} v_{2i-1,n} + \mu_{2i} v_{2i,n} \end{aligned}$$

from which we get (f). □

If in (b) in the above proposition we choose $\beta = 0$, then V is upper triangular. Further, if we choose $\alpha \neq 0$ in (b), (c), and (e) then V is nonsingular. For (b), (c), and (e) we have taken $\alpha = 1$ and $\beta = 0$.

Let us now take a moment to review where we are. Using only real arithmetic, we have calculated the feedback, f, and triangular, nonsingular matrices, L and V, such that $VL^t(H - e_1 f^t) = DL^t V$. The fact that V and L^t are triangular somewhat ameliorates the difficulties associated with the ill-conditioning they typically have, but difficulties remain. We have postponed complex arithmetic as long as possible, but we will need it to diago-

nalize D. This will be done with a 2×2 block diagonal matrix S, which is a matrix of left eigenvectors of D. That is, $SD = \Lambda S$. When the kth diagonal block of D, D_k, has complex eigenvalues,

$$S_k = \begin{bmatrix} \dfrac{\mathrm{Im}(\lambda_{2k-1})i}{\|\hat{l}_{2k}\|} & 1 \\ \dfrac{\mathrm{Im}(\lambda_{2k})i}{\|\hat{l}_{2k}\|} & 1 \end{bmatrix}. \tag{3.28}$$

Otherwise, when D_k has real eigenvalues,

$$S_k = \begin{bmatrix} \dfrac{(\lambda_{2k-1} - \lambda_{2k})}{\|\hat{l}_{2k}\|} & 1 \\ 0 & 1 \end{bmatrix}. \tag{3.29}$$

If n is odd then the last block of S is 1×1 and

$$S_p = [1] \tag{3.30}$$

with $p = \lceil \frac{n}{2} \rceil$. Of course the rows of S may be scaled as needed.

We now have an eigendecompostion of $H - e_1 f^t$, specifically

$$SVL^t(H - e_1 f^t) = \Lambda SVL^t. \tag{3.31}$$

Furthermore, $\omega = SVL^t e_1$ and $\xi^t = f^t(SVL^t)^{-1}$. Of the $O(\frac{1}{3}n^3)$ flops needed for f, ω and ξ, only $O(n)$ are complex. We now have the necessary quantities for the updates in (3.3).

4 Handling Homotopy Path Bifurcations

All of the homotopy path following methods must address the situation where a complex pair coalesces, becomes a multiple eigenvalue, and splits into two distinct eigenvalues or vice versa ([13], [14], [15]). These bifurcations must be accommodated unless the initial conditions preclude their occurance. Subspace iteration, that is, inverse iteration on a subspace of dimension greater than one, is one approach, and may make it possible to avoid complex floating point arithmetic.

Even if only one inverse iteration were necessary for each correction, then about $2n^2$ flops would be required per eigenvalue per time step. In order to compete with the QR algorithm on a serial machine, all paths must be traversed and full accuracy attained in an average of about 5 real inverse iterations per eigenvalue. This has been achieved, at least for randomly generated matrices and a residual tolerance of about 10^{-9}, by Li, et al.[13]

We will handle any paths containing bifurcations by computing a new homotopy that does not contain the troublesome paths. At each iteration we replace m homotopy predictions with predictions that allow for a real-to-complex or complex-to-real interchange. One needs to determine the quantity m, which m predictions will be replaced, and what the replacements should be.

We suggest choosing the m replacements to minimize $\|f\|_2$ for the next homotopy. We choose the m estimates with largest $|\ddot{\lambda}_k|$ to be replaced in favor of the $\|f\|$ minimizers. One justification for this rests in seeing $\frac{h_k^2}{2}\ddot{\lambda}_k$ as an error estimate for the Euler prediction. Thus, we are replacing the predictions in which we are least confident. Another justification is simply that the curvature of the eigenpath as measured by $|\ddot{\lambda}_k|$ is high near a singularity.

How to choose the quantity m seems to be an interesting question. Later we will count flops and provide a heuristic for an optimal m, but for now we note that $m < 2$ would not permit real/complex exchanges, while $m = n$ would solve the eigenvalue problem for H. It is also clear that the minimization will replace homotopy paths with high curvature, even if no real/complex exchange takes place.

4.1 Feedback Minimization

Applying m steps of the Arnoldi process ([7], [21]) for a matrix $A \in \mathbb{R}^{n\times n}$ and starting input vector b provides an orthogonal matrix, $Q_m \in \mathbb{R}^{n\times m}$, a Hessenberg matrix, $W_m \in \mathbb{R}^{m\times m}$, and a vector r_m such that

$$AQ_m = Q_m W_m + r_m e_m^t.$$

Also, $Q_m^t r_m = 0$ and $Q_m^t A Q_m = W_m$. The columns of Q_m are an orthonormal basis for the Krylov subspace,

$$\mathscr{K}_m(A,b) = \text{span}\{b, Ab, \ldots, A^{m-1}b\}.$$

If at some step $k \leq m$ in the Arnoldi iteration $r_k = 0$, then the process must terminate, and $\mathscr{K}_k(A,b)$ is an invariant subspace for A and $\lambda(W_k) \subseteq \lambda(A)$. If the process does not terminate and $r_m \neq 0$, then we can still glean some information about $\lambda(A)$ from $\lambda(W_m)$. Let $\mathscr{P}_m$ be the set of monic polynomials of degree no more than m, and let p_0 be the characteristic polynomial of W_m, then p_0 satisfies

$$\min_{p\in\mathscr{P}_m} \|p(A)b\|_2.$$

Now assume that after stepping forward along each of the homotopy paths, we can partition our current approximation to $\lambda(H)$ into a subset $\Omega_B = \{\lambda_1, \lambda_2, \ldots, \lambda_{n-m}\}$ of "best" approximations and a subset $\Omega_W = \{\lambda_{n-m+1}, \lambda_{n-m+2}, \ldots, \lambda_n\}$ of "worst" approximations where both Ω_B and Ω_W are closed under complex conjugation. We apply the Arnoldi process to the matrix $A = H^t$, with the input vector b, given by

$$b^t = e_n^t \phi_B(H) \tag{4.1}$$

where

$$\phi_B(x) = \prod_{i=1}^{n-m} (x - \lambda_i)$$

with $\lambda_i \in \Omega_B$. This generates an $m \times m$ Hessenberg matrix with characteristic polynomial

$$q_0(x) = \prod_{i=1}^{m} (x - \gamma_i)$$

which optimizes

$$\min_{q \in \mathscr{P}_m} \|e_n^t \phi_B(H) q(H)\|_2.$$

We then replace Ω_W with the roots of q_0. That is, we set

$$\lambda_{n-m+j} \longleftarrow \gamma_j, \quad j = 1, 2, \ldots, m$$

Notice that if ϕ_B divides the characteristic polynomial of H, then the minimization property of the Arnoldi process implies that $q_0 \phi_B$ is the characteristic polynomial of H, or if the process breaks down, $q_0 \phi_B$ will be its minimal polynomial. This means the Arnoldi

process can be viewed as a deflation process, for if the roots of ϕ_B are eigenvalues of H, m Arnoldi steps with input vector $b = \phi_B(H^t)e_n$ generates an $m \times m$ Hessenberg matrix whose eigenvalues are exactly the remaining eigenvalues of H.

If we use the roots of ϕ_B and q_0 as the initial conditions for a new homotopy, then the feedback vector for this new homotopy will be $f^t = \alpha e_n^t \phi_B(H) q(H)$, which is precisely the quantity minimized by the Arnoldi process. Furthermore, if the homotopy paths containing bifurcations are included in Ω_W, then the Arnoldi minimization allows for the real/complex interchanges.

5 The Restarted Homotopy Method

We now present the algorithm for the restarted homotopy method.

Algorithm 5.1: The Restarted Homotopy Method

Given an unreduced Hessenberg matrix, $H \in \mathbb{R}^{n\times n}$, an approximation $\Omega = \{\lambda_1, \ldots, \lambda_n\}$ to $\lambda(H)$, a homotopy error tolerance, τ, and an eigenvalue convergence tolerance, ε, compute a new approximation to $\lambda(H)$.

1. Using Algorithm 3.2, proposition 3.2 with $\alpha = 1$ and $\beta = 0$, (3.28), (3.29), and (3.30) compute $f \in \mathbb{R}^n$, $L, V \in \mathbb{R}^{n\times n}$ and $S \in \mathbb{C}^{n\times n}$, such that $SVL^t(H - e_1 f^t) = \Lambda SVL^t$, where V and L are triangular, S is block diagonal with 2×2 or 1×1 blocks, and $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$.

2. Compute $\omega = SVL^t e_1$ and ξ satisfying $\xi^t SVL^t = f^t$.

3. For each $k = 1, 2, \ldots, n$ compute $\dot{\lambda}_k$ and $\ddot{\lambda}_k$ by (3.5) and (3.7), respectively. If all $|\ddot{\lambda}_k| \leq 2\varepsilon$, then update as in (3.3) and stop.

4. For each $k = 1, 2, \ldots, n$ compute $h_k = \sqrt{2\tau/\ddot{\lambda}_k}$. Select a nonnegative integer m, and update as in (3.3) the $n - m$ eigenvalues with the largest h.

5. Let ϕ_B be the monic polynomial whose roots are the updated eigenvalues from step 4. Compute the vector $b^t = e_n^t \phi_B(H)$.

6. Run m steps of Arnoldi with operator H^t and input vector b. Replace the m eigenvalues not yet updated with the Arnoldi eigenvalues. Go to step 1.

Steps 1 through 4 of each iteration requires $\frac{1}{3}n^3$ flops combined and step 5 requires $\frac{1}{3}(n-m)^3$ flops. The Arnoldi corrections from step 6 require about $mn^2 + 4m^2 n + 10m^3$

flops assuming reorthogonalization at each Arnoldi step and that the QR algorithm is used for computing the eigenvalues of the $m \times m$ Hessenberg matrix. Storage requirements include: $n^2/2$ real words for each of H, L and V, another $nm + \frac{m^2}{2}$ for the Arnoldi minimization, plus $O(n)$ for f, S, ω, ξ, Ω, $\dot{\lambda}$, $\ddot{\lambda}$, and h, giving a total of about $\frac{3}{2}n^2 + (m+10)n + \frac{m^2}{2}$ real words.

A homotopy step tolerance, τ, that is too small forces smaller time steps and can make the homotopy updates ineffective. We are less concerned with τ being too large as the m homotopy corrections with largest $|\ddot{\lambda}|$ are dropped in favor of the Arnoldi eigenvalues. We have chosen $\tau = \sqrt{\frac{n}{2\|H\|_F^2}}$. The eigenvalue convergence tolerance, ε, is currently set at $\sqrt{\mathbf{u}}$, where $\mathbf{u}$ is the machine precision. The rationale here is that the homotopy corrections eventually converge cubically, but $\varepsilon = \mathbf{u}^{1/3}$ may be too aggressive. The only other free parameter is m, the size of the Arnoldi correction. Currently, we have only add hoc procedures for this selection. When f is large compared to $\|H\|$, m can be as large as $n/4$. But when f is small (near convergence) the homotopy predictions are extremely effective, so usually $m = 0$ there. In section 8 we report more details.

6 Practical Issues

On one hand we have a highly parallel method that runs up to 2 times faster than the QR method on serial machines. On the other hand we are proposing a method based on diagonalizing intermediate matrices to drive a performance measure, $\|f\|$, to zero. This idea has severe limitations. When f is near zero, it is necessarily computed with severe cancellation. While this in itself is not always detrimental, it is a fundamental concern

for us, for the quantity ξ, which is used in computing the homotopy updates, is given by $Y^t\xi = f$, where Y is a matrix of left eigenvectors. If, for example, $\bar{f} = f - r$ is a computed version of the exact feedback defined by $\{\lambda_1, \ldots, \lambda_n\}$, and if Y is exactly a matrix of left eigenvectors for $H - e_1 f^t$, then

$$\begin{aligned} Y(H - e_1 f^t) &= \Lambda Y \\ Y(H - e_1 f^t) + Y e_1 r^t &= \Lambda Y + Y e_1 r^t \\ Y(H - e_1 (f - r)^t) &= \Lambda Y + Y e_1 r^t Y^{-1} Y \\ Y(H - e_1 \bar{f}^t) &= (\Lambda + R) Y \end{aligned} \tag{6.1}$$

where $R = Y e_1 r^t Y^{-1}$. When $\|r\|$ is on the order of $\|\bar{f}\|$, as we expect near convergence, our first order homotopy update term, $\dot{\lambda}$, might be mostly error as

$$|\dot{\lambda}_k| = |\omega_k \xi_k| = |e_k^t Y e_1 \bar{f}^t Y^{-1} e_k| \approx |e_k^t Y e_1 r^t Y^{-1} e_k|.$$

If the eigenproblem for $H - e_1 f^t$ is ill-conditioned, then even a relatively small r could lead to relatively large R, thus slowing down convergence. Of course we do expect slower convergence if the eigenproblem for H is ill-conditioned, since $H - e_1 f^t \to H$ as $f \to 0$.

We have introduced the subproblem of computing f or at least $f^t Y^{-1}$. The sensitivity of which will have important consequences for overall stability/efficiency. To help understand the issues, we present a sensitivity analysis.

Theorem 6.1 *If $H \in \mathbb{R}^{n\times n}$ is unreduced Hessenberg and*

$$\lambda(H - e_1 f^t) = \{\lambda_1, \lambda_2, \ldots, \lambda_n\},$$

then there exists a matrix of left eigenvectors, Y, of $H - e_1 f^t$ such that

$$Ye_1 = -[1, 1, \ldots, 1]^t$$

and

$$y_{i,j} = \frac{\partial f_j}{\partial \lambda_i}.$$

Proof: Let H be unreduced Hessenberg and $\lambda(H - e_1 f^t) = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$. Recall from (3.11) that $f^t = \alpha e_n^t \phi(H)$ where $\phi(x) = \prod_{j=1}^{n}(x - \lambda_i)$ and, in this case,

$$\alpha = \left(\prod_{j=1}^{n-1} h_{j+1,j}\right)^{-1}.$$

Let $\phi_i(x) = \prod_{j\neq i}(x - \lambda_j)$. It is not difficult to verify that $e_n^t \phi_i(H) e_1 = \alpha^{-1}$, which is nonzero since H is unreduced. With all of this in mind,

$$\begin{aligned}
\alpha e_n^t \phi_i(H)(H - e_1 f^t - \lambda_i I) &= \alpha e_n^t \phi_i(H)(H - \lambda_I) - \alpha e_n^t \phi_i(H) e_1 f^t \\
&= \alpha e_n^t \phi(H) - \alpha\alpha^{-1} f^t \\
&= f^t - f^t \\
&= 0
\end{aligned}$$

shows that $(\alpha e_n^t \phi_i(H), \lambda_i)$ is a left eigenpair of $H - e_1 f^t$ and $\alpha e_n^t \phi_i(H) e_1 = 1$. Letting $y_i^* = -\alpha e_n^t \phi_i(H)$ and $Y = [y_1, y_2, \ldots, y_n]^*$ yields

$$Y(H - e_1 f^t)Y^{-1} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

with $Ye_1 = -[1, 1, \ldots, 1]^t$.

Now taking the derivative of $f^t = \alpha e_n^t \phi(H)$ with respect to λ_i gives

$$\frac{\partial f^t}{\partial \lambda_i} = -\alpha e_n^t \phi_i(H).$$

Thus,

$$\frac{\partial f_j^t}{\partial \lambda_i} = y_{i,j}. \quad \square$$

The above result says that Y being appropiately scaled is the transpose of the Jacobian of $f(\Omega)$,

$$J_\lambda(f) = Y^t$$

and provides the sensitivity of f with respect to the λ_i. We are also concerned with the sensitivity of the λ_i with respect to perturbations in f. This analysis is already done; for if $J_\lambda(f)$ is nonsingular, then $J_f(\lambda) = (J_\lambda(f))^{-1}$.

To a large extent the efficiency of this method depends on the triangular factorization of a matrix of eigenvectors. This factorization provides an inexpensive estimate of its condition number (see [8] or DLACN2 in [12]). Unfortunately, the factorization can also be a source of instability. Generically, L tends to be ill-conditioned for large n. The accuracy

of the decomposition depends primarily on the accuracy of the matrix L, which is computed recursively. A small example may help clarify some of the difficulties.

Consider the matrix

$$H = \begin{bmatrix} a & 0 \\ \delta & a+\varepsilon \end{bmatrix}.$$

If $\delta = 0$, then I is a matrix of both right and left eigenvectors and the eigenvalues are perfectly conditioned. However, our SVL^t factorization of the matrix of eigenvectors requires an unreduced H, and so does not exist. The condition number of an eigenvalue of H gives a bound on the perturbations in λ in relation to perturbations in H. The 2-norm condition number is defined to be

$$\kappa_\lambda = \frac{1}{|y^* x|}$$

where y^* and x are unit 2-norm left and right eigenvectors corresponding to λ. Note H has the following unit 2-norm eigenpairs:

$$([1,0], a) \quad \text{and} \quad (\tfrac{1}{\sqrt{1+(\delta/\varepsilon)^2}}[1, -\delta/\varepsilon]^t, a)$$

$$([0,1], a+\varepsilon) \quad \text{and} \quad (\tfrac{1}{\sqrt{1+(\delta/\varepsilon)^2}}[\delta/\varepsilon, 1], a+\varepsilon)$$

So the 2-norm condition numbers for the eigenvalues of H are

$$\kappa_a = \kappa_{a+\varepsilon} = \sqrt{1+(\delta/\varepsilon)^2}.$$

Notice that for a fixed $\varepsilon \neq 0$ the eigencondition of H improves as $\delta \to 0$, while

$$L = \begin{bmatrix} 0 & \dfrac{\delta}{\sqrt{\delta^2+\varepsilon^2}} \\ 1 & \dfrac{\varepsilon}{\sqrt{\delta^2+\varepsilon^2}} \end{bmatrix}$$

becomes singular! In short, computing ξ^t by solving $SVL^t\xi^t = f^t$ could potentially introduce ill-conditioned subproblems into a well conditioned problem.

7 Error Analysis

We have suggested that $|\ddot{\lambda}_k/2|$ be used as an error estimate for λ_k, based on a Taylor expansion of the solution to the homotopy initial value problem. This may not give satisfactory assurance of accuracy, especially given the typically ill-conditioned systems that must be solved to compute $\ddot{\lambda}$.

If Y is our matrix of left eigvectors of $H - e_1 f^t$, then the Bauer-Fike theorem says that the eigenvalues of H are within $\|f\|\nu(H - e_1 f^t) \leq \|f\|\kappa(Y)$ of those of Λ, where $\nu(H - e_1 f^t)$ is the spectral condition number of $H - e_1 f^t$. This result is typically too pessimistic to be useful here, for while we can estimate $\kappa(Y)$ efficiently, it is typically poorly scaled, and not a good estimate of $\nu(H - e_1 f^t)$. The difficulty in getting a good estimate of $\nu(H - e_1 f^t)$ is precisely why it is cheap. Y is factored into triangular factors. In order to achieve good row (or column) scaling, it is necessary to explicitly form Y, which requires about $n^3/3$ flops. We can improve the estimate by noticing that $\nu(H - e_1 f^t) \leq \kappa(D^{-1}Y)$ for any nonsingular diagonal matrix D, and we can use a condition estimator for $D^{-1}Y$

without explicitly forming Y. Now

$$D^{-1}Y(H-e_1f^t)Y^{-1}D=\Lambda$$

$$D^{-1}YHY^{-1}D=\Lambda+D^{-1}\omega\xi^tD, \tag{7.1}$$

and we would like D to minimize $\eta=\|D^{-1}\omega\xi^tD\|_2$. Osborne in [19] shows that a diagonal matrix with positive real elements that balances A also minimizes $\|D_\alpha^{-1}AD_\alpha\|_F$ over all nonsingular diagonal matrices D_α. As $\omega\xi^t$ is rank one, it is easy to verify that

$$D=\operatorname{diag}\left(\left|\frac{\omega_1}{\xi_1}\right|^{\frac{1}{2}},\left|\frac{\omega_2}{\xi_2}\right|^{\frac{1}{2}},\ldots,\left|\frac{\omega_n}{\xi_n}\right|^{\frac{1}{2}}\right)$$

balances $\omega\xi^t$. Therefore, it also minimizes η since $\|\cdot\|_F=\|\cdot\|_2$ for rank 1 matrices. We then have

$$\begin{aligned}\|D^{-1}\omega\xi^tD\|_2^2&=\|D^{-1}\omega\|_2^2\,\|\xi^tD\|_2^2\\&=\sum_{k=1}^n\left|\left|\frac{\xi_k}{\omega_k}\right|^{\frac{1}{2}}\omega_k\right|^2\sum_{k=1}^n\left|\left|\frac{\omega_k}{\xi_k}\right|^{\frac{1}{2}}\xi_k\right|^2\\&=\left(\sum_{k=1}^n|\xi_k||\omega_k|\right)^2\\&=(|\xi|^t|\omega|)^2\end{aligned}$$

giving a minimum value of $\eta=|\xi|^t|\omega|$.

The quantity η above is useful from an inclusion-region perspective. From (7.1) we

see that H is similar to $\Lambda + R$, with $\|R\|_2 = |\xi|^t|\omega|$. Bauer-Fike now says H is similar to a matrix whose eigenvalues differ from those of Λ by $|\xi|^t|\omega|$. An even better result holds for the *updated* eigenvalues neglecting the second order terms. Since $r_{ii} = \dot{\lambda}_i = \omega_i \xi_i$, H is similar to a matrix whose eigenvalues differ from those of $\Lambda + \mathrm{diag}(r_{ii})$ by $\|R - \mathrm{diag}(r_{ii})\|_2$. Since R is a rank 1 matrix

$$\begin{aligned} \|R - \mathrm{diag}(r_{ii})\|_2 &\leq \|R - \mathrm{diag}(r_{ii})\|_F \\ &= [(|\xi|^t|\omega|)^2 - \textstyle\sum_k |\dot{\lambda}_k|^2]^{1/2} \\ &= [(|\xi|^t|\omega|)^2 - \textstyle\sum_k |\omega_k \xi_k|^2]^{1/2}. \end{aligned} \tag{7.2}$$

We can also use the off diagonal elements of R to estimate the radii of Gershgorin disks about the updated eigenvalues.

Informally, an algorithm is said to be *backward stable* if it provides an answer that is the exact solution of a nearby problem. In terms of the eigenvalue problem, an algorithm is backward stable if the eigenpair $(\tilde{x}, \tilde{\lambda})$ it outputs for input A is the exact solution of $(A + E)\tilde{x} = \tilde{\lambda}\tilde{x}$ with $\frac{\|E\|}{\|A\|} \approx O(\mathbf{u})$ where $\mathbf{u}$ is the machine precision. A complete error analysis of this algorithm is still left to be done. So how can we have any confidence in our output? The answer is residuals. If $\tilde{x}$ is scaled so that $\|\tilde{x}\| = 1$, then the residual associated with $(\tilde{x}, \tilde{\lambda})$ is

$$r = A\tilde{x} - \tilde{\lambda}\tilde{x}. \tag{7.3}$$

We will now show that if the residual is small, then E is small. From (7.3) we see

$$A\tilde{x} - r = \tilde{\lambda}\tilde{x}$$

$$A\tilde{x} - r\tilde{x}^*\tilde{x} = \tilde{\lambda}\tilde{x}$$

$$(A - r\tilde{x}^*)\tilde{x} = \tilde{\lambda}\tilde{x}$$

giving $E = -r\tilde{x}^*$. Since E is rank 1,

$$\|E\|_2 = \|r\tilde{x}^*\|_2 = \|r\|_2\|\tilde{x}^*\|_2 = \|r\|_2.$$

So a small residual indicates a small E, and we can be confident we have solved a nearby problem. Note an analagous result holds for the residual based on left eigenvectors.

Residual norm estimates are available since

$$y_k^*(H - \lambda_k I) = \omega_k f^t$$

and

$$(H - \lambda_k I)x_k = \xi_k e_1.$$

These would be particularly valuable if we knew the norm of either y_k^* or x_k since the backward error in λ_k is given by $\frac{\|y_k^*(H-\lambda_k I)\|}{\|y_k^*\|}$ or $\frac{\|(H-\lambda_k I)x_k\|}{\|x_k\|}$. Again, the factored form of Y

is frustrating as we don't have y_k explicitly. Notice though, that

$$\begin{aligned}
\frac{\|y_k^*(H-\lambda_k I)\|\|(H-\lambda_k I)x_k\|}{\|y_k^*\|\|x_k\|} &= \frac{\|y_k^* e_1 f^t\|\|e_1 f^t x_k\|}{\|y_k^*\|\|x_k\|} \\
&= \frac{|\omega_k|\ \|f^t\|\ \|e_1\|\ |\xi_k|}{\|y_k^*\|\|x_k\|} \\
&= \frac{|\dot{\lambda}_k|\|f\|}{\|y_k^*\|\|x_k\|} \\
&\leq \frac{|\dot{\lambda}_k|\|f\|}{\|y_k^* x_k\|} \\
&= |\dot{\lambda}_k|\|f\|.
\end{aligned}$$

An upper bound for the minimum of these two relative residuals is $\rho_k = \sqrt{|\dot{\lambda}_k|\|f\|}$.

If the upperbound gives an unsatisfactory error, we are resigned to form y_k from SVL^t and calculate the residual explicitly. If this relative residual is still too big, a (left) inverse iteration is applied with the current eigenvalue and eigenvector estimate from the homotopy. After updating λ with the inverse iteration, the new relative residual is explicitly computed. If accurate eigenvectors of H are required, we suggest an inverse iteration on all of the eigenvalues, not just those that fail the backward error tests.

8 Numerical Experiments

Experiments were performed in MATLAB® [16] on randomly generated matrices with entries uniformly distributed between -1 and 1 using the RAND function. The matrices were reduced to Hessenberg form via MATLAB's HESS function. A total of 50 matrices were tested for each of size 50, 100, 150, 200, 250, 300, 400, and 500. In all of our tests, we have performed the restarted homotopy iterations until $|\ddot{\lambda}| < 2\varepsilon$ where ε is the sqare root

of the machine precision. Upon convergence one additional homotopy update is performed to help reduce residuals. As described at the end of the previous section, inverse iterations are then performed on any eigenpairs that fail to satisfy a relative residual tolerance set to 10^{-9} for these experiments.

Essential to this method is the initial estimate of the eigenvalues of H. This was obtained via the QR algorithm with implicit double shifts but with a relaxed deflation tolerance. Specifically, the subdiagonal elements of the intermediate transformed matrices were considered to be zero when

$$h_{i+1,i} < tol(|h_{i,i}| + |h_{i+1,i+1}|) \tag{8.1}$$

where tol is dependent on the size of H. An effective value of tol in terms of overall flops was determined through experiments to be

$$tol = \begin{cases} 0.01 & : n < 400 \\ \\ 0.001 & : n \geq 400 \end{cases}.$$

This method for the initial guess provides estimates near the actual eigenvalues, which serves to reduce the number of homotopy iterations required and the number of homotopy paths with potential bifurcations. This in turn reduces the size of the Arnoldi step or in many instances avoids it all together. This is all discussed further in section 8.1.

Recall that we use the second order term in (3.3) as an error estimate for the Euler prediction and therefore calculate the time step to be $h_k = \sqrt{2\tau/|\ddot{\lambda}_k|}$. Currently the eigen-

values that are candidates for being replaced by the Arnoldi minimization are those with $h_k < j$, where j is equal to the maximum of 2 and the current iteration number. The eigenvalues to be replaced are selected from the candidates in order of minimum h_k, and the total chosen may not exceed $n/4$. Due to the quality of the initial eigenvalue estimates, of the 400 matrices tested, 219 of them never utilized the Arnoldi minimization step. For the others, typically only the first homotopy iteration required the Arnoldi minimization, m never exceeded 12 and was usually between 2 and 4.

We present two comparisons. The first is of flop counts for the restarted homotopy method versus the QR algorithm with implicit double shifts with tol in (8.1) set to the machine precision. The second is of residuals from the restarted homotopy method versus MATLAB's EIG routine. These experiments indicate that on at least well conditioned problems the restarted homotopy method is as much as two times faster than QR with implicit double shifts. This difference is made more significant by the fact the QR algorithm as employed for flop comparison only calculated the eigenvalues, whereas the homotopy method supplies a factorization of a matrix of left eigenvectors as well as some eigenvectors calculated explicitly from inverse iteration as necessary to meet a residual tolerance. This must be qualified. The computed left eigenvectors, SVL^t, are not as accurate as those from QR as implemented in MATLAB's EIG routine. However, with typically just one inverse iteration after the homotopy process has converged, we have found that this method is significantly more accurate than the QR method for computing the eigenvalues as it gives much smaller residuals. A summary of flop counts is given in Table 8.1 and residuals in Table 8.2. It should be noted that actually 51 matrices of size 500 were run as one of them failed to converge. The results presented for $n = 500$ are only for those that converged.

n	avg n^3 flops		min n^3 flops		max n^3 flops	
	RHM	QR	RHM	QR	RHM	QR
50	3.9	6.1	3.2	5.5	4.7	7.0
100	3.5	5.7	2.9	5.4	4.1	6.6
150	3.4	5.5	2.8	5.0	4.7	6.0
200	3.6	5.4	2.9	5.1	4.5	5.8
250	3.7	5.3	2.9	4.9	4.5	5.6
300	3.8	5.2	2.8	5.0	4.8	5.6
400	4.1	5.1	3.4	4.9	5.6	5.5
500	4.2	5.1	2.8	4.9	5.1	5.4

Table 8.1: Flop Count for Restarted Homotopy Method (RHM) and QR

n	RHM	EIG
50	4.5×10^{-11}	4.5×10^{-12}
100	9.7×10^{-10}	4.4×10^{-10}
150	1.0×10^{-9}	1.7×10^{-9}
200	1.0×10^{-9}	3.5×10^{-8}
250	1.0×10^{-9}	5.3×10^{-8}
300	1.0×10^{-9}	8.8×10^{-8}
400	1.0×10^{-9}	1.4×10^{-6}
500	1.0×10^{-9}	5.2×10^{-6}

Table 8.2: Maximum Residual from 50 Matrices: RHM vs MATLAB's EIG

8.1 Importance of the Initial Eigenvalue Estimate

A few methods were considered for providing the initial eigenvalue estimates needed to begin the first homotopy iteration. The key consideration here was in limiting the total flops. The game one has to play here is this. An inexpsensive initial guess, such as a list of random numbers, may be easy on the front end but is paid for on the back end. Inexpensive estimates are typically a poor approximation to the actual eigenvalues and would require more homotopy iterations and larger Arnoldi minimizations to converge if convergence occurs at all. Increasing the work done in providing the initial estimate moves more of the cost to the front end. Expensive estimates will be nearer the actual eigenvalues and reduce

the homotopy iterations and Arnoldi minizaitons needed. So the goal is to find a balance between the cost of the initial estimate and the homotopy iterations.

We will now present a comparison of the restarted homotopy method for four different techniques of obtaining the initial guess. The testing was done on 30 matrices of size 200. Method 1 uses as the intial guess the eigenvalues of the 2×2 diagonal blocks of H. This method is extremely cheap as it requires only O(n) flops. Method 2 employs a divide and conquer strategy and is presented in Algorithm 8.1.

Algorithm 8.1: A Spectrum Estimate Routine

Input: Hessenberg matrix $H \in \mathbb{R}^{n \times n}$

Output: Ω, which is an approximation to $\lambda(H)$

1. If $n \geq 8$ go to 2 else go to 5.
2. Find k such that $.4n < k < .6n$ and $h_{k,k-1}$ is of smallest magnitude.
3. Find the eigenvalues of $H(k{:}n, k{:}n)$ and append to Ω.
4. Let $H \leftarrow H(1{:}k-1, 1{:}k-1)$ and $n \leftarrow k-1$ and go to 1.
5. Let $p = \lfloor \frac{n}{2} \rfloor$ and append the eigenvalues of $H(1{:}p, 1{:}p)$ and $H(p+1{:}n, p+1{:}n)$ to Ω.

Method 3, the one ultimately implemented, is the QR algorithm with double implicit shifts and a relaxed deflation tolerance. That is again $h_{i+1,i}$ is considered to be zero when $tol = 0.01$ in (8.1). Method 4 is also the QR algorithm with double implicit shifts but with the stricter deflation tolerance where the machine precision replaces tol in (8.1). Method 4 is the opposite extreme of Method 1 in that it solves the eigenvalue problem completely

and renders the homotopy steps unnecessary.

Method	Estimate Flops		Homotopy Flops		Total Flops	
	mean	max	mean	max	mean	max
1	0	0	9.9	18.3	9.9	18.3
2	1.2	1.3	4.5	6.8	5.6	7.8
3	1.8	2.1	1.5	1.7	3.3	3.8
4	5.4	5.7	0.7	0.7	6.0	6.4

Table 8.3: n^3 Flops for Restarted Homotopy Method with Different Initial Estimates

Table 8.3 illustrates the dance that occurs between the initial eigenvalue estimates and the total flops. We do indeed see that extra effort put into generating the initial eigenvalue estimates does result in less work needed for refinement by the homotopy iterations. However, there is a point where the less work required for the homotopy steps due to a quality initial guess does not offset the cost of that guess. This is seen by comparing Method 4 to Method 3. Residual checks were not performed for the purposes of these comparisons.

An efficient value of tol in (8.1) was determined through experiments and depends on n. Fifty matrices of each size $n = 50, 60, 70, \ldots, 500$ were evaluated using the QR algorithm with double implicit shifts and $tol = 10^{-1}, 10^{-2}, \ldots, 10^{-6}$ in (8.1) for the initial eigenvalue estimate. The mean of the total flops requred for the initial estimate and homotopy iterations combined was calculated for each combination of n and tol and then used to determine an effectivel tol based on n.

8.2 Bifurcation Avoidance Through Arnoldi Minimization

Here we will present a demonstration of the Arnoldi minimization handling homotopy paths with bifurcations. We construct a 10×10 matrix

$$A = \begin{bmatrix} 1 & & & & & & & & & \\ & 2 & & & & & & & & \\ & & 3 & & & & & & & \\ & & & 4 & & & 0 & & & \\ & & & & 5 & & & & & \\ & & & & & 6 & & & & \\ & & & 0 & & & 7 & & & \\ & & & & & & & 8 & & \\ & & & & & & & & 1 & 1 \\ & & & & & & & & -1 & 1 \end{bmatrix}$$

whose spectrum is

$$\lambda(A) = \{1,2,3,4,5,6,7,8,1+i,1-i\}.$$

We then perform a similarity transformation of A by a randomly generated matrix R giving a full matrix $B = RAR^{-1}$. We then finally put B in Hessenberg form via MATLAB's HESS function, $H = \text{hess}(B)$. Disregarding rounding errors the eigenvalues of H are those of A.

Matrix H is input into the restarted homotopy program along with the initial eigenvalue estimates

$$\Omega = \{1,2,3,4,5,6,7,8,9,10\}.$$

This guarantees that the initial homotopy paths contain a bifurcation point, specifically those originating from $\lambda_9 = 9$ and $\lambda_{10} = 10$. The program will identify these troublesome paths through the second derivates and will replace them with those from the Arnoldi minimization.

Notice in Table 8.4 that our error estimates, $\ddot{\lambda}$, are very low for the eigenvalues 1

λ	$\dot{\lambda}$	$\ddot{\lambda}$	h
1	-1.3 $\times$ 10^{-17}	-2.6 $\times$ 10^{-17}	6.0 $\times$ 10^{7}
2	-2.4 $\times$ 10^{-16}	-4.7 $\times$ 10^{-16}	1.4 $\times$ 10^{7}
3	-8.5 $\times$ 10^{-16}	-1.5 $\times$ 10^{-15}	8.0 $\times$ 10^{6}
4	6.1 $\times$ 10^{-16}	8.2 $\times$ 10^{-16}	1.1 $\times$ 10^{7}
5	4.5 $\times$ 10^{-15}	1.4 $\times$ 10^{-15}	8.4 $\times$ 10^{6}
6	3.3 $\times$ 10^{-15}	-7.7 $\times$ 10^{-15}	3.5 $\times$ 10^{6}
7	-2.6 $\times$ 10^{-14}	2.7 $\times$ 10^{-13}	5.9 $\times$ 10^{5}
8	3.0 $\times$ 10^{-14}	-1.4 $\times$ 10^{-12}	2.6 $\times$ 10^{5}
9	6.5 $\times$ 10^{1}	1.1 $\times$ 10^{4}	3.0 $\times$ 10^{-3}
10	-8.2 $\times$ 10^{1}	-1.1 $\times$ 10^{4}	3.0 $\times$ 10^{-3}

Table 8.4: Initial Homotopy Values for Matrix H with Initial Estimate Ω

through 8, which are essentially exact. However, those for 9 and 10 wtih paths containing bifurcations are on the order of 10^4. Correspondingly, the time steps, h, are large for the good eigenvalues and small for the bad ones. Indeed the program isolates $\lambda_9 = 9$ and $\lambda_{10} = 10$ as the only eigenvalues to be replaced by those from the Arnoldi minimization.

The Hessenberg matrix output from the Arnoldi minimization process is

$$W = \begin{bmatrix} 2.119736592076983 & -9.433466991510892 \times 10^{-1} \\ 2.389164066258232 & -1.197365920764506 \times 10^{-1} \end{bmatrix}.$$

The eigenvalues of W are calculated to be $1.000000000000266 \pm .9999999999997584i$ and replace 9 and 10 in the next and final homotopy iteration. This illustrates a bifurcation avoidance via a real to complex interchange. Also note that since the 'good' eigenvalues used to generate the Arnoldi input vector are close to actual eigenvalues of H, the eigenvalues of the Arnoldi output are close to those of the remaining eigenvalues of H.

9 Conclusions and Future Work

Although faster and more accurate than *QR* on these randomly generated well conditioned problems, and much more parallelizable, the restarted homotopy method as presented is neither as general purpose nor as robust as the QR algorithm. For example, an easy way to construct a matrix for which the method does not converge is to generate a square random matrix of size $n > 30$, compute its Hessenberg form, and then divide each subdiagonal element by 20. This ensures that the product of the subdiagonals is almost certainly much less than the machine precision, making L singular to working precision as L is lower right triangular and

$$l_{1,n} = \prod_{i=1}^{n-1} \frac{h_{i,i+1}}{\|\hat{l}_{i+1}\|}.$$

Further, even on well conditioned problems, this method fails to consistently converge if the size of the matrix is over 500. The most probable culprit for this is that L becomes increasingly ill-conditioned with increased matrix size.

Two primary avenues may lead toward making this a general purpose eigensolver competitive with *QR*. Perhaps the most natural is a block implementation, where one solves a multi-input inverse eigenvalue problem that computes matrices $B \in \mathbb{R}^{n\times m}$ and $F \in \mathbb{R}^{m\times n}$ so that $A - BF$ has the desired spectrum. This introduces more BLAS 3 operations and may improve the conditioning of the L matrix from step 1. Convergence results may be more complicated in this case, since if $m > 1$ the matrix F is no longer unique. For the homotopy method to converge, it is necessary for F to approach 0. F does become unique once the eigenvectors are restricted, so pursuing this route would require a method for calculating F so that the eigenvectors of $H - BF$ were near those of H. The multi-input eigenvalue as-

signment method of Arnold and Datta [4] computes an F and a triangular matrix L, which block diagonalizes $A - BF$ but does not meet the eigenvector requirement.

Another avenue could be to explicitly calculate an eigenvector matrix for $H - e_1 f^t$. While prohibitively more expensive than the current factorization SVL^t in a serial implementation, it may lead to a more robust parallel version of the restarted homotopy method. The right eigenvectors, x_i, of the matrix $H - e_1 f^t$ can be computed directly by solving the Hessenberg systems

$$(H - \lambda_i I)x_i = e_1, \quad i = 1, 2, \ldots, n. \tag{9.1}$$

Given $X = [x_1, x_2, \ldots, x_n]$, we can compute the quantities needed for the homotopy updates $\omega = X^{-1} e_1$ and $\xi^t = f^t X = e^t$, where e is the vector of ones.

References

[1] J. Ackermann, *Der Entwurf Linear Regelungssysteme im Zustandsraum*. Regulungestechnik und Prozessedatenverarbeitung, vol. 7, 297-300, 1972

[2] M. Arnold and B.N. Datta, *Single-Input Eigenvalue Assignment Algorithms: A Close Look*. SIAM J. Matrix Analysis, vol. 19, no. 2, 444-467, 1998

[3] M. Arnold, *Algorithms and Conditioning for Eigenvalue Assignment,* Ph.D. Dissertation, Northern Illinois University, DeKalb, May 1993.

[4] M. Arnold and B.N. Datta, *An Algorithm for the Multiinput Eigenvalue Assignment Problem*. IEEE Trans. Automat. Contr., vol. 35, no. 10, 1149-1152, 1990

[5] B.N. Datta, *An Algorithm to Assign Eigenvalues in a Hessenberg Matrix: Single Input Case*. IEEE Trans. Automat. Contr., vol. AC-32, no. 5, 414-417, 1987

[6] Francis, J. G. F. *The QR transformation: a unitary analogue to the LR transformationm, Parts I and II.* I. Comput. J. 4 265271, 332-345, 1961/1962

[7] G.H. Golub and C. Van Loan, *Matrix Computations*. Third Edition, John Hopkins University Press, Baltimore, 1996

[8] Higham, N.J., *FORTRAN Codes for Estimating the One-Norm of a Real or Complex Matrix, with Applications to Condition Estimation*. ACM Transactions on Mathematical Software, vol. 14, no.4, 381-396, Dec. 1988

[9] Higham, N.J.; Konstantinov, M.; Mehrmann, V.; Petkov, P., *The sensitivity of computational control problems*. Control Systems Magazine, IEEE , vol.24, no.1, 28-43, Feb. 2004

[10] T. Kailath, *Linear Systems*. Prentice-Hall, Englewood Cliffs, N.J., 1980

[11] J. Kautsky, N.K. Nichols and P. Van Dooren, *Robust Pole Assignment in Linear State Feedback*. Int. J. Control, vol. 41, no. 5, 1129-1155, 1985

[12] Anderson, E., Bai Z., Bischof C., Demmel J., Dongarra J., Du Croz J., Greenbaum A., Hammarling S., McKenney A., Ostrouchov S., and Sorensen D., *LAPACK Users' Guide*, SIAM, Philadelphia, 1992

[13] T.Y. Li, Z. Zeng, and L. Cong, *Solving Eigenvalue Problems of Real Nonsymmetric Matrices with Real Homotopies*, SIAM J. Numer. Anal., vol. 29, no. 1, 229-248, 1992

[14] T.Y. Li, Z. Zeng, *Homotopy-Determinant Algorithm for Solving Nonsymmetric Eigenvalue Problems*, Mathematics of Computation, vol 59, no. 200, 483-502, 1992

[15] S.H. Lui, H.B. Keller, and T.W.C. Kwok, *Homotopy Method for the Large, Sparse, Real Nonsymmetric Eigenvalue Problem*, SIAM J. Matrix Anal, vol. 18, no. 2, 312-333, 1997

[16] MathWorks, Inc., The *MATLAB version 7.11.0.584 R2010b*. The MathWorks, Inc., Natick, MA, 2010

[17] V. Mehrmann and H. Xu, *An analysis of the pole placement problem. I. The singleinput case*. Electr. Trans. Num. Anal., Vol 4, 89-105, 1996

[18] G.S. Miminis and C.C. Paige, *An Algorithm for Pole Assignment of Time Invariant Linear Systems*. Int. J. Control, vol. 35, no. 2, 341-354, 1982

[19] E.E. Osborne, *On Pre-conditioning of Matrices*. J. Assoc. Comput. Mach., vol. 7, 338-345, 1960

[20] P. Hr. Petkov, N.D. Christov, and M.M. Konstantinov, *A computational algorithm for pole assignment of linear signle-input systems*. IEEE Trans. Automat. Contr.,vol. AC-29, pp.1045-1048, 1984

[21] Y. Saad (2011) *Numerical Methods for Large Eigenvalue Problems*. 2nd edition, Retrieved March 11, 2011,
from http://www–users.cs.umn.edu/∼saad/eig_book_2ndEd.pdf

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

[23] W.M. Wonham, *Linear Multivariable Control: A Geometric Approach*. 2nd edition, Springer, New York, 1979

A Main Code: rhm.m

function [ev,S,V,L,f,lambda_dot,t,M,flops,fail] = rhm(H,ev,hom_tol,eig_tol)

%Calculate the eigenvalues of the unreduced hessenberg matrix H.

%

%ev is an estimate to the eigenvalues of H.

%S∗V∗L' is a factorization of left eigvenvalues of H.

%f is the final feedback vector.

%lambda_dot is the derivative of the eigenvalues for the final homotopy

%paths.

%t is the number of iterations performed.

%M is a vector containing the number of eigenvalues recomputed using the

%Arnoldi method per iteration.

%flops is the number of flops performed discounting lower order terms.

%fail will return 0 if the algorithm converged and 1 otherwise.

%

% External calls

% qralg.m

% combined.m

% orgeig3.m

```
% choosem.m

% arnoldi.m

% xpolyvalroots2.m


%check if H is triangular

if min(min(H==triu(H)))==1 || min(min(H==tril(H)))==1

    disp('Matrix is triangular');

    ev=diag(H);

    return

end


%Verify H is unreduced

if ismember(0,diag(H,-1))

    disp('Matrix is not unreduced');

    return

end


n = max(size(H));

nh = norm(H,'fro');

fail=0;

flops=0;

M=zeros(30,1); %Tracks the number of bad eigenvalues per iteration.
```

```
%Specify the homotopy tolerance and eigenvalue tolerance if not supplied
if nargin<3
    hom_tol = sqrt(n/(2*nh^2));
end
if nargin <4
    eig_tol=sqrt(eps);
end

%Get an initial eigenvalue estimate if not supplied.
if nargin<2
    if n<400
        [ev,flops]=qralg(H,.01);
    else
        [ev,flops]=qralg(H,.001);
    end
end
ent=zeros(1,n);
ent(1,n)=1;

if max(isnan(ev)) || max(isinf(ev))
    fail=1;
    return
```

```
end

for t=1:30

%Sort the eigenvalues with complex pairs first.
ev =orgeig3(ev);
rl=real(ev(:,1));
im=imag(ev(:,1));
im2=im.*im;

%Calculate the homotopy terms.
[f,L,S,V,lambda_dot,lambda_double_dot]=combined(H,ev,rl,im,im2,n);
nf=norm(f);

%Check if everything is converged. That is are the eigenvalue paths
%essentially straight.
if max(abs(lambda_double_dot)) < 2*eig_tol
k=1;
  while k < n
    if im(k)
      ev(k) = ev(k) + lambda_dot(k) + lambda_double_dot(k)/2;
      ev(k+1) = conj(ev(k));
```

```
    k = k+2;
  else
    ev(k:n) = ev(k:n) + lambda_dot(k:n) + real(lambda_double_dot(k:n))/2;
    k = n+1;
  end
end
%Perform one additional iteration to get better residuals
ev =orgeig3(ev);
rl=real(ev);
im=imag(ev);
im2=im.^2;
[f,L,S,V,lambda_dot,lambda_double_dot]=combined(H,ev,rl,im,im2,n);
k=1;
while k < n
  if im(k)
    ev(k) = ev(k) + lambda_dot(k) + lambda_double_dot(k)/2;
    ev(k+1) = conj(ev(k));
    k = k+2;
  else
    ev(k:n) = ev(k:n) + lambda_dot(k:n) + real(lambda_double_dot(k:n))/2;
    k = n+1;
  end
```

```
   end



   M=M(1:t-1);
   m=nonzeros(M);
   flops=flops+sum(((n-m).^3)/3)+(t+1)/3*n^3+sum(m)*(n^2)+sum((m.^2))*n+10*sum(m.
        ^3);
   flops=flops/n^3;
   return
end

%Calculate homotopy time steps

h(:,1) = (sqrt(2 * hom_tol ./ abs(lambda_double_dot)));

%----------------------Determine the number of 'bad' eigenvalues.
m=choosem(h,n,nf,nh,t);
M(t)=m;

%find the m 'bad' eigenvalues with the smallest h.

if m≠0
```

```
%Make sure conjugate pairs do not get separated between good and bad.
%The variable bad will hold the indices of ev corresponding to the bad eigenvalues.
%The variable good will hold the indeces of ev corresponding to the good eigenvalues.
    [¬,I] = sort(h);
    bad = I(1:m);
    if im(bad(m)) && rem(bad(m),2)
        bad(m+1) = bad(m)+1;
        m = m+1;
    end
    good=setdiff(I, bad);
else
    good=(1:n);
    bad=[];
end

%Update the 'good' eigenvalues with the largest using the homotopy update.
h=min(h,1);
k=1;
lg=n-m; %lg is number of good eigenvalues

while k ≤lg
```

```
  if ¬im(good(k))
    ev(good(k)) = ev(good(k)) + h(good(k)) ∗ lambda_dot(good(k)) + h(good(k))^2 ∗ real
        (lambda_double_dot(good(k))) / 2;
    k=k+1;
  else
    ev(good(k)) = ev(good(k)) + h(good(k)) ∗ lambda_dot(good(k)) + h(good(k))^2 ∗
        lambda_double_dot(good(k)) / 2;
    ev(good(k)+1) = conj(ev(good(k)));
    k=k+2;
  end
end
if max(isnan(ev)) || max(isinf(ev))
    fail=1;
    return
end

%Update the bad eigenvalues to be the eigenvalues from the arnoldi iteration
if m≠0

%First find the arnoldi input vector, b=en^t∗phi(H') where phi is the
%polynomial with updated good egienvalues as roots.
```

```
    b=xpolyvalroots2(H,ent,ev(good));


    if m≠0
        [Arnoldi_Matrix,¬,¬,c] = arnoldi(H',b',m,1);
        if c<m
            disp(['Arnoldi breakdown: h(c+1,c) = ',num2str(Arnoldi_Matrix(c+1,c)),' c =
                ',int2str(c)])
            return
        end
        if max(max(isnan(Arnoldi_Matrix))) || max(max(isinf(Arnoldi_Matrix)))
            fail=1;
            return
        end
        ev(bad) = eig(Arnoldi_Matrix);
    end
    if max(isnan(ev)) || max(isinf(ev))
    fail=1;
    return
    end
end
end
fail=1;
```

A.1 subroutine: combined.m

```
function [f,L,S,V,lambda_dot,lambda_double_dot]=combined(H,ev,rl,im,im2,n)
%function [f,L,lambda_dot,lambda_double_dot,V,S]=combined(H,ev,rl,im,im2,n)
%
%diag(S)VL'(H-e1*f')=diag(ev)*inv[diag(S)VL']

%----------------------CALCULATE FEEDBACK AND L-----------------
%Hdiag is the diagonal elements of H and used to make the recursion run faster.
Hdiag = diag(H,0);

L(n,1) = 1 ;
scl=ones(n-1,1);
%The recursion is L(i+1) = (H'-real(ev(i))*I) * L(i) + v(i)^2 * L(i-1).
%Where v(i) = imag(ev(i)) if i is odd and v(i) = 0 if i is even.
for j=1:n
   for k=1:n
      H(k,k) = Hdiag(k) -rl(j);
   end
   rblock = n-j+1:n;
   if j<n-1
         cblock= n-j:n;
```

```
   else
        cblock= 1:n;
   end
   if mod(j,2) == 1
      L(cblock,j+1) = H(rblock,cblock)'*L(rblock,j);
   else
      L(cblock,j+1) = H(rblock,cblock)'*L(rblock,j) + im2(j-1)*L(cblock,j-1)/scl(j-1,1);
   end
   if j<n
      scl(j,1)=2^round(log2(norm(L(n-j:n,j+1))));
      L(n-j:n,j+1)=L(n-j:n,j+1)/scl(j,1);
   end
end

f=L(:,n+1)/L(1,n);

L=L(:,1:n);
%-----------------Calculate V and S-----------

nisodd = mod(n,2);

if nisodd
```

```
    nm1=n-1;
else
    nm1=n;
end


%The ith diagonal 2x2 block of V is the identity. Vii = I
V=eye(n);

%The goal is to find V such that VG'=BV where B is block diagonal.
%Note G'(i) * V(ij) + V(ij)G'(j)= V(i,j-1)*e2*e1' is the equation we use to solve for V.
%Using the above equation we come up with a matrix equation Ax=b where the elements
    of x are the elements of V(ij)

 for i=1:nm1/2-1
   for j=i+1:nm1/2
        a1=rl(2*i-1)-rl(2*j-1);
        a2=rl(2*i-1)-rl(2*j);
        a3=rl(2*i)-rl(2*j);
        a4=im2(2*j-1)-im2(2*i-1);
        a5=rl(2*i)-rl(2*j-1);
        A(1,1)= (a1*a2+a4)/scl(2*j-1);
```

```
        A(1,2)= scl(2*i-1)*(a1+a3)/scl(2*j-1);
        A(2,1)= -im2(2*i-1)*(a1+a3)/(scl(2*i-1)*scl(2*j-1));
        A(2,2)= (a5*a3+a4)/scl(2*j-1);
        C(1,1) = a2/scl(2*j-1);
        C(1,2) = scl(2*i-1)/scl(2*j-1);
        C(2,1) = -im2(2*i-1)/(scl(2*i-1)*scl(2*j-1));
        C(2,2) = a3/scl(2*j-1);
        V(2*i-1:2*i,2*j)=A\(scl(2*j-2)*V(2*i-1:2*i,2*j-2));
        V(2*i-1:2*i,2*j-1)=C*V(2*i-1:2*i,2*j);
    end
end

if nisodd
    for i=1:2:nm1
        V(i:i+1,n)=[rl(i)-ev(n) scl(i); -im2(i)/scl(i) rl(i+1)-ev(n)]\(scl(n-1)*V(i:i+1,nm1)
            );
    end
end

%Calculate 2x2 blocks of eigenvectors for B which is VGt=BV
S=zeros(n,2);
```

```
for k=1:nm1/2
   S(2∗k-1,2) = 1;
   if im(2∗k-1) == 0
      S(2∗k-1,1) = (ev(2∗k-1)- ev(2∗k))/scl(2∗k-1);
   else
      S(2∗k-1,1) = im(2∗k-1)∗sqrt(-1)/scl(2∗k-1);
   end

   S(2∗k,2) = 1;
   if im(2∗k-1) == 0
       S(2∗k,1) = 0;
    else
       S(2∗k,1) = im(2∗k)∗sqrt(-1)/scl(2∗k-1);
    end
end

if nisodd
      S(n,1) = 1;
end
```

```
%-----------------------Calculate Derivatives----------------

if nisodd
   p = n-1;
else
   p = n;
end

for j = 1:p/2
   w(2*j-1:2*j,1) = S(2*j-1:2*j,1:2)*V(2*j-1:2*j,n)*L(1,n);
end

if p ≠n
   w(n,1) = S(n,1)*V(n,n)*L(1,n);
end

%In solving for z*, first solve f' * L'^(-1).
%Do this by solving x where x*L' = f' using a 'weird backsub'.

x(1,n) = f(1)/L(1,n);
for j=2:n
```

```
 temp = f(j);
   for k=n-j+2:n
    temp = temp -L(j,k)*x(1,k);
   end
   x(1,n-j+1) = temp/L(j,n-j+1);
end


%Next solve a backsub where yV=x.
y(1,1) = x(1,1)/V(1,1);
for j=2:n
   temp = x(1,j);
   for k = 1:j-1
      temp = temp-y(1,k)*V(k,j);
   end
   y(1,j) = temp/V(j,j);
end


% Next solve z*S=y
   z=zeros(n,1);
   for k = 1:p/2
      z(2*k-1,1) = (y(1,2*k-1)-y(1,2*k)*S(2*k,1)/S(2*k,2))/(S(2*k-1,1) -S(2*k-1,2)*S(2*
           k,1)/S(2*k,2));
```

```
    z(2*k,1) = (y(1,2*k)-z(2*k-1,1)*S(2*k-1,2))/S(2*k,2);
  end

  if p ~= n
    z(n,1) = y(1,n)/S(n,1);
  end

%Calculate the derivative of the eigenvalue

lambda_dot = w.*z;

%Calculate the second derivative of the eigenvalue
k=1;
numerator=zeros(n,1);
lambda_double_dot=zeros(n,1);
while k<=n
    if im(k)
        numerator(1:n,1)=1;
        numerator(k)=0;
        denominator = -ev(:,1) + ev(k,1);
        denominator(k) = 1;
        lambda_double_dot(k,1) = 2*lambda_dot(k)*sum(lambda_dot.*(numerator./
```

```
            denominator));
        lambda_double_dot(k+1,1) = conj(lambda_double_dot(k));
        k=k+2;
    else
        numerator(1:n,1)=1;
        numerator(k)=0;
        denominator = -ev(:,1) + ev(k,1);
        denominator(k) = 1;
        lambda_double_dot(k,1) = real(2*lambda_dot(k)*sum(lambda_dot.*(numerator./
            denominator)));
        k=k+1;
    end
end
```

A.2 subroutine: arnoldi.m

```
function [H,Q,v,j]=arnoldi(A,b,m,reorth)
%Arnoldi method using MGS with optional reorthogonalization
%
%Uses MGS to find hessenberg H and unitary Q such that AQ=QH -v e_m^t
%b is the input vector.
%A is nxn
%b is nx1
%m is the size of the Arnoldi iterartion
%Q is nxm and span(q1, q2, ..., qk) = span(b, Ab,...,A^(k-1)b) for k=1:m
%j is the actual number of Arnoldi steps performed. If the procedure
%breaks down then j<m and eig(H) is contained in eig(A);

if nargin < 4
    reorth = 1;
end
zero_tol = 1e-10;
n=length(b);
H=zeros(m,m);
Q=zeros(n,m);
Q(:,1)=b/norm(b);
```

```
for j=1:m
    v = A*Q(:,j);
    for k = 1:j
        H(k,j) = Q(:,k)'*v;
        v = v -H(k,j)*Q(:,k);
    end
    if reorth == 1
        for k = 1:j
            tmp = Q(:,k)'*v;
            v = v -tmp * Q(:,k);
            H(k,j) = H(k,j) + tmp;
        end
    end
    if j<m
        H(j+1,j) = norm(v);
        if H(j+1,j) ≤zero_tol
            return
        end
        Q(:,j+1) = v/H(j+1,j);
    end
end
```

A.3 subroutine: xpolyvalroots2.m

```
function y = xpolyvalroots2(h,x,lam)
%function y = xpolyvalroots(h,x,lam)
%
%y = x*(a-lam(1))*(a-lam(2))*...*(a-lam(n))/norm(x*(a-lam(1))*...*(a-lam(n-2))
d = length(lam); [m,n] = size(x); ln2 = log(2); t = 0;
l = orgeig3(lam);
if rem(d,2), flag=1; else flag=2; end
for i=1:2:d-flag,
% Compute ak and bk, where x^2 + ak*x + bk = (x -l(i))*(x -l(i+1))
    ak = -(real(l(i)) + real(l(i+1)));
    bk = real(l(i))*real(l(i+1)) -imag(l(i))*imag(l(i+1));
% Compute x = x*(H^2 + ak*H + bk*I)
    y = x*h + ak*x;
    x = bk*x + y*h;
% Scale x
x=x/2^(ceil(log2(norm(x))));
end
if flag==2,
    ak = -(real(l(d-1)) + real(l(d)));
    bk = real(l(d-1))*real(l(d)) -imag(l(d-1))*imag(l(d));
```

```
    y = x∗h + ak∗x;

    y = bk∗x + y∗h;

else

    y = x∗h -l(d)∗x;

end
```

A.4 subroutine: orgeig3.m

```
function [lambda,ipvt]=orgeig(eigs,ctol)
%function [lambda,ipvt]=orgeig(eigs,ctol)
%
% sorts an n-vector of real numbers and complex pairs
% so that complex pairs appear consecutively in the first
% lk positions, and the real numbers appear grouped
% as nearest nbrs in the last n-lk positions.
% eigs is the vector of such numbers
% ctol is a tolerance for snapping a cplx number to a real number
% lambda is the sorted vector
% ipvt is the permutation (lambda = ipvt(eigs))
%
% calls findreal

n = length(eigs); lambda = zeros(n,1); ipvt = 1:n;

%snap to real axis
if nargin < 2, ctol = 1e-10; end
jj = findreal(eigs,ctol); eigs(jj) = real(eigs(jj));
```

```
%get complex pairs
kk = find(imag(eigs)); lk = length(kk);
if rem(lk,2), disp('not self conjugate'); end
nk = ones(n,1); nk(kk) = zeros(lk,1); nk = find(nk); %nk = ¬kk
ipvt = ipvt([kk;nk]); eigs = eigs(ipvt);

%make sure cplx pairs are together
jj = find(imag(eigs) > 0); [lc,li] = sort(eigs(jj));
lambda(1:2:lk) = lc;
ip1 = ipvt(jj(li));
jj = find(imag(eigs) < 0); [lc,li] = sort(eigs(jj));
lambda(2:2:lk) = lc;
ip2 = ipvt(jj(li));
ipvt([1:2:lk 2:2:lk]) = [ip1 ip2];
jj = find(imag(eigs) == 0); [lc,li] = sort(eigs(jj)); ip2 = ipvt(jj(li));
lambda(lk+1:n) = lc;
ipvt(lk+1:n) = ip2;
%sort real eigs into pairs of close values
for j=lk+1:2:n-2,
   [m,jj] = min(abs(diff(lambda(j:n))));
   t = lambda(j); lambda(j) = lambda(j-1+jj); lambda(j-1+jj) = t;
   t = lambda(j+1); lambda(j+1) = lambda(j+jj); lambda(j+jj) = t;
```

```
    t = ipvt(j); ipvt(j) = ipvt(j-1+jj); ipvt(j-1+jj) = t;
    t = ipvt(j+1); ipvt(j+1) = ipvt(j+jj); ipvt(j+jj) = t;
    [s,jj] = sort(lambda(j+2:n));
    lambda(j+2:n) = s; ipvt(j+2:n) = ipvt(j+1+jj);
end
```

A.5 subroutine: findreal.m

```
function t = findreal(x,tol)
%function t = findreal(x,tol)
%
%finds the indices of real elements of a vector
%a scalar is called real if |imagpart| < tol or |imagpart/modulus| < tol

if nargin < 2, tol = 500*eps*max(abs(x)); end
if any(abs(x)==0),
   t = find(abs(imag(x)) < tol);
else
   t = find( abs(imag(x))./abs(x) < tol | abs(imag(x)) < tol);
end
```

A.6 subroutine: choosem.m

```
function [m]=choosem(h,n,nf,nh,j)

%Finds the number of eigenvalues to find using the Arnoldi minimization.

maxarni=floor(n/4);

    %Candidates for Arnoldi minimization:
  candidates = find(h < 10); maxshift = length(candidates);

  %number of eigs to replace (ad hoc)
  if nf > nh/4,
    if rem(j,2), m = 5; else m = min([maxshift maxarni]); end
  else
    if rem(j,2), m = 2; else m = min([maxshift maxarni]); end
  end
  if j < log2(n)-1, m = min([maxarni length(find(h<j))]); end
  if j == 1, m = min([floor(n/4) length(find(h<2))]); end
```

A.7 subroutine: qralg.m

```
function [ev, flop,h,Q]=qralg(h,tol)
%function [ev, flop,h,Q]=qralg(h,tol)
%QR algorithm with implicit double shifts. Input h is Hessenberg.
%tol is the tolerance to make subdiagonal elements zero.
%Output h is real Schur form of input h. h=Q'hQ.
%flop is the total flops.
%
%calls francisqr.m
[n,¬] = size(h);
findQ=0;

if nargout>3
    findQ=1;
    Q=eye(n);
end
if nargin==1;
    tol=eps;
end
p=0;
q=0;
```

```
flop = 0;
if n==1
    ev=h;
    return
end

if n>2
while q<n
    %Set to zero all subdiagonal elements sufficiently small.
    row1=p+2;
    row2=n-q;
    for j=row1:row2
        if abs(h(j,j-1))≤tol∗(abs(h(j,j))+abs(h(j-1,j-1)))
            h(j,j-1)=0;
        end
    end
    %Find largest q and smallest p such that h(n-q+1:n,n-q+1:n) is quasi-upper
    %triangluar and h(p+1:n-q,p+1:n-q) is unreduced.
    d=diag(h,-1);

    [p,q]=findpq(d,n);
    %Perform Francis QR step on h(p+1:n-q,p+1:n-q)
```

```
    if q < n
        if findQ
            [h(p+1:n-q,p+1:n-q),P]=francisqr(h(p+1:n-q,p+1:n-q));
            Q(:,p+1:n-q)=Q(:,p+1:n-q)*P;
            h(1:p,p+1:n-q)=h(1:p,p+1:n-q)*P;
            h(p+1:n-q,n-q+1:n)=P'*h(p+1:n-q,n-q+1:n);
            flop=flop+13.5*(n-q-p)^2+(n^2*(n-2*(q+p)+5)+n*(2*q*p+q^2+p^2-5*(q+p)
                -6))+(p^2*(2*(q-n)+p-5)+p*(n^2-2*q*n+5*n+q^2-5*q-6))+(q^2*(2*(p-n
                )+q-5)+q*(n^2-2*p*n+5*n+p^2-5*p-6));
        else
            h(p+1:n-q,p+1:n-q)=francisqr(h(p+1:n-q,p+1:n-q));
            flop = flop + 8 *(n-q-p)^2;
        end

    end
end
else
    d=diag(h,-1);
end

%Find eigenvalues of the 2x2 or 1x1 diagonal blocks on h.
t=1;
```

```
z=find(d==0);
numzeros=length(z);
ev=zeros(n,1);
for j=1:numzeros+1
    if j==numzeros+1;
        m=n;
    else
        m=z(j);
    end
    %Diagonalize any 2x2 blocks with real eigenvalues

    if t<m
        desc=((h(t,t)+h(m,m))^2-4*(h(t,t)*h(m,m)-h(t,m)*h(m,t)));
        if desc>0
            [P,T]=schur(h(t:m,t:m),'complex');
            P=real(P);
            T=real(T);
            ev(t:m)=(diag(T));
            h(t:m,t:m)=T;
            h(m,t)=0;
            h(1:t-1,t:m)=h(1:t-1,t:m)*P;
            h(t:m,m+1:n)=P'*h(t:m,m+1:n);
```

```
            if findQ
                Q(:,t:m)=Q(:,t:m)*P;
            end
        end
    end
    ev(t:m)=eig(h(t:m,t:m));
    t=m+1;
end
```

A.8 subroutine: francisqr.m

```
function [h,Q]=francisqr(h)
%Does one Francis QR Step for Hessenberg H
%Code from Golub & Van Loan Matrix Computations
%calls house.m
n = max(size(h));
if n<3
    Q=eye(n);
    return
end
nm1 = n-1;
findQ=0;
if nargout==2
    findQ=1;
    Q=eye(n);
    V=zeros(3,nm1);
end
%find x=(h-aI)*(h-bI)*e1 = (h^2 -s*h + tI)e1 where a and b are
%eigenvalues of 2x2 trailing principal submatrix.
s=h(nm1,nm1)+h(n,n);
t=h(nm1,nm1)*h(n,n)-h(nm1,n)*h(n,nm1);
```

```
x=h(1:3,1:2)*h(1:2,1)-s*h(1:3,1)+[t;0;0];
for k=0:n-3
    [v,b]=house(x);
    q=max([1,k]);
    h(k+1:k+3,q:n) = h(k+1:k+3,q:n)-(b*v)*(v'*h(k+1:k+3,q:n));
    r=min([k+4,n]);
    h(1:r,k+1:k+3)=h(1:r,k+1:k+3) -(h(1:r,k+1:k+3)*(b*v))*v';
    if k<n-3
        x=h(k+2:k+4,k+1);
    else
        x=h(k+2:k+3,k+1);
    end
    if findQ
        V(2:3,k+1)=v(2:3);
        V(1,k+1)=b;
    end
end
[v,b]=house(x);
h(nm1:n,n-2:n) = h(nm1:n,n-2:n)-(b*v)*(v'*h(nm1:n,n-2:n));
h(1:n,nm1:n) = h(1:n,nm1:n)-(h(1:n,nm1:n)*(b*v))*v';
if findQ
    V(2,nm1)=v(2);
```

```
    V(1,nm1)=b;
    Q(nm1:n,nm1:n)=Q(nm1:n,nm1:n)-(b*v)*v';
    for j=n-2:-1:1
        v=[1;V(2:3,j)];
        Q(j:j+2,j:n)= Q(j:j+2,j:n) -(V(1,j)*v)*(v'*Q(j:j+2,j:n));
    end
end
h=triu(h,-1);
```

A.9 subroutine: findpq.m

```
function [p,q]=findpq(d,n)
%Find largest q and smallest p such that h(n-q+1:n,n-q+1:n) is
%quasi-upper triangluar and h(p+1:n-q,p+1:n-q) is unreduced.
%d=diag(h,-1) where h is nxn Hessenberg.

z=find(d==0)+1;
numzeros=length(z);
if numzeros == 0
    p=0;
    q=0;

elseif ¬ismember(z(numzeros),[n n-1])
    p=z(numzeros)-1;
    q=0;
else
    flag=1;
    k=numzeros;
    q=n-z(k)+1;
    while flag && k>1
        if (z(k)-z(k-1))<3
```

```
            q=n-z(k-1)+1;
            k=k-1;
        else
            flag=0;
        end
    end
    if q>n-3
        q=n;
    end
    if flag==0
        p=z(k-1)-1;
    else
        p=0;
    end
end
```

A.10 subroutine: house.m

```
function [v,b]=house(x)
%Given x in Rn, house returns v in Rn and b in R, st Px=norm(x)*e1.
%P=I-bvv' is a householder reflector for x.

n=length(x);
s=x(2:n)'*x(2:n);
v=[1;x(2:n)];
if s==0
    b=0;
else
    m=sqrt(x(1)^2+s);
    if x(1)<=0
        v(1)=x(1)-m;
    else
        v(1)=-s/(x(1)+m);
    end
    b=2*v(1)^2/(s+v(1)^2);
    v=v/v(1);
end
```

A.11 subroutine: residualcheck.m

```
function [ev,res,Y,flops] = residualcheck(ev,H,S,V,L,ld,f,rtol)
%function [ev, res] = residualcheck[ev,S,V,L,ld,ldd]
%Estimates the residual and if the error criterion is not met,
%performs an inverse iteration. Any eigenvalues subject to inverse
%iteration will be updated and stored back in ev.
%H is matrix with eigenvalues ev
%S,V,L are left eigenvector factors from homotopy
%ld is the first derivative of ev
%n is size of ev
%f is feedback
%nh is frobenious norm of matrix H
%The last n columns of Y will be a matrix of left eigenvectors calculated
%using inverse iteration if necessary. The first column of Y will hold
%the index of the corresponding eigenvalue of ev.
%flops will have the total flops based solely on inverse iterations
%performed and calculation of left eigenvector from SVL' as necessary.
%A maximum of two inverse iterations is permitted per eigenvalue.
%
%Calls hessys1.m
```

```
n=length(ev);
k=1;
Y=0;
flops=0;
nf=norm(f);
absld=abs(ld);
res=sqrt(absld∗nf);%upperbound for minimum of left or right residual
updatedev=0; %counter tracking number of eigenvalues updated through
                %inverse iterations
total_invit=0;%counts number of inverse iterations performed;
while k≤n
    invit=0;
    if res(k)>rtol
        %Calculate the residual using the estimated left eigenvector
        %if residual upperbound is too high.
        if ¬mod(k,2)
            y=(S(k,1)∗V(k-1,k-1:n)+S(k,2)∗V(k,k-1:n))∗L(:,k-1:n)';
        elseif k==n
            y=S(n,1)∗V(n,n)∗L(:,n)';
        else
            y=(S(k,1)∗V(k,k:n)+S(k,2)∗V(k+1,k:n))∗L(:,k:n)';
        end
```

```
flops = flops + 2*(n-k+1)+(n-k+1)^2;
y=y/norm(y);
res(k)=norm(y*H-ev(k)*y);
if res(k)>rtol
    invit=1;
    updatedev=updatedev+1;
    %Perform inverse iteration if residual is too high
    y = hessysl(H,y,ev(k));
    y=y/norm(y);
    yH=y*H;
    ev(k)=(yH)*y'; %update eigenvalue
    res(k)=norm(yH-ev(k)*y);
end
if res(k)>rtol
    invit=2;
    %Perform second inverse iteration if residual still too high.
    y = hessysl(H,y,ev(k));
    y=y/norm(y);
    yH=y*H;
    ev(k)=(yH)*y'; %update eigenvalue
    res(k)=norm(yH-ev(k)*y);
```

```
        end
        total_invit=total_invit+invit;
        if imag(ev(k)) && invit
            Y(updatedev,1:n+1)=[k,y];
            updatedev=updatedev+1;
            Y(updatedev,:)=[k+1,conj(y)];
            ev(k+1)=conj(ev(k));
        elseif invit
            Y(updatedev,1:n+1)=[k,y];
        end
    end
    if imag(ev(k))
        res(k+1)=res(k);
        k=k+2;
    else
        k=k+1;
    end
end
flops = (flops+ total_invit*2*n^2)/n^3;
```

A.12 subroutine: hessysl.m

```
function x = hessysl(a,b,l)
%function x = hessys(a,b,l);
%
% x solves x*(a -l*I) = b, where a is a Hessenberg matrix
[p,n] = size(b);
if nargin > 2, for j=1:n, a(j,j) = a(j,j) -l; end, end
x = zeros(p,n); p = 1:n;
a = [b;a];
% Triangularization
for j=n:-1:2,
   jm1 = j -1;
   if abs(a(j+1,p(jm1))) > abs(a(j+1,p(j))),
      t = p(j); p(j) = p(jm1); p(jm1) = t; % Pivot
   end
   m = -a(j+1,p(jm1))/a(j+1,p(j));
   if abs(m) > 1/eps, disp('small pivot'); end
   a(1:j+1,p(jm1)) = m*a(1:j+1,p(j)) + a(1:j+1,p(jm1));
end
% Forward substitution
x(1) = a(1,p(1))/a(2,p(1));
```

```
for j = 2:n,
   x(j) = (a(1,p(j)) -x(1:j-1)∗a(2:j,p(j)))/a(1+j,p(j));
end
```