An Analysis of Eigenvalue Calculation Algorithms

A deep dive in to the QR algorithm

by

Aditya Tripathy

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY HYDERABAD

November 2024

Certificate

It is certified that the work contained in this thesis entitled **"An Analysis of Eigenvalue Calculation Algorithms"** by **Aditya Tripathy** has been carried out under my supervision and that it has not been submitted elsewhere for a degree.

Prof. GVV Sharma

Professor

EE Department

Indian Institute of Technology Hyderabad

Declaration

This is to certify that the thesis titled **"An Analysis of Eigenvalue Calculation Algorithms"** has been authored by me. It presents the research conducted by me under the supervision of **Prof. GVV Sharma** .

To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted elsewhere, in part or in full, for a degree. Further, due credit has been attributed to the relevant state-of-the-art and collaborations with appropriate citations and acknowledgments, in line with established norms and practices.

Aditya Tripathy
Roll No. EE24BTECH11001
EE Department
Indian Institute of Technology Hyderabad

Abstract

Name of the student: **Aditya Tripathy** Roll No: **EE24BTECH11001**

Degree for which submitted: **BTECH** Department: **EE Department**

Thesis title: **An Analysis of Eigenvalue Calculation Algorithms**

Thesis supervisors: **Prof. GVV Sharma**

Month and year of thesis submission: **November 2024**

In general, any method for computing eigenvalues necessarily involves an infinite number of steps. This is because finding eigenvalues of an $n \times n$ matrix is equivalent to finding the roots of its characteristic polynomial of degree n and Computing coefficients of characteristic polynomial requires computation of the determinant, however, the problem of finding the roots of a polynomial can be very ill-conditioned and for $n > 4$ such roots cannot be found (By Abel's theorem), in general, in a finite number of steps. In other words, we must consider iterative methods producing, at step k, an approximated eigenvector x_k associated with an approximated eigenvalue λ_k that converge to the desired eigenvector x and eigenvalue λ as the number of iterations becomes larger. Here I try to motivate the concept of the QR algorithm for finding eigenvalues of arbitrary matrices.

Contents

Chapter 1

Introduction

1.1 Why is this hard?

Symmetry makes life less complex. This statement is especially true when applied to the problem at hand. The problem of finding eigenvalues is already quite involved in the case of real and symmetric matrices. At least, these symmetric matrices have a guarantee of all eigenvalues being real. In the general case of complex matrices, it becomes mandatory for us to use iterative methods to "converge" to the true eigenvalues.
The spectrum of algorithms available on the mathematical market can be classified broadly into two categories :

1. **Direct Methods:** Direct methods are typically used on dense matrices and cost $O(n^3)$ operations to compute all eigenvalues and eigenvectors; this cost is relatively insensitive to the actual matrix entries.

2. **Iterative Methods:** These are usually applied to sparse matrices or matrices for which matrix-vector multiplication is the only convenient operation to perform. Iterative methods typically provide approximations only to a subset of the eigenvalues and eigenvectors and are usually run only long enough to get a few adequately accurate eigenvalues rather than a large number. Their convergence properties depend strongly on the matrix entries.

Here I will try and motivate one of the most often used direct algorithms for the task at hand - namely, the QR algorithm.

1.2 Design Considerations

Before we set out to accomplish our goal to calculate eigenvalues for arbitrary matrices, we must put some thought behind what qualities our "ideal" algorithm should have :

1. **Accuracy:** We wouldn't want to waste time and effort implementing an algorithm which doesn't provide the true value of eigenvalues up to a desired precision.
2. **Efficiency:** While we need to come to terms with the fact that we can't reap the efficiency rewards of exploiting structure of specific types of matrices, we should strive for the fastest algorithm which gets the job done.
3. **Convergence:** One might argue that this is a subset of the previous design consideration, but it is important enough to point out for itself. Oceans worth of ink has been used to improve convergence properties of various algorithms, so we must also look into this aspect.
4. **Stability:** A slightly more nuanced design consideration. But we want the algorithm not to blow up if we change the entries within the matrix by a relatively small amount.

Chapter 2

The Basic QR Algorithm

2.1 Preliminaries

2.1.1 Similarity Transformations

Definition: Two matrices $A, B \in \mathbb{C}^{n\times n}$ are said to be similar if there exists an invertible matrix $P \in \mathbb{C}^{n\times n}$ such that

$$B = P^{-1}AP \tag{2.1}$$

Similar matrices represent the same linear map under two (possibly) different bases, with P being the change-of-basis matrix. Because matrices are similar if and only if they represent the same linear operator with respect to (possibly) different bases, similar matrices share all properties of their shared underlying operator like rank, the characteristic polynomial and by association the eigenvalues and the eigenvectors.

2.1.2 Unitary Matrices

An invertible complex square matrix U is unitary if its matrix inverse U^{-1} equals its conjugate transpose U^*, that is, if

$$UU^* = U^*U = I \tag{2.2}$$

where I is the identity matrix.

2.1.3 Schur Decomposition

It allows one to write an arbitrary complex square matrix as unitarily similar to an upper triangular matrix whose diagonal elements are the eigenvalues of the original matrix. If $A \in \mathbb{C}^{n\times n}$ is an $n \times n$ square matrix with complex entries, then A can be expressed as

$$A = QUQ^{-1} \tag{2.3}$$

for some unitary matrix Q (so that the inverse Q^{-1} is also the conjugate transpose Q^* of Q), and some upper triangular matrix U. Since U is similar to A, it has the same spectrum, and since it is triangular, **its eigenvalues are the diagonal entries of** U. It must be noted that all square have a Schur decomposition, although it may not be unique.

2.1.4 Hessenberg Form

A Hessenberg matrix is a special kind of square matrix, one that is "almost" triangular. To be exact, an upper Hessenberg matrix has zero entries below the first sub-diagonal, and a lower Hessenberg matrix has zero entries above the first super-diagonal.

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix} \tag{2.4}$$

2.2 The Basic Idea Behind QR

Our primary goal is to get the Schur decomposition of the given matrix and read off the diagonal values to be the eigenvalues.

The QR algorithm works off of the following observation,
For every matrix A we can represent it as the product of a unitary matrix Q and an upper triangular matrix R.

$$A_1 = Q_1 R_1 \tag{2.5}$$
$$A_2 = R_1 Q_1 = Q_1^* A Q_1 = Q_2 R_2 \tag{2.6}$$
$$A_3 = R_2 Q_2 = Q_2^* Q^* 1 A Q_1 Q_2 \tag{2.7}$$
$$\vdots \tag{2.8}$$
$$A_{k+1} = Q_{k-1}^* Q_{k-2}^* \cdots Q_2^* Q_1^* A Q_1 Q_2 \cdots Q_{k-2} Q_{k-1} \tag{2.9}$$

We can see that A_{k+1} is similar to A_k. It can be shown that under certain conditions, as we run these iterations, A_{k+1} will converge to an upper triangular matrix[1]. So now if we take

$$U = Q_1 Q_2 \cdots Q_{k-2} Q_{k-1} \tag{2.10}$$

then,

$$A = U^* A_\infty U \tag{2.11}$$

So we have successfully calculated the Schur decomposition of the desired matrix.

2.3 Improving the QR Algorithm

Two of the major drawbacks of this unoptimized algorithm are as follows:

1. The convergence of the algorithm is slow. In fact it can be arbitrarily slow if eigenvalues are very close to each other.

2. The algorithm is expensive. Each iteration step requires the computation of the QR factorization of a full $n \times n$ matrix, i.e., each single iteration step has a complexity $O(n^3)$. Even if we assume that the number of steps is proportional to n we would get an $O(n^4)$ complexity. The latter assumption is not even assured, see point 1 of this discussion.

Because of the number of arithmetic operations involved, the QR algorithm is practical only when applied to a matrix in upper Hessenberg form (i.e., $a_{ij} = 0$ for $i > j+1$). Hence

[1] We will explore this notion in Chapter 4

the method consists of two steps. In the first step, we find a similar matrix which is in upper Hessenberg form. The second step is then to apply the QR algorithm to this new matrix. One can show that if A is in upper Hessenberg form, applying the QR algorithm results in a sequence of matrices that are also in this form.

Chapter 3

The Hessenberg QR Algorithm

The flow of this algorithm is as follows:

1. Convert given matrix to upper Hessenberg form using Householder Reflectors in $O(n^3)$ time complexity.
2. Use Givens Rotations to reduce the matrix to upper triangular matrix in $O(n^2)$ time.

The motivation to convert to upper Hessenberg form is that the computations are much cheaper and hence faster, which achieves Design Consideration 2.

3.1 Householder Reflectors

Definition: A matrix of the form

$$P = I - 2\mathbf{u}\mathbf{u}^* \tag{3.1}$$

$$\tag{3.2}$$

is called a Householder reflector.
It is easy to verify that Householder reflectors are Hermitian and that $P^2 = I$. From this we deduce that P is unitary. It is clear that we only have to store the Householder vector u to be able to multiply a vector (or a matrix) with P,

$$P\mathbf{x} = \mathbf{x} - \mathbf{u}(2\mathbf{u}^*\mathbf{x}) \tag{3.3}$$

$$\tag{3.4}$$

A task that we repeatedly want to carry out with Householder reflectors is to transform a vector $\mathbf{x}$ on a multiple of e_1

$$P\mathbf{x} = \mathbf{x} - \mathbf{u}(2\mathbf{u}^*\mathbf{x}) = \alpha e_1 \tag{3.5}$$

$$\tag{3.6}$$

Since P is unitary, we must have $\alpha = \rho||\mathbf{x}||$, where $\rho \in \mathbb{C}$ has absolute value one. Therefore,

$$\mathbf{u} = \frac{\mathbf{x} - \rho||\mathbf{x}||e_1}{||\mathbf{x} - \rho||\mathbf{x}||e_1||} \tag{3.7}$$

We can freely choose ρ provided that $|\rho| = 1$. Let $x_1 = |x_1|e^{iu_k{}^*\phi}$. To avoid numerical cancellation we set $\rho = -e^{i\phi}$. In the real case, one commonly sets ρ = -sign(x_1). If $x_1 = 0$ we can set ρ in any way.

3.1.1 Algorithm to Convert to Hessenberg Form

The main algorithm is as follows

Algorithm 1 An algorithm with caption

for k = 1 to n - 2 **do**
  Generate the Householder reflector P_k
  $A_{k+1:n,k:n} := A_{k+1:n,k:n} - u_k(2u_k{}^*A_{k+1:n,k:n})$ {Apply P_k to A from left}
  $A_{1:n,k+1:n} := A_{1:n,k+1:n} - 2(A_{1:n,k+1:n}u_k)u_k{}^*$ {Apply P_k to A from right}
end for

To get a visualization of the transformation,

$$P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0}^\top \\ \mathbf{0} & I_4 - 2u_1u_1^* \end{bmatrix} \tag{3.8}$$

$$A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \xrightarrow[P_1 *]{} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} \xrightarrow[* P_1]{} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} \tag{3.9}$$

The multiplication of P_1 from the left inserts the desired zeros in column 1 of A. The multiplication from the right is necessary in order to have similarity. Because of the nonzero structure of P_1 the first column of $P_1 A$ is not affected. Hence, the zeros stay there.

The reduction continues in a similar way:

$$P_1^* A P_1 = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} \xrightarrow[P_2 * / * P_2]{} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{bmatrix} \xrightarrow[P_3 * / * P_3]{} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}$$

$$= P_3 P_2 P_1 A P_1 P_2 P_3 \tag{3.10}$$

It is worth pointing out that each of the pairs of multiplications by P_k on the right and left are similarity transformations, hence they have no effect on the eigenvalues of the underlying matrix.

3.2 Givens Rotations

Givens rotations are used to zero specific elements of a vector or matrix by rotating the vector in the plane of two coordinates. A Givens rotation matrix is defined as:

$$G(i,j,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & -\overline{s} & \cdots & \overline{c} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix},$$

where

$$c_k = \frac{\overline{x_i}}{\sqrt{|x_i|^2 + |x_j|^2}} \tag{3.11}$$

$$s_k = \frac{\overline{x_j}}{\sqrt{|x_i|^2 + |x_j|^2}} \tag{3.12}$$

Now the algorithm to run one iteration for the upper Hessenberg form is as follows:

Algorithm 2 A Hessenberg QR Step

1: **for** $k = 1, 2, \ldots, n-1$ **do**
2: Generate G_k and then apply it: $H := G(k, k+1, \theta_k)^* H$.
3: Compute: $[c_k, s_k] :=$ givens$(H_{k,k}, H_{k+1,k})$.
4: Update rows $k : k+1$: $H_{k:k+1,k:n} = \begin{bmatrix} c_k & -s_k \\ s_k & c_k \end{bmatrix} H_{k:k+1,k:n}$.
5: **end for**
6: **for** $k = 1, 2, \ldots, n-1$ **do**
7: Apply the rotations G_k from the right:
8: $H_{1:k+1,k:k+1} = H_{1:k+1,k:k+1} \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix}$.
9: **end for**

3.3 Complexity

We will now compute time complexities for both the major steps in our Hessenberg QR algorithm:

1. **Computing Hessenberg :**

   (a) Application of P_k from the left : $\sum_{k=1}^{n-2} 4(n-k-1)(n-k) \approx \frac{4n^3}{3}$

   (b) Application of P_k from the right : $\sum_{k=1}^{n-2} 4n(n-k) \approx 2n^3$

2. **Running Givens Rotations:** Neglecting the minimal overhead of calculating c_k and s_k, executing the 2 loops in the algorithm costs :

$$2 \times \sum_{i=1}^{n-1} 6i = 6n(n-1) \approx 6n^2 \tag{3.13}$$

As we can see, while running the iterations to converge to the Schur decomposition, the Hessenberg algorithm requires $6n^2$ arithmetic computations for each iteration while the naive QR algorithm takes $\frac{7n^3}{3}$ operations.

Chapter 4

Addressing Convergence Issues

4.1 Does the Algorithm Always Converge?

At the start of the discussion I talked about the convergence of the QR iteration to an upper triangular matrix "under certain conditions". We will now explore what these conditions are, and when does the convergence fail to occur.

4.1.1 Necessary Condition For Convergence

The iterations will converge in a "reasonable" number of iterations if

1. If all eigenvalues have distinct absolute values. Precisely, if $\lambda_1, \lambda_2 \cdots \lambda_n$ be the eigenvalues of the matrix, for convergence to occur

$$|\lambda_1| < |\lambda_2| < \cdots < |\lambda_n| \tag{4.1}$$

2. If the ratio of any two eigenvalues is "reasonably"[1] large. Precisely,

$$\frac{\lambda_i}{\lambda_j} >> 1, i \neq j \tag{4.2}$$

[1] See Reference No. 2 for more elaborate discussion

4.1.2 What Happens if Absolute Values are not Unique?

The short answer to this question is that you will get 2×2 blocks called **Jordan Blocks** where there is a deviation from upper triangular matrix as shown below

$$\begin{bmatrix} a_0 & & & & \\ & a_1 & & & \\ & & a_2 & & \\ & & & a_3 & a_4 \\ & & & a_5 & a_6 \end{bmatrix} \tag{4.3}$$

To "fix" this problem, we can stop the iterations at some place and check for these blocks and check for these blocks and solve for the eigenvalues of this trivial 2×2 manually.

An elegant solution to the problem comes through this notion of **shifts** and **deflation** which will be explored in the next chapter

Chapter 5

Rayleigh Quotients and Wilkinson Shifts

5.1 Why Shift?

As discussed in previous chapter, the convergence of Hessenberg QR algorithm is proportional to $\frac{\lambda_i}{\lambda_j}$. Suppose if the eigenvalues are really close, in that case the algorithm will converge arbitrarily slowly. A way to get around this problem is to take a number μ and perform the QR iterations on $H - \mu I$and add back μI to the new RQ matrix. Precisely

Algorithm 3 Demonstration of Shifts

$H_k - \mu I :=$ QR
$H_{k+1} = RQ + \mu I$

What this accomplishes is the following:
The rate of convergence is now proportional to

$$\frac{|\lambda_i - \mu|}{|\lambda_j - \mu|} \tag{5.1}$$

(5.2)

This increases the rate of convergence and the amount by which the rate increases depends on how close we can get to λ_j. The problem is that we don't know the eigenvalues (obviously). So the agenda now is to find a way to come up with an on-the-fly way to approximate the eigenvalues in between the iterations

5.2 Rayleigh Quotients

5.2.1 The Method

One such heuristic is the Rayleigh quotient shift: Set the shift σ_k in the k^{th} step of the QR algorithm equal to the last diagonal element:

$$\sigma_k := h_{n,n}^{(k-1)}) \tag{5.3}$$

Algorithm 4 The Hessenberg QR Algorithm with Rayleigh Quotient Shift

1: Let $H_0 = H \in \mathbb{C}^{n\times n}$ be an upper Hessenberg matrix. This algorithm computes its Schur normal form $H = UTU^*$.
2: Initialize $k := 0$.
3: **for** $m = n, n-1, \ldots, 2$ **do**
4: **repeat**
5: $k := k + 1$.
6: Compute the shift: $\sigma_k := h_{m,m}^{(k-1)}$.
7: Factorize $H_{k-1} - \sigma_k I = Q_k R_k$.
8: Update $H_k := R_k Q_k + \sigma_k I$.
9: Update $U_k := U_{k-1} Q_k$.
10: **until** $|h_{m,m-1}^{(k)}|$ is sufficiently small.
11: **end for**
12: Set $T := H_k$.

Algorithm 4.4 implements this heuristic. Notice that the shift changes in each iteration step! Notice also that deflation is incorporated in Algorithm 4.4. As soon as the last lower off-diagonal element is sufficiently small, it is declared zero, and the algorithm proceeds with a smaller matrix. In Algorithm 4.4 the "active portion" of the matrix is $m \times m$. One thing which I won't elaborate much on is that implementing Rayleigh shifts achieves what is called quadratic convergence, as compared to the linear convergence of the naive QR algorithm.

5.2.2 Where Does it Fail?

We still haven't addressed the issue of equal eigenvalues. To illustrate this, consider the following matrix

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{5.4}$$

Here the value of σ_k will be stuck at 0 since the eigenvalues are -1 and 1 and the algorithm can't choose which way to go, so it slowly decays to nothingness, leaving us with an infinite loop. This is the true reason why Jordan blocks show up. This problem showed up due a symmetry between the eigenvalues and σ_k. The next method breaks this symmetry and gives us a final method which will converge regardless of the entries in the matrix.

5.3 Wilkinson Shift

The heuristic chosen by this version is,

$$\sigma_k := \text{eigenvalue of } \begin{bmatrix} h_{n-1,n-1}^{(k-1)} & h_{n-1,n}^{(k-1)} \\ h_{n,n-1}^{(k-1)} & h_{n,n}^{(k-1)} \end{bmatrix} \tag{5.5}$$

Implementing this gives us our ultimate goal. An algorithm which produces eigenvalues of any arbitrary matrix, complex or not. (Also, as an added bonus it also achieves cubic convergence)

Chapter 6

Final Form of the Algorithm

Algorithm 5 Eigenvalue Calculation

To Upper Hessenberg form (via Householder reflection):
for $k = 1$ TO $n - 2$ **do**
  Compute Householder reflector P_k
  Update A: $A = P_k A P_k^T$
end for
QR Iterations (via Givens Rotation):
while $h_{n,n} > \epsilon$ **do**
  for $k = 1$ TO $n - 1$ **do**
    Compute Givens rotation matrix G_k to zero out the $(k + 1, k)$ element.
    Apply G_k from the left and right to A.
  end for
  Form $A = RQ$
end while
Extract Eigenvalues from Quasi-Triangular Matrix
for $i = 1$ TO n **do**
  if $|A_{i+1,i}| < \epsilon$ **then**
    Eigenvalue: $\lambda_i = A_{i,i}$
  else
    Compute eigenvalues of the 2×2 sub-matrix by solving quadratic:

$$\begin{bmatrix} A_{i,i} & A_{i,i+1} \\ A_{i+1,i} & A_{i+1,i+1} \end{bmatrix}$$

  end if
end for
return Eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_n$

Chapter 7

Comparison With Other Algorithms

In true Indian fashion, we shall compare our little algorithm with its cousins and debate which one is the best. Below is a table[1] which outlines pros and cons of other eigenalgorithms on the market.

Algorithm	**Pros**	**Cons**	**Time Complexity**	**Conditions**
Householder + Givens	Stable, precise; preserves sparsity.	Complex preprocessing for Hessenberg form.	$O(n^3)$	Any square matrix.
Jacobi Method	Accurate for symmetric matrices; simple.	Slow for non-symmetric matrices; inefficient for large ones.	$O(n^3)$ (dense)	Symmetric (real/complex).
Lanczos Algorithm	Efficient for sparse symmetric matrices.	Sensitive to rounding; limited to Hermitian matrices.	$O(kn^2)$	Sparse, symmetric/Hermitian.
QR Algorithm	Robust for eigenvalue computation.	High cost for large dense matrices.	$O(n^3)$	Any square matrix.

[1] Credits: Wikipedia

Algorithm	**Pros**	**Cons**	**Time Complexity**	**Conditions**
Arnoldi Algorithm	Handles non-symmetric sparse matrices.	Requires re-orthogonalization; less stable.	$O(kn^2)$	Sparse, any matrix.
Power Iteration	Simple, finds dominant eigenvalue.	Only computes the largest eigenvalue; slow for close eigenvalues.	$O(kn^2)$	Any square matrix.

In conclusion, the algorithm which we chose was the only true algorithm which can compute all eigenvalues of an arbitrary matrix in a time which is less than expected lifetime of the universe.

Chapter 8

Bibliography

1. A Handout on Eigenvalue Algorithms, by ETH Zurich
2. A Handout on Convergence of QR Algorithm, Illinois Institute of Technology
3. Lecture 16 on Shifted QR Algorithms, Massachusets Institute of Technology
4. Necessary and Sufficient Conditions for Convergence of QR Algorithm on Hessenberg Matrix, BERESFORD PARLETT, University of California Berkeley, California