15/11/2013

Iterative methods for eigenvalue problem
Project Report

Group members: Radharaman Roy, Mouktik Chattopadhyay

Necessity of an Iterative Method for Eigenvalues: 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 we present some iterative methods called the Schur factorization, QR method, power method, Bisection method, Jacobi's method, and Divide and conquer method.

Jacobi eigenvalue algorithm: Jacobi eigenvalue algorithm is an iterative method for calculation of the eigenvalues and eigenvectors of a symmetric matrix.

- **Givens rotation:** A Givens rotation is represented by a matrix of the form

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

where $c = \cos\theta$, $s = \sin\theta$ appear intersection of i th and j th rows and columns. That is the non zero elements of Givens matrix is given by $g_{kk} = 1$ for $k \neq i, j$

g_{ii}=c

g_{jj}=-s

$g_{ij} = s$ for $i > j$.

- **Description of Jacobi algorithm:** $S' = G^T S G$ where G be a Givens rotation matrix & S be a symmetric matrix then S' is also a symmetric matrix. S' has entries

$$S'_{ij} = S'_{ji} = (c^2 - s^2)S_{ij} + sc(S_{ii} - S_{jj}),$$

$$S'_{ik} = S'_{ki} = cS_{ik} - sS_{jk} \text{ for } k \neq i,j$$
$$S'_{jk} = S'_{kj} = sS_{ik} + cS_{jk} \text{ for } k \neq j,i$$
$$S'_{ii} = c^2 S_{ii} - 2scS_{ij} + s^2 S_{jj},$$
$$S'_{jj} = s^2 S_{ii} + 2scS_{ij} + c^2 S_{jj},$$

,where $c = \cos\theta$, $s = \sin\theta$, G is orthogonal,S and S' have the same Frobenious norm , however we can choose θ s.t $S'_{ij} = 0$, in which case S' has a large sum of squares on the diagonal.

$S'_{ij} = (\cos 2\theta)S_{iJ} + \frac{1}{2}(\sin 2\theta)(S_{ii} - S_{jj})$
Set this equal to 0 , $\tan 2\theta = \frac{2S_{ij}}{S_{jj}-S_{ii}}$.
The Jacobi eigenvalue method repeatedly performs rotation until the matrix becomes almost diagonal. Then the diagonal elements are approximations of the eigenvalues of S.

- **Convergence of Jacobi's method:**

  Let od(A)is the root-sum-of-squares of the (upper) off diagonal entries of A, so A is diagonal if and only if od(A) = 0. Our goal is to make od(A) approach 0 quickly. The next lemma tells us that od(A) decreases monotonically with every Jacobi rotation.

  - **Lemma:** Let A' be the matrix after calling Jacobi-Rotation (A, j, k)for any j,k. Then $od^2(A') = od^2(A) - a_{jk}^2$.

    Proof: Here A' = A except in rows and columns j and k. Write

    $$od^2(A) = \frac{1}{2} \times \text{sum of square of all off diagonal elements of A except } a_{jk} + a_{jk}^2$$

    and similarly $od^2(A') = S'^2 + {a'_{jk}}^2 = S'^2$, since $a'_{jk} = 0$ after calling Jacobi- Rotation(A,j,k). Since $\| X \|_F = \| QX \|_F$ and $\| X \|_F = \| XQ \|_F$ for any X and any orthogonal Q, we can show $S^2 = S'^2$. Thus $od^2(A') = od^2(A)$ - a_{jk}^2 as desired.

  - **Theorem:** After one Jacobi's rotation in the classical Jacobi's algorithm, we have $od(A') \leq \sqrt{(}1 - \frac{1}{N})$, where $N = \frac{n(n-1)}{2}$=the no. of sub-per diagonal entries of A. After k Jacobi rotation od(.) is no more than $(1 - \frac{1}{N})^{\frac{k}{2}} od(A)$.

    Proof: $od^2(A') = od^2(A) - a_{jk}^2$, where a_{jk} is the largest off diagonal entry. Thus $od^(A) \leq \frac{n(n-1)}{2} a_{jk}^2$. So $od^2(A) - a_{jk}^2 \leq (1 - \frac{1}{N}) od^2(A)$.
    So the classical Jacobi's algorithm converges at least linearly with the error (measured by od(A)) decreasing by a factor at least $\sqrt{(}1 - \frac{1}{N})$ at time. It eventually converges quadratically.

- **Algorithm:** This method determines the eigenvalues and eigenvectors of a real symmetric matrix A, by converting A into a diagonal matrix by similarity transformation.

  - **Step 1.** Read the symmetric matrix A.

  - **Step 2.** Initialize $D = A$ and S = I, a unit matrix.

- **Step 3.** Find the largest off-diagonal element (in magnitude) from $D = [d_{ij}]$ and let it be d_{ij} .

- **Step 4.**Find the rotational angle .
  If $d_{ii} = d_{jj}$ then
  if $d_{ij} > 0$ then $\theta = \frac{\pi}{4}$ else $\theta = -\frac{\pi}{4}$ endif;
  else
  $\theta = \frac{1}{2}\tan^1\left(\frac{2d_{ij}}{d_{ii}-d_{jj}}\right)$;
  endif;

- **Step 5.** Compute the matrix $S_1 = [s_{pq}]$
  Set $s_{pq} = 0$ for all $p, q = 1, 2, ..., n$
  $s_{kk} = 1$, $k = 1, 2, ..., n$
  and $s_{ii} = s_{jj} = \cos$, $s_{ij} = \sin$, $s_{ji} = sin$.

- **Step 6.** Find $D = S_1^T D S_1$ and $S = SS_1$;

- **Step 7.** Repeat steps 3 to 6 until D becomes diagonal.

- **Step 8.** Diagonal elements of D are the eigenvalues and the columns of S are the corresponding eigenvectors.

- **Program:** This program finds all the eigenvalues and the corresponding eigenvectors of a real symmetric matrix. Assume that the given matrix is real symmetric.

```
#include < stdio.h >
#include < math.h >
void main()
{
  int n,i,j,p,q,flag;
  float a[10][10],d[10][10],s[10][10],s1[10][10],s1t[10][10];
  float temp[10][10],theta,zero=1e-4,max,pi=3.141592654;
  printf("Enter the size of the matrix ");
  scanf("%d",&n);
  printf("Enter the elements row wise ");
  for(i = 1; i <= n; i + +) for(j = 1; j <= n; j + +) scanf("%f", &a[i][j]);
  printf("The given matrix is");
  for(i = 1; i <= n; i + +)
```

```
    {
      for(j = 1; j <= n; j++) printf("%8.5f", a[i][j]); printf("\n");
    }
  printf("\n");
for(i = 1; i <= n; i++) for(j = 1; j <= n; j++) {
    d[i][j]=a[i][j]; s[i][j]=0;
  }
for(i = 1; i <= n; i++) s[i][i]=1;
do
{
flag=0;
i=1; j=2; max=fabs(d[1][2]);
for(p = 1; p <= n; p++) for(q = 1; q <= n; q++)
  {
    if(p!=q)
      if(max < fabs(d[p][q]))
       {
         max=fabs(d[p][q]); i=p; j=q;
       }
  }
if(d[i][i]==d[j][j]) {
    if(d[i][j] > 0) theta=pi/4; else theta=-pi/4;
  }
else
    {
      theta=0.5*atan(2*d[i][j]/(d[i][i]-d[j][j]));
    }
for(p = 1; p <= n; p++) for(q = 1; q <= n; q++)
    s1[p][q]=0; s1t[p][q]=0;
for(p = 1; p <= n; p++) s1[p][p]=1; s1t[p][p]=1;
s1[i][i]=cos(theta); s1[j][j]=s1[i][i];
s1[j][i]=sin(theta); s1[i][j]=-s1[j][i];
s1t[i][i]=s1[i][i]; s1t[j][j]=s1[j][j];
s1t[i][j]=s1[j][i]; s1t[j][i]=s1[i][j];
for(i = 1; i <= n; i++)
    for(j = 1; j <= n; j++){
```

```
    temp[i][j]=0;
    for(p = 1; p <= n; p++) temp[i][j]+=s1t[i][p]*d[p][j];
  }
  for(i = 1; i <= n; i++)
   for(j = 1; j <= n; j++) {
    d[i][j]=0;
    for(p = 1; p <= n; p++) d[i][j]+=temp[i][p]*s1[p][j];
  }
  for(i = 1; i <= n; i++)
  for(j = 1; j <= n; j++)
   {
    temp[i][j]=0;
    for(p = 1; p <= n; p++) temp[i][j]+=s[i][p]*s1[p][j];
   }
  for(i = 1; i <= n; i++) for(j = 1; j <= n; j++) s[i][j]=temp[i][j];
  for(i = 1; i <= n; i++) for(j = 1; j <= n; j++)
   {
    if(i!=j) if(fabs(d[i][j] > zero)) flag=1;
   }
}
while(flag==1);
printf("The eigenvalues are \n");
for(i = 1; i <= n; i++) printf("%8.5f ",d[i][i]);
printf("\nThe corresponding eigenvectors are \n");
for(j = 1; j <= n; j++)
  {
   for(i = 1; i < n; i++) printf("% 8.5f,",s[i][j]);
    printf("%8.5f\n",s[n][j])
}
}
```

Divide and conquer method eigenvalue algorithm: This method we can use for symmetric tridiagonal matrix.

- **Divide:** The divide part of the divide conquer algorithm from the realization that a tridiagonal matrix is almost block diagonal.

  $T = \left(\begin{array}{c|c} T_1 & A \\ \hline B & T_2 \end{array}\right)$ .where A and B be a matrix whose elements are 0 except $(n,1)$th element of A and $(1,n)$th element of B and these elements are a.

  The size of sub-matrix T_1 is $n \times n$ and then T_2 is $(m-n) \times (m-n)$

  $T = \left(\begin{array}{c|c} T_1' & 0 \\ \hline 0 & T_2' \end{array}\right) + \left(\begin{array}{c|c} a & a \\ \hline a & a \end{array}\right)$

  The only difference between T_1 and T_1' is that the lower right entry in T_1' has been replaced with t_{nn} - a and similarly in T_2'.

- **Conquer:** First define $z^t = (q_1^t, q_2^t)$ where q_1^t is the last row of Q_1 and q_2^t is the first row of Q_2 then

  $T = \left(\begin{array}{c|} Q_1 & \\ \hline & Q_2 \end{array}\right)\left(\left(\begin{array}{c|} D_1 & \\ \hline & D_2 \end{array}\right) + azz^t\right)\left(\begin{array}{c|} Q_1^t & \\ \hline & Q_2^t \end{array}\right)$

  Now we have to find the eigenvalues of a diagonal matrix plus a rank one correction . We are looking for the eigenvalues of the matrix $(D + ww^t)$,where D is diagonal. If w is zero, (e_i, d_i) is an eigen pair of $D + ww^t$ since $(D + ww^t)e_i = De_i = d_ie_i$. If x is an eigenvalue $(D + ww^t)q = xq$ where q is the corresponding eigenvector.$(D - xI)q + w(w^tq) = 0$ which implies $w^tq + w^t(D - xI)^{-1}\ w(w^tq) = 0$. Now $w^tq \neq 0$ because if w^tq were to be 0, q would be eigenvector of D by $(D + ww^t)q = xq$ then q contain only one non zero element since D is distinct diagonal and thus the inner product w^tq can not be 0 after all.Then $1 + w^t(D - xI)^{-1}w = 0$ or written as a scalar equation

  $1 + \frac{w_1^2}{d_j - x} + \text{- - - } + \frac{w_m^2}{d_j - x} = 0.$

  This equation is known as secular equation. The problem has therefore been reduced to find the roots of the rational function defined by the left hand side of the equation.

Power iteration method:

- **Description:**

  The power iteration is an eigenvalue algorithm given a matrix A. It will find only one eigenvalue(which is the greatest absolute value eigenvalue) & it's corresponding eigenvector.

  The power iteration algorithm starts with a vector b_0, which may be an approximation of the dominant eigenvector. The method is described by the iteration $b_{k+1} = \frac{Ab_k}{||Ab_k||}$ under the assumptions,

  1) A has an eigenvalue whose absolute value is strictly greater than all eigenvalue $|x_1| > |x_i|$ for all $i > 1$,

  2) The starting vector $b_0 = c_1v_1 + c_2v_2 + ... + c_nv_n$ where c_i are real and $c_1 \neq 0$,

  3) A sub-sequence of (b_k) converges to an eigenvector associated with the dominant eigenvector.

  Let A be decomposition into it's Jordan canonical form VJV^{-1} where the first column of V is an eigenvector of A is unique, the first Jordan block of J is the 1×1 matrix $[x_1]$.

  Now

  $$b_{k+1} = \frac{Ab_k}{||Ab_k||}$$

$$= \frac{(VJ^kV^{-1})b_0}{||VJ^kV^{-1}b_0||}$$

$$= \frac{VJ^kV^{-1}(c_1v_1+c_2v_2+\cdots+c_nv_n)}{||VJ^kV^{-1}(c_1v_1+c_2v_2+\cdots+c_nv_n)||}$$

$$= \frac{x_1}{|x_1|}^k \frac{c_1}{|c_1|} \frac{v_1+\frac{\frac{v(J)^k}{x_1^k}(c_2e_2+\ldots+c_ne_n)}{c_1}}{||v_1+\frac{\frac{v(J)^k}{x_1^k}(c_2e_2+\ldots+c_ne_n)}{c_1}||}$$

$$=e^{i\theta_k}\frac{c_1}{|c_1|}V_1 + r_k, \text{ where } e^{i\theta_k} = (\frac{x_1}{|x_1|})^k$$

$$\text{As } (\tfrac{1}{x_1}J)^k = \begin{pmatrix} [1] & & & \\ & \left[\frac{1}{x_1}J_2\right]^k & & \\ & & \ddots & \\ & & & \left[\frac{1}{x_1}J_m\right]^k \end{pmatrix} \to \begin{pmatrix} 1 & & & \\ & 0 & & \\ & & 0 & \\ & & & \ddots \\ & & & & 0 \end{pmatrix}, \text{ as } k \to \infty$$

$$V_1 = \frac{v_1}{||v_1+\frac{\frac{v(J)^k}{x_1^k}(c_2e_2+\ldots+c_ne_n)}{c_1}||}$$

The sequence (b_k) is bounded. So it contains a convergent sub sequence. The presence of the term $e^{i\theta_k}$ implies that (b_k) does not converge unless $e^{i\theta_k} = 1$.

If A is diagonalizable then $A^kb_0 = c_1x_1^k(v_1 + \frac{c_2(x_2)^kv_2}{c_1(x_1)^k}v_2 + \ldots + \frac{c_m(x_m)^kv_m}{c_1(x_1)^k}v_m)$.

The expression within the parenthesis converges to v_1. Because $|\frac{x_j}{x_1}| < 1$ for $j > 1$.

Therefor (b_k) converges to the eigenvector v_1. Thus the sequence $b_{k+1} = \frac{Ab_k}{||Ab_k||}$ and $\mu_k = \frac{b_k^*Ab_k}{b_k^*b_k}$ converges to the dominant eigenvector and the dominant eigenvalue respectively under the above assumptions.

- **Note:**

  The power method is also used to find the least eigenvalue of a matrix A. If X is the eigenvector corresponding to the eigenvalue λ then $AX = \lambda X$. If A is non-singular then A^{-1} exist. Therefore, $A^{-1}(AX) = \lambda A^{-1}$ or, $A^{-1}\text{X} = \frac{1}{\lambda}\text{X}$. This means that if λ is an eigenvalue of A then $\frac{1}{\lambda}$ is an eigenvalue of A^{-1} and the same eigenvector X corresponds to the eigenvalue $1/\lambda$ of the matrix A^{-1}. Thus, if λ is largest (in magnitude) eigenvalue of A then $1/\lambda$ is the least eigenvalue of A^{-1}.

- **Applications of fixed point theory in power method for a symmetric matrix:** Let A be symmetric matrix. So the eigenvector of A orthogonal. Let $\{u_1, u_2, \cdots, u_n\}$ and $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n| \geq 0$ be the sets of of orthonormal eigenvectors and eigenvalues of A respectively. Let x$\in \mathbb{R}^n$. Because the

eigenvectors of A are orthonormal and therefore span $\mathbb{R}^n$, we can write: x=$\sum_{i=1}^{n} \alpha_i u_i$ where $\alpha_i \in \mathbb{R}$. Then we have the following

$$\text{Ax}=\sum_{i=1}^{n} \alpha_i A u_i=\sum_{i=1}^{n} \alpha_i \lambda_i u_i$$

$$A^k x = \sum_{i=1}^{n} \alpha_i \lambda_i^k u_i$$

$$A^k x=\sum_{i=1}^{n} \alpha_i \lambda_i^k u_i=\lambda_1^k \alpha_1 u_1 + \lambda_1^k \sum_{i=2}^{k} \alpha_i (\tfrac{\lambda_i}{\lambda_1})^k u_i$$

- **Contraction** Let X be a metric space. A transformation, T : $X \to X$, is called a contraction if for some $c \leq 1$, $d(Tu, Tv) \leq cd(u, v)$ for all u,v in X.

- **The Contraction Mapping Theorem:** Let T be a contraction on a complete metric space X. Then there exists an unique solution, u in X, s.t u=Tu and here u is the fixed point.

- **The contraction:** Define the transformation $Tx = \frac{1}{\lambda_1} Ax$. Let x an y be any two points in $\mathbb{R}^n$. Let us consider a set of vectors V define by

  $$V = \{y \in \mathbb{R}^n : (y.u_1) = (x_0.u_1)\}$$

  (ie, V is the set of vectors such that $\alpha_1 = \beta_1$ and let x_0 be the initial guess to the dominant vector).Let x an y be any two points in V. As u_i are orthonormal vectors so we have the following

$$\begin{aligned}
x &= \sum_{i=1}^{n} \alpha_i u_i \\
y &= \beta \sum_{i=1}^{n} \alpha_i u_i \\
Tx &= \alpha_1 u_1 + \sum_{i=2}^{n} \alpha_i (\frac{\lambda_i}{\lambda_1}) u_i \\
Tx &= \beta_1 u_1 + \sum_{i=2}^{n} \beta_i (\tfrac{\lambda_i}{\lambda_1}) u_i \\
Tx - Ty &= \alpha_1 u_1 - \beta_1 u_1 + \sum_{i=2}^{n} (\alpha_i - \beta_i)(\frac{\lambda_i}{\lambda_1}) u_i \\
\| Tx - Ty \|^2 &= (\alpha_1 - \beta_1) + \sum_{i=2}^{n} (\alpha_i - \beta_i)^2 (\frac{\lambda_i}{\lambda_1})^2 \\
\| Tx - Ty \|^2 &\leq \sum_{i=2}^{n} (\alpha_i - \beta_i)^2 (\frac{\lambda_i}{\lambda_1})^2 \\
&\leq (\frac{\lambda_2}{\lambda_1})^2 \| x - y \|^2;
\end{aligned}$$

  as x and y are in V so $\alpha_1 = \beta_1$.
  Thus we have

$$\| Tx - Ty \|^2 \leq (\frac{\lambda_i}{\lambda_1})^2 \| x - y \|^2$$

$$\text{i.e, } \| Tx - Ty \| \leq (\frac{\lambda_i}{\lambda_1}) \| x - y \|$$

$$\text{i.e, } \| Tx - Ty \| \leq k \| x - y \|$$

where $k = (\frac{\lambda_2}{\lambda_1}) < 1$, therefore T is a contraction mapping.
Here V is a closed subset of $\mathbb{R}^n$ and therefore V is also complete.Then by contraction mapping theorem, there is exactly one fixed point f s.t f=Tf. Again $Tf = \frac{1}{\lambda_1} Af$.i.e, $Af = \lambda_1 f$ hence f is the required fixed point as well as the dominant vector of the symmetric matrix.

- **Finding non-dominant eigenvalues:**Once the dominant eigenpair (λ_1, V_1) of A is computed, we may wish to compute λ_2. Now if A is symmetric, then its eigenvectors are orthogonal. Let $U_1 = \frac{V_1}{||V_1||}$, then

$$B = A - \lambda_1 U_1 U_1^T$$

i.e,

$$\begin{aligned} BV_1 &= AV_1 - \lambda_1 U_1 U_1^T V_1 \\ &= \lambda_1 V_1 - \lambda_1 \frac{V_1}{||V_1||} U_1^T V_1 \\ &= \lambda_1 V_1 (1 - U_1^T U_1) \end{aligned}$$

This implies that $U_1^T U_1 = 1$ and therefore 0 is an eigenvector of B. Since A is a symmetric matrix so for $j > 1$ we have,

$$\begin{aligned} BV_j &= AV_j - \lambda_j U_1 U_1^T V_j \\ &= \lambda_j V_j - \lambda_1 \frac{V_1}{||V_1||} U_1^T V_j \\ &= \lambda_j V_j \end{aligned}$$

as $U_1^T V_j = 0$, therefore λ_j $(j > 1)$ is also an eigenvalue of B. Therefore, to find λ_2 we could apply the power method to A. The application of the power method to A to find λ_2 is called the method of deflation.

- **Algorithm:**. This method determines the largest eigenvalue (in magnitude) and its corresponding eigenvector of a square matrix A. Algorithm Power Method
  - **Step 1.** Read the matrix A.
  - **Step 2.** Set initial vector $X_0 = (1, 1, 1, ..., 1)^T$ of n components.
  - **Step 3.** Find the product $Y = AX_0$.
  - **Step 4.** Find the largest element (in magnitude) of the vector Y and let it be λ.

- **Step 5.** Divide all the elements of Y by λ and take it as X_1, i.e., $X_1 = \frac{Y}{\lambda}$.

- **Step 6.** Let $X_0 = (x_{01}, x_{02}, ..., x_{0n})$ and $X_1 = (x_{11}, x_{12}, ..., x_{1n})$. If $|x_{0i} - x_{1i}| > \varepsilon$ for at least i then set $X_0 = X_1$ and goto Step 3.

- **Step 7.** Print λ as largest eigenvalue and corresponding eigenvector X_1 of A.

- **Program:** This program finds the largest eigenvalue (in magnitude) of a square matrix.

```
#include < stdio.h >
#include < math.h >
void main()
{
int n,i,j,flag;
float a[10][10],x0[10],x1[10],y[10],lambda,eps=1e-5;
printf("Enter the size of the matrix ");
scanf("%d",&n);
printf("Enter the elements row wise ");
for(i = 1; i <= n; i + +)for(j = 1; j <= n; j + +)scanf("%f", &a[i][j]);
printf("The given matrix is");
for(i = 1; i <= n; i + +) {
    for(j = 1; j <= n; j + +) printf("%f ",a[i][j]);
}
for(i = 1; i <= n; i + +)
   {
    x0[i]=1; x1[i]=1;
   }
do
   flag=0;
   for(i = 1; i <= n; i + +)x0[i] = x1[i];
   for(i = 1; i <= n; i + +)
     {
      y[i]=0;
      for(j = 1; j <= n; j + +) y[i]+ = a[i][j] * x0[j];
     }
   lambda=y[1];
```

$for(i = 2; i <= n; i++)\ if(lambda < y[i])lambda = y[i];$

$for(i = 1; i <= n; i++)x1[i] = y[i]/lambda;$

$for(i = 1; i <= n; i++)\ if(fabs(x0[i] - x1[i]) > eps)flag = 1;$

}while(flag==1);

printf("The largest eigenvalue is %8.5f ",lambda);

printf("The corresponding eigenvector is ");

$for(i = 1; i <= n; i++)$ printf("%8.5f ",x1[i]);

}

Bisection method for eigenvalues:

- **Description of bisection method:**Let ξ be a root of the equation f(x) = 0 lies in the interval [a, b], i.e., $f(a).f(b) < 0$, and (b-a) is not sufficiently small. The interval [a, b] is divided into two equal intervals [a, c] and [c, b], each of length $\frac{b-a}{2}$, and $c = \frac{a+b}{2}$. If $f(c) = 0$, then c is an exact root.

  Now, if $f(c) \neq 0$, then the root lies either in the interval [a, c] or in the interval [c, b]. If $f(a).f(c) < 0$ then the interval [a, c] is taken as new interval, otherwise [c, b] is taken as the next interval. Let the new interval be $[a_1, b_1]$ and use the same process to select the next new interval. In the next step, let the new interval be $[a_2, b_2]$. The process of bisection is continued until either the midpoint of the interval is a root, or the length $(b_n - a_n)$ of the interval $[a_n, b_n]$ (at nth step) is sufficiently small. The number a_n and b_n are the approximate roots of the equation $f(x) = 0$. Finally, $x_n = \frac{a_n+b_n}{2}$ is taken as the approximate value of the root .

- **Theorem:** Assume that f(x) is a continuous function on [a, b] and that there exists a number ξ [a, b] such that f(ξ) = 0. If f(a) and f(b) have opposite signs, and $x_n = \frac{a_n+b_n}{2}$ represents the sequence of midpoints generated by the bisection method, then

  $$|\xi\text{-}x_n| \leq \frac{b-a}{2^{n+1}} \text{ for n = 0, 1, 2, . . .}$$

  and therefore the sequence x_n converges to the root ξ i.e.,

  $$\lim_{x\to\infty} x_n = \xi$$

- **Description:**Bisection method is generally used for finding the eigenvalues of the symmetric tridiagonal matrix. Since the eigenvalues of real symmetric matrix are real, we can find them by searching in the real line. Now the idea is to find the roots by evaluating $p(x)$ at different points of x, without ever looking at its coefficient, applying bisection process for non-linear functions.

$$A = \begin{pmatrix} a_1 & b_1 & & & & \\ b_1 & a_2 & b_2 & & & \\ & b_2 & a_3 & b_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & & & b_{n-1} \\ & & & & b_{n-1} & a_n \end{pmatrix}, \text{ where } b_j \neq 0.$$

Let $A^{(k)}$ denotes the upper-left $k \times k$ sub-matrix of A.

- **Advantages of using symmetric matrix:**

The eigenvalues of $A^{(k)}$ are distinct as every $A^{(k)}$ is symmetric tridiagonal. Let them be denoted by $x_1^{(k)} < x_2^{(k)} <...< x_k^{(k)}$. The crucial property that makes bisection powerful is that these eigenvalues strictly interlace, satisfying the inequalities $x_j^{(k+1)} < x_j^{(k)} < x_{j+1}^{(k+1)}$ for $k = 1, 2, ...m-1$ and $j = 1, 2, 3, ..., k-1$.

Second property of symmetric tridiagonal matrix in $\mathbb{R}^{n\times n}$ which is advantageous for bisection method is the number of negative eigenvalues is equals to the number of sign changes in the sequence 1,det($A^{(1)}$),det($A^{(2)}$),...,det(. We define the sign change to mean a transition from + or 0 to − or from − or 0 to + but not from + or − to 0.

One more observation completes the description of the bisection method for tridiagonal matrix. The determinants of the matrices$A^{(k)}$ are related by a three term recurrence relation $\det(A^{(k)}) = a_k \det(A^{(k-1)}) - b_{(k-1)}^2 det(A^{(k-2)})$. Introducing the shift by xI and writing $P^{(k)}$(x)=det($A^{(k)} - xI$). We get

$$P^{(k)}(x) = (a_k - x)P^{(k-1)}(x) - b_{k-1}^2 P^{(k-2)}(x).\text{If we define } P^{(-1)}(x) = 0 \ \& \ P^0(x) = 1$$

then this recurrence is valid for $k = 1, 2, ..., m$.

- **Example:** Let us consider a symmetric matrix

$$A = \begin{pmatrix} 1 & -1 & -2 & 1 & 1 \\ -1 & 0 & 1 & 3 & 2 \\ -2 & 1 & 3 & 1 & 1 \\ 1 & 3 & 1 & 4 & 0 \\ 1 & 2 & 1 & 0 & 5 \end{pmatrix}$$

Then its tri-diagonal form is (using Householder method)

$$\begin{pmatrix} 1 & 2.64575 & 0 & 0 & 0 \\ 2.64575 & 1 & 2.03540 & 0 & 0 \\ 0 & 2.03540 - 0.58621 & 0.94489 & 0 & \\ 0 & 0 & 0.94489 & 5.44237 & -1.26864 \\ 0 & 0 & 0 & -1.26864 & 6.14384 \end{pmatrix}$$

Here

$p_1 = (1 - x)$

$p_2 = (1 - x)^2 - (2.64575)^2$

$p_3 = (7 - (1 - x)^2)(x + 0.58621) - (2.0354)^2(1 - x)$

$p_4 = ((((7-(1-x)(1-x))(x+0.58621)-(2.0354)(2.0354)(1-x))(5.44237-x))-(0.94489\times 0.94489)((1-x)(1-x)-(2.64575)(2.64575)))$

$p_5 = ((((7-(1-x)(1-x))(x+0.58621)-(2.0354)(2.0354)(1-x))(5.44237-x))-(0.94489\times 0.94489)((1-x)(1-x)-(2.64575)(2.64575)))(6.14384-x)-(1.26864\times 1.26864((7-(1-x)(1-x))(x+0.58621)-(2.0354)(2.0354)(1-x)))$

Roots P_2 are -1.64574 and 3.64574. That of P_3 are -2.76317, 0.05492, 4.12203 and that of p_4 are -2.80402, -0.03044, 4.06531, 5.62531 (Using the program).

So using the pair of consecutive roots of p_4 as interval we are going to find the required eigenvalues of the matrix A using program which is given bellow.

- **Program to find a root of an equation by bisection method. Assume that a root lies between a and b.**

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
# define f(x) ((((7-(1-x)(1-x))(x+0.58621)-(2.0354)(2.0354)(1-x))(5.44237-x))-(0.94489*0.94489)((1-x)(1-x)-(2.64575)(2.64575)))(6.14384-x)-(1.26864×1.26864((7-(1-x)(1-x))(x+0.58621)-(2.0354)(2.0354)(1-x)))
void main()
{
float a,b,fa,fb,c,fc;
float eps=1e-5;
printf("Enter the value of a and b ");
scanf("%f%f",&a,&b);
fa = f(a); fb = f(b);
if(fa*fb>0)
  {
   printf("There is no guarantee for a root within [a,b]");
  exit(0);
  }
do
{
   c = (a+b)/2;
```

```
    fc=f(c);
    if((fc == 0)|| (fabs(fc) < eps))
     {
      a=c;b=c;
     }
    else if(fb * fc > 0)
     {
      b=c; fb=fc;
     }
      else
    {
      a=c; fa=fc;
     }
}while(fabs(b - a) > eps);
printf("The desired root is %8.5f ",c);
}
```

Input: Enter the value of a and b -2.80402 -0.03044

Output

The desired root is -0.03456 Press any key to continue . . .

Input: Enter the value of a and b -0.03044 4.06531

Output

The desired root is 4.01737 Press any key to continue . . .

Input: Enter the value of a and b 4.06531 5.62531

Output

The desired root is 4.66391 Press any key to continue . . .

Therefore the three eigenvalues of the matrix A are -0.03456, 4.01737, 4.66391.

- **QR algorithm:** Let, $A = A_1$ be a square matrix and also let its QR decomposition be Q_1R_1. Now let us define another matrix $A_2 = R_1Q_1$. Next the QR factorization of A_2 be Q_2R_2. Similarly we can define A_3, A_4, A_5 ...

  where $A_k = Q_kR_k$(QR factorization of A_k), and $A_{k+1} = R_kQ_k$.

i.e $A_{k+1} = R_k Q_k = Q_k^* Q_k R_k Q_k = Q_k^* A_k Q_k = Q_k^{-1} A_k Q_k$.

Hence all A_k are similar and therefore they have the same eigenvalues as A. Also we have from above

$A_{k+1} = Q^* A_k Q_k = ... = (Q_1 Q_2 ... Q_k)^* A (Q_1 Q_2 ... Q_k)$.

If the process continued for a long time then the matrices A_k becomes upper triangular matrix (not always). Or in other word we can say that the sense $\lim_{k\to\infty} (A_k)_{ij} = 0$ for $j < i$, while the diagonal elements of the matrices A_k converge to the eigenvalues of the matrix A.

- **Theorem (Convergence of the QR algorithm):** Suppose A be a square and also suppose that A is invertible and its all its eigenvalues are distinct in modulus i.e. the algorithm gives us a Schur factorization of A. Then, there exists (at least) one invertible matrix P such that

  $A = P\Lambda P^{-1}$, with $\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_n)$, and $|\lambda_1| > |\lambda_2| > ... > |\lambda_n| > 0$. Suppose that the matrix P^{-1} has an LU factorisation. Then the sequence of matrices (A_k) is such that

$$\lim_{k\to\infty}(A_k)_{ii} = \lambda_i, 1 \le i \le n,$$
$$\lim_{k\to\infty} (A_k)_{ij} = 0, 1 \le j < i \le n.$$

  This conditions are sufficient condition to converge the iterative method. If the matrix A is symmetric then the method converge cubically. There may be several eigenvalues which are equal in modulus, then in the limiting case A_k becomes a block triangular matrix and each diagonal sub-matrix correspond to a set of eigenvalues which are equal in modulus. That is if there exists eigenvalues of some particular modulus of multiplicity p, then there will appear in matrices A_k a diagonal $sub - matrix$ of order p and at the same time the the elements of the sub-matrix may not necessarily converge, but the eigenvalues of the matrix A converges to the eigenvalues if they have some particular modulus. For example, let A be a 10×10 matrix whose eigenvalues are $\lambda_1 = \lambda_2 = \lambda_3 < \lambda_4 < \lambda_5 < \lambda_6 = \lambda_7 < \lambda_8 < \lambda_9 < \lambda_{10}$. Then in limiting case A_k will be of the form

$$A_k = \begin{pmatrix} * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * \\ & & & * & * & * & * & * & * & * \\ & & & & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * \\ & & & & & & & * & * & * \\ & & & & & & & & * & * \\ & & & & & & & & & * \end{pmatrix}.$$

Example:

A_1=**A**=$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$

$$A_1 = \left(\frac{1}{\sqrt{5}}\begin{pmatrix} 2 & -1 \\ 1 & 2 \end{pmatrix}\right)\left(\frac{1}{\sqrt{5}}\begin{pmatrix} 5 & 4 \\ 0 & 3 \end{pmatrix}\right) = Q_1 R_1.$$

A_2=$R_1Q_1 = \frac{1}{5}\begin{pmatrix}14 & 3\\ 3 & 2\end{pmatrix}$=$\begin{pmatrix}2.8 & 0.6\\ 0.6 & 1.2\end{pmatrix}$.

...

$A_4 \approx \begin{pmatrix}2.997 & -0.074\\ -0.074 & 1.0027\end{pmatrix}$.

...

$A_{10} \approx \begin{pmatrix}3 & -0.0001\\ -0.0001 & 1.0000\end{pmatrix}$.

A_{10} is almost diagonal and contains approximations to the eigenvalues $\lambda_1 = 3$ and $\lambda_2 = 1$ on the diagonal.

The method which is discussed above,it may or may not be convergent and it also can be extremely slow. The algorithm may be modified to make it converge both more quickly and for more matrices. One modification is shifting: at each step a scalar quantity s_k is chosen, then the QR factorization is done not on A_k but $A_k - s_kI$. If the scalars s_k are chosen so that they get closer and closer to an eigenvalue of the matrix, this will dramatically speed up convergence.

- **The Modified QR Method:**

Step 1: Choose s_1 be an approximation to the jth eigenvalue of the $n \times n$ matrix A. Let$A_1 = A - s_1I = Q_1R_1$ be a QR factorization of $A - s_1I$; create $A_2 = R_1Q_1 + s_1I$

Step 2: Choose s_2 to be be an approximation to the jth eigenvalue of A. Let $A_2 - s_2I = Q_2R_2$ be a QR factorization of $A_2 - s_2I$; create $A_3 = R_2Q_2 + s_2I$

Step 3: Continue this process; Once A_k has been created, choose s_k to be an approximation to the jth eigenvalue of A. Let $A_k - s_kI = Q_kR_k$ be a QR factorization of $A_k - s_kI$ and create $A_{k+1} = Q_kR_k + s_kI$.

Step 4: When the entries in the final row of A_k (except for the final entry) are tends to zero (to machine accuracy), make A_k by removing the final row and column. The final entry in the row just removed is an eigenvalue.

Step 5: Repeat Steps 1 through 4 on the matrix until all the eigenvalues have been found or until it appears that convergence to a triangular limit matrix will not happen. The method is also known as shift QR method and s_k is called the shift at kth step.

The rate convergence depends on choice of s_k. If s_k be too closer to the eigenvalue then convergence rate will be more faster. Some best choice for s_k are given bellow

1) The shift s_k can be chosen the last entry of diagonal of A_k in each step.

2) The shift $s_k = e_n^T A_K e_n$ is called the Rayleigh quotient shift.

3) The eigenvalue of the lower right 2×2 corner of A_k closest to the (n, n) element of A_k is called the Wilkinson shift. This shift can be used to find complex eigenvalues of a real matrix. The convergence is very fast and at least quadratic both for the Rayleigh quotient shift and the Wilkinson shift. If λ_1; λ_2 are eigenvalues of

$$\begin{pmatrix} a_{n-1,n-1} & a_{n-1,n} \\ a_{n,n-1} & a_{n,n} \end{pmatrix}$$

with $|\lambda_1 - a_{n,n}| \leq |\lambda_2 - a_{n,n}|$ then $shift = \lambda_1$.

Algorithm:

Let $A_0 = A$, we iterate

i=0

repeat

Choose a shift s_i

$A_i - s_i I = Q_i R_i$ (QR decomposition)

$A_{i+1} = R_i Q_i + s_i I$

$i = i + 1$

until convergence

- **Hessenberg Reduction:** Note that the QR decomposition in the algorithm takes $O(n^3)$ flops. Even if the algorithm took n iterations to converge, the overall cost of the algorithm will be $O(n^4)$ This is too expensive today the complexity of algorithms for all standard matrix computation problems is at $O(n^3)$.

  However, if the matrix is initially reduced to upper Hessenberg form, then the QR decomposition of a Hessenberg form costs $O(n^3)$ flops. As a result the overall cost of the algorithm is reduced to $O(n^3)4$

  An upper Hessenberg matrix H is the matrix whose elements below the sub diagonal are all zero.

$$\text{H=}\begin{pmatrix} * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ & * & * & * & * & * & * \\ & & * & * & * & * & * \\ & & & * & * & * & * \\ & & & & * & * & * \end{pmatrix}$$

- **Proposition:** For any real square matrix A,there is an orthogonal matrix Q such that $Q^T AQ = H$ an upper Hessenberg matrix.

- **An observation:** Instead of finding $A_{i+1} = R_iQ_i + s_iI$ we can only find $A_{i+1} = R_iQ_i$ but for the eigenvalue we have to add all the shifts which already being used.

- **Example of the QR method with shifts to find the eigenvalues of A where:**

$$\text{A}= \begin{pmatrix} 4 & 2 & 2 & 1 \\ 2 & -3 & 1 & 1 \\ 2 & 1 & 3 & 1 \\ 1 & 1 & 1 & 2 \end{pmatrix}$$

First we transform A to tri-diagonal matrix using Hessenberg form. We have the tri-diagonal Hessenberg matrix is

$$\text{H}= \begin{pmatrix} 4 & -3 & 0 & 0 \\ -3 & 2 & 3.1622777 & 0 \\ 0 & 3.1622777 & -1.4 & -0.2 \\ 0 & 0 & -0.2 & 1.4 \end{pmatrix}$$

Step I: We choose a shift $s_1 = 1.4$, which is $(4, 4)$ element of H. Now we are going to find the QR factorization of $H - s_1I$ i.e. $H - s_1I = Q_1R_1$

$$\text{where } Q_1= \begin{pmatrix} -0.6549305 & -0.3852774 & -0.5660663 & 0.3196814 \\ 0.7556891 & -0.3339071 & -0.4905908 & 0.2770572 \\ 0 & 0.8602717 & -0.4439346 & 0.2507085 \\ 0 & 0 & 0.4917434 & 0.8707402 \end{pmatrix}$$

$$R_1= \begin{pmatrix} -3.9698866 & 2.4182051 & 2.3896987 & 0 \\ 0 & 3.6759059 & -3.4646677 & -0.1720543 \\ 0 & 0 & -0.4067162 & 0.0887869 \\ 0 & 0 & 0 & -0.0501417 \end{pmatrix}$$

Step II: For next step we have $A_2 = R_1Q_1$. Therefore

$$A_2 = R_1Q_1= \begin{pmatrix} 4.4274112 & 2.777842 & 0 & 0 \\ 2.777842 & -4.2079666 & -0.3498864 & 0 \\ 0 & -0.3498864 & 0.2242158 & -0.0246568 \\ 0 & 0 & -0.0246568 & -0.0436604 \end{pmatrix}$$

Similarly now we have to find out the QR factorization of $A_2 - s_2I = Q_2R_2$ (say), where $s_2 = -0.0436604$. Then we have

$$Q_2= \begin{pmatrix} -0.8494108 & 0.5264465 & -0.0366818 & 0.0031409 \\ -0.5277323 & -0.8473413 & 0.0590411 & -0.0050555 \\ 0 & -0.0697627 & -0.9939266 & 0.0851068 \\ 0 & 0 & 0.0853147 & 0.9963541 \end{pmatrix}$$

$$R_2 = \begin{pmatrix} -5.2637332 & -0.1618900 & 0.1846463 & 0 \\ 0 & 5.0153827 & 0.2777854 & 0.0017201 \\ 0 & 0 & -0.2890105 & 0.0245071 \\ 0 & 0 & 0 & 7 - 0.0020985 \end{pmatrix}$$

Step III: $A_3 = \text{RQ} = \begin{pmatrix} 4.5565062 & -2.6467794 & 0 & 0 \\ -2.6467794 & -4.2691198 & 0.0201621 & 0 \\ 0 & 0.0201621 & 0.2893460 & -0.0001790 \\ 0 & 0 & -0.0001790 & -0.0020908 \end{pmatrix}$

and then $A_3 - s_3 I = Q_3 R_3$, where $s_3 = -0.0020908$ where

$$Q_3 = \begin{pmatrix} -0.8648011 & -0.5021105 & -0.0020170 & 0.0000012 \\ 0.5021145 & -0.8647941 & -0.0034739 & 0.0000021 \\ 0 & 0.0040170 & -0.9999917 & 0.0006141 \\ 0 & 0 & 0.0006142 & 0.9999998 \end{pmatrix}$$

$$R_3 = \begin{pmatrix} -5.2712662 & 0.1464005 & 0.0101237 & 0 \\ 0 & 5.0191584 & -0.0162654 & -0.0000007 \\ 0 & 0 & -0.2915046 & 0.0001790 \\ 0 & 0 & 0 & -0.0000001 \end{pmatrix}$$

Step IV: Similarly $A_4 = R_3 Q_3 = \begin{pmatrix} 4.6321068 & 2.5201924 & 0 & 0 \\ 2.5201924 & -4.3406042 & -0.0011710 & 0 \\ 0 & -0.0011710 & 0.2915023 & 0 \\ 0 & 0 & 0 & -0.0000001 \end{pmatrix}$

Hence we obtain A_4 whose all elements in the last column and last row are zero except at the position (4,4). Thus we have obtain an eigenvalue and which is $s_1 + s_2 + s_3 - 0.0000001 = 1.3584303$

Step V: Now we have do same thing for remaining 3×3 matrix A_4 by deleting the last row and column and let us denote the matrix by

$$B_1 = \begin{pmatrix} 4.6321068 & 2.5201924 & 0 \\ 2.5201924 & -4.34060427 - 0.001171 & \\ 0 & -0.001171 & 0.2915023 \end{pmatrix}$$

For shift we choose $s_{11} = 0.2915023$. Now we have to find out the QR factorization of $B_1 - s_{11} = Q_4 R_4$. We have

$$Q_4 = \begin{pmatrix} -0.8648027 & 0.5021119 & 0.0001115 \\ -0.5021119 & -0.8648026 & -0.0001921 \\ 0 & -0.0002221 & 1.0000000 \end{pmatrix}$$

$$R_4 = \begin{pmatrix} -5.0191849 & 0.1463666 & 0.0005880 \\ 0 & 5.2712767 & 0.0010127 \\ 0 & 0 & 0.0000002 \end{pmatrix}$$

Step VI: $B_2 = R_4Q_4 = \begin{pmatrix} 4.2671121 & -2.6467707 & 0 \\ -2.6467707 & -4.5586143 & 0 \\ 0 & 0 & 0.0000002 \end{pmatrix}$

Hence we have obtain the 2nd eigenvalue which is $s_1 + s_2 + s_3 - 0.0000001 + s_{11} + 0.0000002 = 1.6499326$.

Step vII: The next step is similar as Step V. Choosing $C_1 = \begin{pmatrix} 4.2671121 & -2.6467707 \\ -2.6467707 & -4.5586143 \end{pmatrix}$ and choosing $s_{111} = -4.5586143$ and the QR factorization of $C_1 - s_{111} = Q_5R_5$ (say), where $s_{111} = -4.5586143$

$$Q_5 = \begin{pmatrix} -0.9578546 & 0.2872536 \\ 0.2872536 & 0.9578546 \end{pmatrix} \text{ and}$$

$$R_5 = \begin{pmatrix} -9.2140567 & 2.5352214 \\ 0 & -0.7602943 \end{pmatrix}$$

Step VIII: $C_2 = R_5Q_5 = \begin{pmatrix} 9.5539778 & -0.2183973 \\ -0.2183973 & -0.7282514 \end{pmatrix}$ Now the QR factorization of $C_2 - s_{222} = Q_6R_6$ where $s_{222} = -0.7282514$ and

$$Q_6 = \begin{pmatrix} -0.9997745 & 0.0212355 \\ 0.0212355 & 0.9997745 \end{pmatrix}$$

$$R_6 = \begin{pmatrix} -10.284548 & 0.2183480 \\ 0 & -0.0046378 \end{pmatrix}$$

Step IX $C_3 = \begin{pmatrix} 10.286866 & -0.0000985 \\ -0.0000985 & -0.0046367 \end{pmatrix}$

and here $s_{333} = -0.0046367$ and the QR factorization of $C_3 - s_{333}$ is given by

$$Q_7 = \begin{pmatrix} -1 & 0.0000096 \\ 0.0000096 & 1 \end{pmatrix}$$

$$R_7 = \begin{pmatrix} -10.291503 & 0.0000985 \\ 0 & 0 \end{pmatrix}$$

Thus $C_4 = R_7Q_7 = \begin{pmatrix} 10.291503 & 0 \\ 0 & 0 \end{pmatrix}$

So the third eigenvalue is $s_1 + s_2 + s_3 - 0.0000001 + s_{11} + 0.0000002 + s_{111} + s_{222} + s_{333} = -3.6415698$ and the last one is 10.291503-3.6415698=6.6499332. Hence the required eigenvalues are 6.6499332, -3.6415698, 1.6499326, 1.3584303.

Conclusion:

While there is no simple algorithm to directly calculate eigenvalues for general matrices, there are numerous special classes of matrices where eigenvalues can be directly calculated. The purpose of this paper was to provide an overview of some of the numeric techniques used to compute eigenvalues and eigenvectors of matrices. These methods are all based on simple ideas that were steadily generalized and adapted until they became powerful iterative algorithms. In our project, We discussed about some iterative methods for finding eigenvalues of a matrix. There are many methods for finding eigenvalues of a matrix, but here we discussed about six methods which are Jacobi's method, Schur factorization, bisection method, QR method, Power method and Divide and conquer method.

We see that the three methods bisection method, Divide and conquer method, Jacobi's method applicable for symmetric matrices and the other methods Power method, QR method are applicable all types of matrices. Also the power method gives us only the dominant eigenvalue and corresponding eigenvector.

However, even the QR iteration method as presented is generally not suitable for use in practice, even in the situations when it can be applied. There are many ways to refine the algorithms we have seen in order to speed up the implementations. For example, before applying the QR algorithm, it is common to reduce the matrix A into a simpler form, such as a tridiagonal matrix: performing QR decompositions on the new matrix then becomes much faster. It is also possible to improve the QR iteration method by incorporating shifts and Rayleigh quotients, just as these concepts helped improve the original power iteration method.

As with all algorithms, the question of improvement is always an open problem. Modern information processing deals with increasingly large matrices, and efficient methods for computing eigenvalues and eigenvectors are extremely necessary. The techniques of the future may become much more complicated in order to keep up with this growing demand, but as we have seen, there is a lot that can be done using simple ideas and well-chosen generalizations. Iterative algorithms solve the eigenvalue problem by producing sequences that converge to the eigenvalues. Some algorithms also produce sequences of vectors that converge to the eigenvectors. Now let us look at the following table

Method	Applies to	Produces	Cost per step	Convergence	Description
Power iteration	General	eigenpair with largest value	$O(n^2)$	Linear	Repeatedly applies the matrix to an arbitrary starting vector and re normalizes
Bisection method	Real Symmetric Tridiagonal	any eigenvalue		linear	Uses the bisection method to find roots of the characteristic polynomial, supported by the Sturm sequence.
Jacobi eigenvalue algorithm	Real Symmetric	all eigenvalues	$O(n^3)$	quadratic	Uses Givens rotations to attempt clearing all off-diagonal entries. This fails, but strengthens the diagonal.
QR algorithm	General	1)Hessenberg, 2) all eigenvalues	1)$O(n^2)$, 2)$6n^3 + O(n^2)$	cubic	Factors A = QR, where Q is orthogonal and R is triangular, then applies the next iteration to RQ. all eigenpairs
Divide-and-conquer Hermitian	Tridiagonal	1) all eigenvalues, 2) all eigenpairs,	1) $O(n^2)$, 2) $(\frac{4}{3})n^3 + O(n^2)$		Divides the matrix into sub matrices that are diagonalized then recombined.

Therefore we can conclude that to find the eigenvalues of a matrix

- First transform the matrix to a suitable form, like tridiagonal form,
- To calculate only eigenvalues: use QR,
- Small eigenvalues with high accuracy: use Jacobi,
- Eigenvalues in the interval [a,b]: use bisection,
- For dominant eigenvalues and dominant eigenvector: use power method.

References:

1) `http://www.wikibooks.org`

2) Applied linear algebra by James Demmel

3) Numerical linear algebra by Lloyd N. Trefethen, David Bau