Eigensystem

In physics, the needs for calculating eigenvalues and eigenvectors appear in widely varying systems. Their importance in numerical determination perhaps comes just next to Monte Carlo methods. The standard way to express an eigenvalue problem of a $n \times n$ matrix $\mathbf{A}$ is,

$$\mathbf{A} \cdot \mathbf{x} = \lambda \mathbf{x} \;\text{ or }\; \mathbf{A}\,\mathbf{x}_\nu = \lambda_\nu\,\mathbf{x}_\nu, \;\; \forall \nu \in \{1, 2, \ldots, n\} \tag{1}$$

where λ_ν are the eigenvalues and $\mathbf{x}_\nu$ the corresponding eigenvectors. For the purpose of solving the above eigenvalue problem, we rewrite it as,

$$\Big(\mathbf{A} - \lambda_\nu \mathbb{1}\Big)\,\mathbf{x}_\nu = 0 \tag{2}$$

with $\mathbb{1}$ being the identity matrix. The above equation has nonzero solution $\mathbf{x}$ if the matrix $\mathbf{A} - \lambda\mathbb{1}$ is singular,

$$p_A(\lambda) = \det\Big(\mathbf{A} - \lambda_\nu \mathbb{1}\Big) = 0. \tag{3}$$

The function $p_A(\lambda)$ is called the characteristic polynomial of degree n of $\mathbf{A}$. From the Fundamental Theorem of Algebra it follows that there are n roots of the above characteristic equation $p_A(\lambda) = 0$ *i.e.* n eigenvalues of $\mathbf{A}$ (may or may not be real or distinct). $p_A(\lambda)$ is a monic polynomial,

$$p_A(\lambda) = \lambda^n + c_{n-1}\lambda^{n-1} + \cdots + c_2\lambda^2 + c_1\lambda + c_0 = \prod_{i=1}^{k} (\lambda_i - \lambda)^{\alpha_i} \tag{4}$$

where λ_i are all distinct eigenvalues of $\mathbf{A}$ and α_i are the corresponding multiplcities. However, computing eigenvalues using characteristic polynomial is generally not recommended, roots of polynomials $n > 4$ cannot always be computed in finite number of steps and requires iterative process. One word of wisdom before you start to determine eigenvalues numerically – *in general the eigenvalues and eigenvectors of a matrix cannot be determined using a finite number of rational operations.* They have to be approximate!

Solving eigenvalue problems numerically is a difficult undertaking with respect to stability, convergence and computing time. One generally makes use of various properties of the matrix $\mathbf{A}$, such as symmetric, Hermitian, normal, unitary, tridiagonal, Hessenberg, sparse etc. and design algorithms accordingly. The choice of algorithm is also affected by whether you need all or only a few eigenvalues, eigenvectors needed or not and so on. Other concerns are degenerate eigenvalues, zero or extreme eigenvalues and many such. There is no one-size-fits-all eigen algorithm, if you forget this then do so at your own peril!

A brief aside : there are two different kind of eigenvectors – right and left. For most of our needs it is sufficient to consider the right eigenvectors and the term just *eigenvector* will always be the right eigenvector.

$$\mathbf{A} \cdot \mathbf{x}_R = \lambda_R \mathbf{x}_R \;\Rightarrow\; \det(\mathbf{A} - \lambda_R \mathbb{1}) = 0 \tag{5}$$

$$\mathbf{x}_L \cdot \mathbf{A} = \lambda_R \mathbf{x}_L \;\Rightarrow\; \det\left(\mathbf{A}^T - \lambda_L \mathbb{1}\right) = 0 \tag{6}$$

The left and right eigenvalues are equal $\lambda_L = \lambda_R = \lambda$ but it is not true in general for eigenvectors (`prove it`). If $\mathbf{A}$ is symmetric, then left and right eigenvectors are each other's transpose and, if $\mathbf{A}$ is Hermitian then they are adjoint.

Additionally, there is *generalized eigenvalue problem* (GEVP) involving two symmetric matrices $\mathbf{A}$ and $\mathbf{B}$ defined through,

$$\mathbf{A} \cdot \phi_\nu = \lambda_\nu \mathbf{B} \phi_\nu \;\Rightarrow\; \mathbf{A} \cdot \boldsymbol{\phi} = \mathbf{B}\boldsymbol{\phi}\boldsymbol{\Lambda} \tag{7}$$

where the later is in matrix form, $\boldsymbol{\phi} = \{\phi_1, \ldots, \phi_n\}$ and $\boldsymbol{\Lambda} = \{\lambda_1, \ldots, \lambda_n\}^T$. Assuming $\mathbf{B}$ to be positive definite (has all positive eigenvalues), the eigenvectors ϕ can be made to satisfy

$$\phi_\nu^T \mathbf{B} \phi_\mu = \delta_{\nu\mu} \;\text{ and }\; \boldsymbol{\Lambda} = \boldsymbol{\phi}^T \mathbf{A} \boldsymbol{\phi} \tag{8}$$

If we perform a Cholesky decomposition of $\mathbf{B}$ *i.e.* $\mathbf{B} = \mathbf{L}\mathbf{L}^T$, then using orthonormality condition (8) we get,

$$\boldsymbol{\phi}^T \mathbf{B} \boldsymbol{\phi} = \mathbb{1} \quad \Rightarrow \quad \boldsymbol{\phi}^T L L^T \boldsymbol{\phi} \equiv \boldsymbol{\Psi}^T \boldsymbol{\Psi} = \mathbb{1} \quad \Rightarrow \quad \boldsymbol{\phi} = \mathbf{L}^{-T} \boldsymbol{\psi} \tag{9}$$

From the definition of GEVP (7) it follows that,

$$\begin{aligned} \mathbf{A} \cdot \boldsymbol{\phi} &= \mathbf{B} \boldsymbol{\phi} \boldsymbol{\Lambda} \\ \mathbf{A} \mathbf{L}^{-T} \boldsymbol{\psi} &= \mathbf{L} \mathbf{L}^T \mathbf{L}^{-T} \boldsymbol{\Psi} \boldsymbol{\Lambda} \\ \mathbf{L}^{-1} \mathbf{A} \mathbf{L}^{-T} &= \boldsymbol{\Psi} \boldsymbol{\Lambda} \\ \Rightarrow \; \tilde{\mathbf{A}} \boldsymbol{\Psi} &= \boldsymbol{\Psi} \boldsymbol{\Lambda} \end{aligned} \tag{10}$$

implying GEVP has been transformed to a regular eigenvalue problem through Cholesky decomposition of $\mathbf{B}$. This is the simplest way to solve GEVP provided $\mathbf{A}$ and $\mathbf{B}$ are symmetric matrices.

Before we headed towards discussing algorithm for eigenvalue problems, an important point to note is that *small changes in* $\mathbf{A}$ *do not necessarily lead to small changes in the eigenvalues of the* $\mathbf{A}$. For instance,

$$\begin{aligned} \mathbf{A} &= \begin{pmatrix} 1 & 1000 \\ 0 & 1 \end{pmatrix} \qquad \text{and } \lambda_A = 1,\ 1 \\ \mathbf{B} &= \begin{pmatrix} 1 & 1000 \\ 0.001 & 1 \end{pmatrix} \quad \text{and } \lambda_B = 0,\ 2 \end{aligned} \tag{11}$$

The above example tells us that when (rounding) errors get into any of the entries of $\mathbf{A}$, then exercise extra caution while using their eigenvalues. However, if $\mathbf{A}$ is symmetric, small changes do not lead to large changes in the eigenvalues. Hence, the algorithms dealing with the symmetric matrices are in general pretty successful.

To determine the errors in eigenvalues, define Frobenius norm of a matrix $\mathbf{A}$ as

$$||\mathbf{A}||_F = \sqrt{\sum_{1 \le i,j \le n} |a_{ij}|^2} \tag{12}$$

where $\mathbf{A}$ is an $n \times n$ real, symmetric matrix. Let $\mathbf{E}$ be $n \times n$ *error matrix*, again real and symmetric, and suppose $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{E}$. If λ_ν and $\hat{\lambda}_\nu$ are the eigenvalues of the $\mathbf{A}$ and its error version $\hat{\mathbf{A}}$ respectively, then

$$(\lambda_1 - \hat{\lambda}_1)^2 + (\lambda_2 - \hat{\lambda}_2)^2 + \cdots + (\lambda_n - \hat{\lambda}_n)^2 \le ||\mathbf{E}||_F^2 \quad \text{and} \quad |\lambda_\nu - \hat{\lambda}_\nu| \le ||\mathbf{E}||_F \tag{13}$$

The above condition is known as stability theorem which implies the process of finding λ_ν is stable *i.e.* small errors in $\mathbf{A}$ result in small errors in λ_νs. Let us illustrate this with an example. Consider the following matrix, its error version and, the corresponding error matrix,

$$\mathbf{A} = \begin{pmatrix} 1 & -1 & 2 \\ -1 & 2 & 7 \\ 2 & 7 & 5 \end{pmatrix}, \;\; \hat{\mathbf{A}} = \begin{pmatrix} 1.01 & -1.05 & 2.1 \\ -1.05 & 1.97 & 7.1 \\ 2.1 & 7.1 & 4.9 \end{pmatrix} \Rightarrow \mathbf{E} = \begin{pmatrix} 0.01 & -0.05 & 0.1 \\ -0.05 & -0.03 & 0.1 \\ 0.1 & 0.1 & -0.1 \end{pmatrix} \tag{14}$$

Hence, the Frobenius norm of $\mathbf{E}$ and maximum error in λ_ν are,

$$\begin{aligned} ||\mathbf{E}||_F &= \sqrt{(0.01)^2 + 2(-0.05)^2 + 4(0.1)^2 + (0.03)^2 + (-0.1)^2} = 0.2366\ldots \quad (15) \\ |\lambda_\nu - \hat{\lambda}_\nu| &\leq 0.2366 \quad (16) \end{aligned}$$

Power method : The power method (or von Mises iteration) is an eigenvalue algorithm which returns the largest $|\lambda|$ and the corresponding eigenvector v. This is rather a simple algorithm and may converge slowly. Despite being simple, this algorithm was part of the Google's first version of PageRank algorithm (ref. Wikipedia). The power method is used when

1. $n \times n$ matrix $\mathbf{A}$ has n linearly independent eigenvectors, and
2. eigenvalues can be ordered in magnitude : $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$. The λ_1 is called the dominant eigenvalue and the corresponding eigenvector is the dominant eigenvector of $\mathbf{A}$.

Let $\mathbf{x}_0$ be the initial guess for the dominant eigenvector, then

$$\mathbf{x}_0 = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n \quad (17)$$

where $\{\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_n\}$ is a set of linearly independent eigenvectors. Then by repeated multiplication by $\mathbf{A}$ we form the following set of equations,

$$\begin{aligned} \mathbf{x}_1 &= \mathbf{A}\mathbf{x}_0 = c_1\lambda_1\mathbf{v}_1 + c_2\lambda_2\mathbf{v}_2 + \cdots + c_n\lambda_n\mathbf{v}_n \\ \mathbf{x}_2 &= \mathbf{A}^2\mathbf{x}_0 = c_1\lambda_1^2\mathbf{v}_1 + c_2\lambda_2^2\mathbf{v}_2 + \cdots + c_n\lambda_n^2\mathbf{v}_n \\ &\vdots \\ \mathbf{x}_k &= \mathbf{A}^k\mathbf{x}_0 = c_1\lambda_1^k\mathbf{v}_1 + c_2\lambda_2^k\mathbf{v}_2 + \cdots + c_n\lambda_n^k\mathbf{v}_n \quad (18) \end{aligned}$$

Dividing the last equation in (18) by λ_1^k, we get

$$\frac{\mathbf{x}_k}{\lambda_1^k} = \frac{1}{\lambda_1^k}\mathbf{A}^k\mathbf{x}_0 = c_1\mathbf{v}_1 + \left(\frac{\lambda_2}{\lambda_1}\right)^k c_2\mathbf{v}_2 + \cdots + \left(\frac{\lambda_n}{\lambda_1}\right)^2 c_n\mathbf{v}_n \approx c_1\mathbf{v}_1 \quad (19)$$

$\mathbf{x}_0$ has to be so chosen such that it is not orthogonal to $\mathbf{v}_1$ which ensures $c_1 \neq 0$. The next iteration of (19) will give us

$$\frac{\mathbf{x}_{k+1}}{\lambda_1^{k+1}} = \frac{1}{\lambda_1^{k+1}}\mathbf{A}^{k+1}\mathbf{x}_0 = c_1\mathbf{v}_1 \quad (20)$$

Let $\mathbf{y}$ be any vector not orthogonal to $\mathbf{v}_1$ (no problem if it is identical to $\mathbf{x}_0$), then

$$\begin{aligned} \frac{\mathbf{x}_k \cdot \mathbf{y}}{\lambda_1^k} &= \frac{1}{\lambda_1^k}\mathbf{A}^k\mathbf{x}_0 \cdot \mathbf{y} = c_1\mathbf{v}_1 \cdot \mathbf{y} \\ \frac{\mathbf{x}_{k+1} \cdot \mathbf{y}}{\lambda_1^{k+1}} &= \frac{1}{\lambda_1^{k+1}}\mathbf{A}^{k+1}\mathbf{x}_0 \cdot \mathbf{y} = c_1\mathbf{v}_1 \cdot \mathbf{y} \quad (21) \end{aligned}$$

Equating the two, we obtain the *Rayleigh quotient* formula for determining the largest eigenvalue λ_1,

$$\frac{\mathbf{x}_{k+1} \cdot \mathbf{y}}{\mathbf{x}_k \cdot \mathbf{y}} = \frac{\mathbf{A}^{k+1}\mathbf{x}_0 \cdot \mathbf{y}}{\mathbf{A}^k\mathbf{x}_0 \cdot \mathbf{y}} = \lambda_1 \quad (22)$$

The convergence to *true* λ_1 depends on how $(\lambda_2/\lambda_1)^k$ is decreasing with k. The next is the eigenvector $\mathbf{v}_1$ calculation. Generally, the sequence of vectors given by

$$\mathbf{A}\mathbf{x}_0,\ \mathbf{A}^2\mathbf{x}_0,\ \ldots,\ \mathbf{A}^k\mathbf{x}_0,\ \ldots \ \rightarrow \ \mathbf{v}_1 \quad (23)$$

approaches a multiple of the dominant eigenvector of $\mathbf{A}$. This is not difficult to see by considering the eigenvalue equation for the dominant eigenvalue λ_1,

$$\mathbf{A}\mathbf{v}_1 = \lambda_1 \mathbf{v}_1 \quad \Rightarrow \quad \lambda_1 = \frac{\mathbf{A}\mathbf{v}_1 \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1} \tag{24}$$

If we choose $\mathbf{y} = \mathbf{A}^k \mathbf{x}_0$ then from (22) it follows that,

$$\lambda_1 = \frac{\mathbf{A}\mathbf{A}^k\mathbf{x}_0 \cdot \mathbf{A}^k\mathbf{x}_0}{\mathbf{A}^k\mathbf{x}_0 \cdot \mathbf{A}^k\mathbf{x}_0} \equiv \frac{\mathbf{A}\mathbf{v}_1 \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1} \tag{25}$$

Therefore, it stands to reason that $\mathbf{A}^k\mathbf{x}_0$ is approximately an eigenvector $\mathbf{v}_1$ corresponding to λ_1.
Although it is not recommended, but one can attempt to find non-dominant eigenvalues by defining,

$$\boldsymbol{\mathcal{A}} = \mathbf{A} - \lambda_1 \mathbf{U}_1 \mathbf{U}_1^T, \quad \text{where,} \quad \mathbf{U}_1 = \mathbf{v}_1/|\mathbf{v}_1| \tag{26}$$

which has eigenvalues $0, \lambda_2, \lambda_3, \ldots, \lambda_n$ and the eigenvectors of $\boldsymbol{\mathcal{A}}$ are eigenvectors of $\mathbf{A}$. Depending on accuracy achieved for λ_1, there can be pretty large errors that can creep into λ_2. Pressing this method even further for other eigenvalues is absolutely inadvisable. In any case, the application of power method to $\boldsymbol{\mathcal{A}}$ to find $\lambda_2, \lambda_3, \ldots$ is called the *method of deflation.*

Jacobi method : Suppose we want to know all (or most) eigenvalues and are not interested in the eigenvectors. Then we can take the approach of repeated similarity transformations on $\mathbf{A}$ so as to render it either diagonal or tridiagonal form. The way to similarity transform the matrix $\mathbf{A}$, assuming it is real and symmetric, is

$$\boldsymbol{\mathcal{S}}^T \mathbf{A} \boldsymbol{\mathcal{S}} = \text{diag}\left(\lambda_1, \lambda_2, \ldots, \lambda_n\right), \quad \text{where } \boldsymbol{\mathcal{S}}^T\boldsymbol{\mathcal{S}} = \boldsymbol{\mathcal{S}}^{-1}\boldsymbol{\mathcal{S}} = \mathbb{1} \tag{27}$$

We know that similarity transformation does not change the eigenvalues of a matrix but does change its eigenvectors to $\boldsymbol{\mathcal{S}}^T\mathbf{x}$. The basic philosophy is to apply series of $\boldsymbol{\mathcal{S}}$ to render $\mathbf{A}$ diagonal

$$\boldsymbol{\mathcal{S}}_N^T \boldsymbol{\mathcal{S}}_{N-1}^T \cdots \boldsymbol{\mathcal{S}}_2^T \boldsymbol{\mathcal{S}}_1^T \; \mathbf{A} \; \boldsymbol{\mathcal{S}}_1 \boldsymbol{\mathcal{S}}_2 \cdots \boldsymbol{\mathcal{S}}_{N-1} \boldsymbol{\mathcal{S}}_N = \mathbf{D} \tag{28}$$

We will achieve this by Jacobi's method or *Given's rotation.* Consider an $n \times n$ orthogonal transformation matrix

$$\boldsymbol{\mathcal{S}} = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \vdots & \vdots & \vdots & \ddots & 0 & \vdots \\ 0 & 0 & \cdots & \cos\theta & 0 & \cdots & \sin\theta & 0 \\ 0 & 0 & \cdots & 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & \vdots & \vdots & \vdots & \ddots & 0 & 0 \\ 0 & 0 & \cdots & -\sin\theta & 0 & \cdots & \cos\theta & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 1 \end{pmatrix}, \quad \text{where, } \boldsymbol{\mathcal{S}}^T = \boldsymbol{\mathcal{S}}^{-1} \tag{29}$$

This is simply a rotation matrix in n-dimensional plane, the idea is to kill off symmetric pairs of matrix entries in $\mathbf{A}$ eventually converging to a diagonal form. The $\sin\theta$ and $\cos\theta$ appear at the intersections of the k-th and l-th rows and columns ($k > l$). For a fixed (k, l), the non-zero elements of Givens matrix are

$$s_{ii} = 1 \;\text{ for } i \neq k, l \qquad s_{ii} = \cos\theta \;\text{ for } i = k, l \qquad \text{and} \quad s_{kl} = -s_{lk} = -\sin\theta \tag{30}$$

A similarity transformation with $\boldsymbol{\mathcal{S}}$ above in (29) $\mathbf{B} = \boldsymbol{\mathcal{S}}^T \mathbf{A} \boldsymbol{\mathcal{S}}$ results in

$$\begin{aligned}
B_{ii} &= A_{ii} \quad \text{for } i \neq k, l \\
B_{kk} &= A_{kk}\cos^2\theta + A_{ll}\sin^2\theta - 2A_{kl}\cos\theta\sin\theta \\
B_{ll} &= A_{ll}\cos^2\theta + A_{kk}\sin^2\theta + 2A_{lk}\cos\theta\sin\theta \\
B_{ik} &= A_{ik}\cos\theta - A_{il}\sin\theta \quad \text{for } i \neq k, l \\
B_{il} &= A_{il}\cos\theta + A_{ik}\sin\theta \quad \text{for } i \neq k, l \\
B_{kl} &= (A_{kk} - A_{ll})\cos\theta\sin\theta + A_{kl}\left(\cos^2\theta - \sin^2\theta\right)
\end{aligned} \tag{31}$$

The θ is arbitrary and is chosen to make all non-diagonal elements $B_{ij} \to 0$. In practice, it reduces systematically the norm of the off-diagonal elements A_{ij},

$$\sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}^2} \;\to\; 0, \quad i \neq j \tag{32}$$

The most trivial demonstration of the Givens rotation is with 2×2 matrix,

$$\begin{aligned}
\boldsymbol{\mathcal{S}}^T \mathbf{A} \boldsymbol{\mathcal{S}} &= \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \\
&= \begin{pmatrix} a_{11}c^2 - 2a_{12}cs + a_{22}s^2 & a_{12}c^2 + (a_{11} - a_{22})cs - a_{12}s^2 \\ a_{12}c^2 + (a_{11} - a_{22})cs - a_{12}s^2 & a_{22}c^2 + 2a_{12}cs + a_{11}s^2 \end{pmatrix}
\end{aligned} \tag{33}$$

where $c = \cos\theta$, $s = \sin\theta$ and $a_{12} = a_{21}$. To render the matrix $\mathbf{A}$ diagonal, choose θ such that it satisfies the following

$$a_{12}c^2 + (a_{11} - a_{22})cs - a_{12}s^2 = 0 \quad \Rightarrow \quad 1 + \frac{a_{11} - a_{22}}{a_{12}}t - t^2 = 0 \tag{34}$$

where $t = \tan\theta = s/c$. As an example,

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \quad \text{for } \theta = -\pi/4 \;\to\; \begin{pmatrix} 3 & 0 \\ 0 & -1 \end{pmatrix} \tag{35}$$

The Jacobi method using Givens rotation is a reliable, simple to code, relatively accurate algorithm, for sparse matrices have low arithmetic operations but converges extremely slowly and difficult to extend beyond symmetric matrices. One source of inefficiency is that previously annihilated entries can subsequently become nonzero again and hence require repeated annihilation. Say,

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 2 & 1 \\ 2 & 1 & 1 \end{pmatrix} \quad \text{and } \theta = 45^o\,(1,3),\ 27^o\,(1,2),\ 13^o\,(2,3),\ 4^o\,(1,3) \text{ etc.} \tag{36}$$

Repeated application of $\boldsymbol{\mathcal{S}}$, as in (28) yields,

$$\begin{aligned}
\begin{pmatrix} 1 & 0 & 2 \\ 0 & 2 & 1 \\ 2 & 1 & 1 \end{pmatrix} &\to \begin{pmatrix} 3 & 0.707 & -0.707 \\ 0.707 & 2 & 0.707 \\ 0 & 0.707 & -1 \end{pmatrix} \to \begin{pmatrix} 3.366 & 0 & 0.325 \\ 0 & 1.634 & 0.628 \\ 0.325 & 0.628 & -1 \end{pmatrix} \\
&\to \begin{pmatrix} 3.366 & 0.072 & 0.317 \\ 0.072 & 1.776 & 0 \\ 0.317 & 0 & -1.142 \end{pmatrix} \to \begin{pmatrix} 3.388 & 0.072 & 0 \\ 0.072 & 1.776 & -0.005 \\ 0 & -0.005 & -1.164 \end{pmatrix} \to \cdots
\end{aligned} \tag{37}$$

and the process continues till the off-diagonals are small enough for our purpose.

QR algorithm : It targets to compute multiple / all eigenvalue-eigenvector pairs of the *dense* matrix $\mathbf{A}$. The QR algorithm consists of two separate steps. The first is to transform the matrix, by means of similarity transformation, in finite number of steps to Hessenberg form or to real tridiagonal form in Hermitian / symmetric case. In the next step the actual QR iterations are applied to the Hessenberg or tridiagonal matrix. Unlike the Jacobi method, the QR iteration preserves the zero entries introduced into matrix.

The QR factorization of a matrix is writing the matrix as a product of an orthogonal matrix $\mathbf{Q}$ and an upper triangular matrix $\mathbf{R}$, which in the k-th step is,

$$\mathbf{A}_k = \mathbf{Q}_k \mathbf{R}_k \quad \text{but} \quad \mathbf{A}_{k+1} = \mathbf{R}_k \mathbf{Q}_k \tag{38}$$

A quick matrix manipulation shows that the successive matrices $\mathbf{A}_k$ and $\mathbf{A}_{k+1}$ are *unitarily similar*,

$$\mathbf{A}_{k+1} = \mathbf{R}_k \mathbf{Q}_k = \mathbf{Q}_k^{-1} \mathbf{Q}_k \mathbf{R}_k \mathbf{Q}_k = \mathbf{Q}_k^{-1} \mathbf{A}_k \mathbf{Q}_k = \mathbf{Q}_k^T \mathbf{A}_k \mathbf{Q}_k \quad \text{or} \quad \mathbf{A}_k = \mathbf{Q}_k^T \mathbf{A}_{k-1} \mathbf{Q}_k \tag{39}$$

On continuing the QR iteration, we see that,

$$\mathbf{A}_k = \mathbf{Q}_k^T \mathbf{A}_{k-1} \mathbf{Q}_k = \mathbf{Q}_k^T \mathbf{Q}_{k-1}^T \mathbf{A}_{k-2} \mathbf{Q}_{k-1} \mathbf{Q}_k = \mathbf{Q}_k^T \mathbf{Q}_{k-1}^T \dots \mathbf{Q}_1^T \mathbf{A}_0 \mathbf{Q}_1 \dots \mathbf{Q}_{k-1} \mathbf{Q}_k \tag{40}$$

If we assume the eigenvalues are mutually different in magnitude, such that

$$|\lambda_1| > |\lambda_2| > \dots > |\lambda_n| \tag{41}$$

then the elements of $\mathbf{A}_k$ below the diagonal converge to 0 like

$$\left|a_{ij}^k\right| = \mathcal{O}\left(|\lambda_i/\lambda_j|^k\right) \quad i > j \tag{42}$$

Under this same assumption, $\mathbf{A}_k$ converge to an upper triangular matrix – the *Schur form* of $\mathbf{A}$. (`Schur decomposition is writing an arbitrary complex square matrix as unitarily equivalent to an upper triangular matrix whose diagonal elements are the eigenvalues of the original matrix.`) As k increases the eigenvalues of $\mathbf{A}_k$ approach the eigenvalues of $\mathbf{A}$. The product of orthogonal matrices $\mathbf{Q}_k$ converges to a matrix of corresponding eigenvectors. Finally, if $\mathbf{A}$ is symmetric, then symmetry is preserved by QR iterations and hence, $\mathbf{A}_k$ converges to a matrix that is both triangular and symmetric *i.e.* diagonal.

The main drawbacks of QR algorithm are

1. The convergence is slow. In fact it can be arbitrarily slow if eigenvalues are very close to each other despite obeying (41).

2. The algorithm requires large computational resources. In each iteration, QR factorization of a full $n \times n$ matrix has to be carried out requiring $\mathcal{O}(n^3)$ computation.

Hence there is need to improve on both issues. A matrix form that is close to an upper triangular matrix and is preserved by QR algorithm is the Hessenberg form. Let us recall that a matrix $\mathbf{H}$ is (upper) Hessenberg if its elements below the lower off-diagonal are zero,

$$h_{ij} = 0 \quad \text{for} \quad i > j + 1 \tag{43}$$

Secondly, the Hessenberg form is preserved by QR algorithm,

$$\text{if} \quad \mathbf{H} = \mathbf{Q}\mathbf{R} \quad \text{then} \quad \overline{\mathbf{H}} = \mathbf{R}\mathbf{Q} \quad \text{is also Hessenberg} \tag{44}$$

The above statement in (44) can be proved through *Given rotation*, but will not be discussed here. It turns out that the cost of simple QR factorization is $\mathcal{O}(n^3)$ then Hessenberg QR reduces it to $\mathcal{O}(n^2)$. Hence it is a good idea to perform QR algorithm with Hessenberg form rather than the full matrix. The reduction of a matrix to Hessenberg form is achieved in finite number of steps through *Householder's transformation. (In matrix inversion we used Gram-Schmidt orthogonalization method to achieve Hessenberg.)* Besides, a symmetric Hessenberg matrix is tridiagonal. A matrix of the form

$$\mathcal{P} = \mathbb{1} - 2\mathbf{u}\mathbf{u}^T, \quad \text{where} \quad ||\mathbf{u}|| = 1 \tag{45}$$

is called a *Householder matrix or reflector*. The $\mathcal{P}$ is symmetric and orthogonal (Hermitian and unitary), has eigenvalues +1 with multiplicity $n-1$ and -1 with multiplicity 1 and, therefore, its determinant is -1. The task here is to repeatedly apply Householder reflectors to transform a vector $\mathbf{x}$ (just a column of the original matrix $\mathbf{A}$) to a multiple of $\mathbf{e}_1$.

$$\mathcal{P}\mathbf{x} = \mathbf{x} - \mathbf{u}\left(2\mathbf{u}^T\mathbf{x}\right) = \alpha\,\mathbf{e}_1 \tag{46}$$

Let us illustrate this more explicitly,

$$\begin{aligned}
\mathcal{P}_1\mathbf{A} &= \mathcal{P}_1 \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} \bar{a}_{11} & \bar{a}_{12} & \bar{a}_{13} & \cdots & a_{1n} \\ 0 & \bar{a}_{22} & \bar{a}_{23} & \cdots & \bar{a}_{2n} \\ 0 & \bar{a}_{32} & \bar{a}_{33} & \cdots & \bar{a}_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \bar{a}_{n2} & \bar{a}_{n3} & \cdots & \bar{a}_{nn} \end{pmatrix} \\
\mathcal{P}_2\mathcal{P}_1\mathbf{A} &= \mathcal{P}_2 \begin{pmatrix} \bar{a}_{11} & \bar{a}_{12} & \bar{a}_{13} & \cdots & a_{1n} \\ 0 & \bar{a}_{22} & \bar{a}_{23} & \cdots & \bar{a}_{2n} \\ 0 & \bar{a}_{32} & \bar{a}_{33} & \cdots & \bar{a}_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \bar{a}_{n2} & \bar{a}_{n3} & \cdots & \bar{a}_{nn} \end{pmatrix} = \begin{pmatrix} \hat{a}_{11} & \hat{a}_{12} & \hat{a}_{13} & \cdots & a_{1n} \\ 0 & \hat{a}_{22} & \hat{a}_{23} & \cdots & \hat{a}_{2n} \\ 0 & 0 & \hat{a}_{33} & \cdots & \hat{a}_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \hat{a}_{n3} & \cdots & \hat{a}_{nn} \end{pmatrix}
\end{aligned} \tag{47}$$

Usually the operation performed is $\mathcal{P}\mathbf{A}\mathcal{P}^T$ where the multiplication of $\mathcal{P}$ from left inserts the desired zeros in appropriate column of $\mathbf{A}$ and the one from right is required to have similarity. The process continues until we have reached upper Hessenberg,

$$\mathcal{P}_{n-1}\mathcal{P}_{n-2}\cdots\mathcal{P}_2\mathcal{P}_1\mathbf{A}\mathcal{P}_1^T\mathcal{P}_2^T\cdots\mathcal{P}_{n-2}^T\mathcal{P}_{n-1}^T = \mathbf{H} \tag{48}$$

The only problem remaining is to find the $\mathcal{P}_k$ matrices *i.e.* basically the vectors $\mathbf{u}$. Depending on the initial form of matrix $\mathbf{A}$, there are various ways to do this. The mostly used one is the following (`ref: https://en.wikipedia.org/wiki/Householder_transformation`)

1. Choose the first column of $\mathbf{A}$ and normalize it : $\{a_{i1}\} \rightarrow \{\hat{a}_{i1}\}$.
2. Evaluate the following 2 constants

$$\alpha = \pm\sqrt{\sum_{j=2}^{n}\hat{a}_{j1}^2} \quad \text{and} \quad r = \sqrt{\frac{1}{2}\left(\alpha^2 - \hat{a}_{21}\alpha\right)} \tag{49}$$

   Choose minus sign if $\hat{a}_{21} \leq 0$.

3. Construct the Householder vector $\mathbf{u}^{(1)} = \{u_i\}$ as,

$$u_1 = 0, \;\; u_2 = \frac{\hat{a}_{21} - \alpha}{2r} \quad \text{and} \quad u_k = \frac{\hat{a}_{k1}}{2r} \text{ for } k = 3, 4, \ldots, n \tag{50}$$

So after the first step the Householder vector looks like

$$\mathbf{u}^{(1)} = \begin{pmatrix} 0 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{pmatrix} \tag{51}$$

4. The Householder reflector and its operation on $\mathbf{A}$ are,

$$\mathcal{P}^{(1)} = \mathbb{1} - 2\mathbf{u}^{(1)}\mathbf{u}^{(1)\,T} \quad \text{and} \quad \mathbf{A}^{(2)} = \mathcal{P}^{(1)}\mathbf{A}\mathcal{P}^{(1)} \tag{52}$$

5. Having obtained $\mathbf{A}^{(2)}$, the process is repeated for $k = 2, 3, \ldots, n-2$ as above,

$$\begin{aligned}
\alpha &= \pm\sqrt{\sum_{j=k+1}^{n}\left(\hat{a}_{jk}^{(k)}\right)^2} \quad \text{and} \quad r = \sqrt{\frac{1}{2}\left(\alpha^2 - \hat{a}_{k+1,k}^{(k)}\alpha\right)} \\
u_1^{(k)} &= u_2^{(k)} = \cdots = u_k^{(k)} = 0 \\
u_{k+1}^{(k)} &= \frac{\hat{a}_{k+1,k}^{(k)} - \alpha}{2r} \quad \text{and} \quad u_j^{(k)} = \frac{\hat{a}_{jk}^{(k)}}{2r} \quad \text{for} \quad vj = k+2, k+3, \ldots, n \\
\mathcal{P}^{(k)} &= \mathbb{1} - 2\mathbf{u}^{(k)}\mathbf{u}^{(k)\,T} \quad \text{and} \quad \mathbf{A}^{(k+1)} = \mathcal{P}^{(k)}\mathbf{A}^{(k)}\mathcal{P}^{(k)}
\end{aligned}$$

Once the Hessenberg matrix is obtained, it can be further reduced to a triangular matrix (and consequently Schur form) through *shifted QR*. No matter how complicated it sounds to reduce a general matrix to Hessenberg and then to triangular, these step economizes the arithmetic involved for eigenvalue problems. Convergence rate of QR iteration can be accelerated by incorporating *shifts*,

$$\mathbf{Q}_k\mathbf{R}_k = \mathbf{A}_{k-1} - \mu_k\mathbb{1} \quad \text{followed by} \quad \mathbf{A}_{k+1} = \mathbf{R}_k\mathbf{Q}_k + \mu_k\mathbb{1} \tag{53}$$

where μ_k is an approximation to eigenvalue. Basically this implies we shift $\mathbf{A}$ by a scalar μ and compute QR factorization of $\mathbf{A} - \mu\mathbb{1}$ instead, which is the first part of the above equation (53) and then update $\mathbf{A}$ by forming reverse product $\mathbf{RQ}$ and adding back the shift. Remembering (39) that $\mathbf{A}_{k+1} = \mathbf{Q}_k^T\mathbf{A}_k\mathbf{Q}_k$, during each iteration we have

$$\begin{aligned}
\mathbf{A}_{k+1} &= \mathbf{R}_k\mathbf{Q}_k + \mu_k\mathbb{1} \\
&= \mathbf{Q}_k^T\left(\mathbf{A}_k - \mu_k\mathbb{1}\right)\mathbf{Q}_k + \mu_k\mathbb{1} \\
&= \mathbf{Q}_k^T\mathbf{A}_k\mathbf{Q}_k - \mu_k\mathbf{Q}_k^T\mathbf{Q}_k + \mu_k\mathbb{1} \\
&= \mathbf{Q}_k^T\mathbf{A}_k\mathbf{Q}_k - \mu_k\mathbb{1} + \mu_k\mathbb{1} = \mathbf{Q}_k^T\mathbf{A}_k\mathbf{Q}_k
\end{aligned} \tag{54}$$

Krylov subspace methods : We have discussed Krylov subspace briefly in the context of matrix inversion on which the algorithm like conjugate gradient and GMRES are based. This method are among the most successful methods employed in numerical linear algebra. We start by recapitulating the basics. For the linear equation $\mathbf{Ax} = \mathbf{b}$,

$$\begin{aligned}
\text{Krylov sequence} &: \left\{\mathbf{b}, \mathbf{Ab}, \mathbf{A}^2\mathbf{b}, \cdots, \mathbf{A}^{i-1}\mathbf{b}\right\} \\
\text{Krylov subspace} &: \text{span}\left\{\mathbf{b}, \mathbf{Ab}, \mathbf{A}^2\mathbf{b}, \cdots, \mathbf{A}^{i-1}\mathbf{b}\right\} = \mathcal{K}_i \\
\text{Krylov matrix} &: \left(\mathbf{b}\middle|\mathbf{Ab}\middle|\mathbf{A}^2\mathbf{b}\middle|\cdots\middle|\mathbf{A}^{i-1}\mathbf{b}\right) = K_{n\times i}
\end{aligned} \tag{55}$$

Krylov sequence forms a basis for Krylov subspace but it is ill-conditioned. Due to multiple multiplication (power iteration), the vectors soon become almost linearly dependent and therefore some schemes for orthogonalization is necessary. Lanczos algorithm builds an orthonormal basis for Krylov subspace for Hermitian matrices and Arnoldi algorithm generalizes this to non-Hermitian.

`Please note that Lanczos and Arnoldi methods do not return eigenvalues by themselves. They just help reducing the matrix` $\mathbf{A}$ `to tridiagonal or upper Hessenberg, for which eigenvalues are far easier than the original matrix. This is typically done by Power method, Jacobi method etc.`

Let the orthonormal basis for Krylov subspace $\mathcal{K}_i$, *i.e.* the columns of matrix $\mathbf{K}_{n\times i}$ be $\mathbf{Q}_i = (\mathbf{q}_1|\mathbf{q}_2|\cdots|\mathbf{q}_i)$, then

$$\mathbf{K}_i = \mathbf{Q}_i\mathbf{R}_i \quad \text{so that} \quad \mathbf{T}_i \equiv \mathbf{Q}_i^\dagger\mathbf{A}\mathbf{Q}_i \tag{56}$$

where $\mathbf{T}$ is tridiagonal matrix for Lanczos and upper Hessenberg for Arnoldi. Let $(\lambda, \mathbf{y})$ be an eigenvalue and eigenvector of $\mathbf{T}$. The λ, is called `Ritz value` of $\mathbf{A}$ and provides an approximation for eigenvalue of $\mathbf{A}$. The $\mathbf{x} = \mathbf{Q}_i\mathbf{y}$ is called Ritz vector of $\mathbf{A}$ and provides an approximation for eigenvector of $\mathbf{A}$. Given symmetric / Hermitian matrix $\mathbf{A}$, the tridiagonal matrix $\mathbf{T}$ is also symmetric. From equation (56) we then have

$$\mathbf{AQ} = \mathbf{QT} \;\Rightarrow\; \mathbf{A}\Big(\mathbf{q}_1\Big|\mathbf{q}_2\Big|\cdots\Big|\mathbf{q}_i\Big) = \Big(\mathbf{q}_1\Big|\mathbf{q}_2\Big|\cdots\Big|\mathbf{q}_i\Big)\begin{pmatrix} \alpha_1 & \beta_2 & & & & 0 \\ \beta_2 & \alpha_2 & \beta_3 & & & 0 \\ 0 & \beta_3 & \alpha_3 & \ddots & & \\ & & \ddots & \ddots & \beta_{i-1} & \\ & & & \beta_{i-1} & \alpha_{i-1} & \beta_i \\ 0 & & & & \beta_i & \alpha_i \end{pmatrix} \tag{57}$$

Equating columns of either sides of the equation (57), we get

$$\mathbf{A}\mathbf{q}_j = \beta_{j-1}\mathbf{q}_{j-1} + \alpha_j\mathbf{q}_j + \beta_j\mathbf{q}_{j+1} \tag{58}$$

Initializing $\beta_0 = 0, \mathbf{q}_0 = \mathbf{0}$ and $\mathbf{q}_1$ to an arbitrary vector with Euclidean norm 1, the α_j, β_j and $\mathbf{q}_{j+1}$ are updated (as three term recurrence) as long as $\beta_j \neq 0$. $\beta_j = 0$ usually happens long before $j \to n$.

$$\begin{aligned} \mathbf{q}_j^T\mathbf{A}\mathbf{q}_j &= \alpha_j\,\mathbf{q}_j^T\mathbf{q}_j \\ \mathbf{r}_j \equiv \beta_j\mathbf{q}_{j+1} &= \mathbf{A}\mathbf{q}_j - \alpha_j\mathbf{q}_j - \beta_{j-1}\mathbf{q}_{j-1} \quad \Rightarrow\; \beta_j = ||\mathbf{r}_j||^2 \\ \mathbf{q}_{j+1} &= \frac{\mathbf{r}_j}{\beta_j} \end{aligned} \tag{59}$$

This is **Lanczos iteration** and works wonderfully well for sparse system and where memory is a premium. The Lanczos algorithm does not by itself give the eigenvalues, it takes additional steps to calculate them (like Power method, QR) but still it is a significant step towards calculating them.