Eigensystem

A word of wisdom:

In general the eigenvalues and eigenvectors of a matrix cannot be determined using a finite number of rational operations.

The determination of eigenvalues and eigenvectors that we are about to discuss have to be approximate.

With this disclaimer in mind, it is ironic that by importance determination of eigen values and vectors comes next to Monte Carlo methods. The standard way to express an eigenvalue problem of a $n \times n$ matrix $\mathbf{A}$ is,

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

where λ_ν are the eigenvalues and $\mathbf{x}_\nu$ the corresponding eigenvectors. For solving the eigenvalue equation, it is rewritten as,

$$\left(\mathbf{A} - \lambda_\nu \mathbb{1}\right) \mathbf{x}_\nu = 0$$

Nonzero solution $\mathbf{x}$ is only possible if the matrix $\mathbf{A} - \lambda \mathbb{1}$ is singular,

$$p_A(\lambda) = \det\left(\mathbf{A} - \lambda_\nu \mathbb{1}\right) = 0.$$

Function $p_A(\lambda)$ is called characteristic polynomial of degree n of $\mathbf{A}$.

From 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,

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

where λ_i are all distinct eigenvalues of $\mathbf{A}$ and α_i are their multiplicities.

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.

- Numerical solution of eigen system: stability, convergence, computing time
- Choice of algorithm: all or few or only e-values / e-vectors, degenerate or extremes
- Algorithm depends on $\mathbf{A}$: Hermitian, tridiagonal, Hessenberg, sparse etc.

In short, no one-size-fits-all eigen algorithm

A few asides on eigensystem :

1) Two different kind of eigenvectors – right, left

$$\mathbf{A} \cdot \mathbf{x}_R = \lambda_R \mathbf{x}_R \quad \Rightarrow \quad \det\left(\mathbf{A} - \lambda_R \mathbb{1}\right) = 0$$

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

For most of our needs $\lambda_L = \lambda_R$ and $\mathbf{x}_L = \mathbf{x}_R$. We call right eigenvalues / vectors as eigenvalues/vectors. If $\mathbf{A}$ is not specifically symmetric or Hermitian, then $\lambda_L \neq \lambda_R$

2) Generalized eigenvalue (GEVP): For two symmetric matrices $\mathbf{A}$, $\mathbf{B}$,

$$\mathbf{A} \cdot \phi_\nu = \lambda_\nu \mathbf{B} \phi_\nu \quad \Rightarrow \quad \mathbf{A} \cdot \boldsymbol{\phi} = \boldsymbol{\Lambda}\, \mathbf{B} \boldsymbol{\phi}$$

Assuming $\mathbf{B}$ to be positive definite (has all positive e-values), the e-vectors ϕ can be made to satisfy

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

Cholesky factorization $\mathbf{B} = \mathbf{L}\mathbf{L}^T$ leads to

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

From the definition of GEVP it follows that

$$\begin{aligned} \mathbf{A} \cdot \phi &= \mathbf{B}\phi\mathbf{\Lambda} \\ \mathbf{A}\mathbf{L}^{-T}\mathbf{\Psi} &= \mathbf{L}\mathbf{L}^{T}\mathbf{L}^{-T}\mathbf{\Psi}\mathbf{\Lambda} \\ \mathbf{L}^{-1}\mathbf{A}\mathbf{L}^{-T}\mathbf{\Psi} &= \mathbf{\Psi}\mathbf{\Lambda} \\ \Rightarrow \quad \tilde{\mathbf{A}}\mathbf{\Psi} &= \mathbf{\Psi}\mathbf{\Lambda} \end{aligned}$$

implying GEVP has been transformed to a regular eigenvalue problem through Cholesky decomposition of **B**.

3) Stability can be a problem occasionally – small changes in **A** do not necessarily lead to small changes in the eigenvalues.

$$\mathbf{A} = \begin{pmatrix} 1 & 1000 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad \lambda_A = 1,\ 1$$

$$\mathbf{B} = \begin{pmatrix} 1 & 1000 \\ 0.001 & 1 \end{pmatrix} \quad \text{and} \quad \lambda_B = 0,\ 2$$

This tells when (rounding) errors get into any of the entries of **A**, then exercise extra caution while using their eigenvalues.

For **A** symmetric, small changes do not lead to large changes in evalues. Hence, algorithms for symmetric matrices are generally successful.

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

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

where $\mathbf{A}$ is $n \times n$ real, symmetric matrix. Let $\mathbf{E}$ be real, symmetric $n \times n$ error matrix.

Suppose $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{E}$. If λ_ν and $\hat{\lambda}_\nu$ are evalues of $\mathbf{A}$ and its error version $\hat{\mathbf{A}}$ respectively, then

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

This is stability theorem – λ_ν is stable *i.e.* small errors in $\mathbf{A}$ result in small errors in λ_ν. Consider for example

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

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

$$|\lambda_\nu - \hat{\lambda}_\nu| \le ||\mathbf{E}||_F = \sqrt{(0.01)^2 + 2(-0.05)^2 + 4(0.1)^2 + (0.03)^2 + (-0.1)^2} = 0.2366\ldots$$

Disturbingly large error bound even for a symmetric matrix!

Power method

Power method or von Mises iteration returns the largest $|\lambda|$ and corresponding evector $\mathbf{v}$. It is a simple algorithm and may converge slowly. Despite being simple, this algorithm was part of the Google's first version of PageRank algorithm. Power method is used when

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

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

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

where $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ is a set of linearly independent evectors. By repeated multiplication by $\mathbf{A}$ the following set of equations are formed,

$$\mathbf{x}_1 = \mathbf{A}\mathbf{x}_0 = c_1\lambda_1\mathbf{v}_1 + c_2\lambda_2\mathbf{v}_2 + \cdots + c_n\lambda_n\mathbf{v}_n$$
$$\mathbf{x}_2 = \mathbf{A}^2\mathbf{x}_0 = c_1\lambda_1^2\mathbf{v}_1 + c_2\lambda_2^2\mathbf{v}_2 + \cdots + c_n\lambda_n^2\mathbf{v}_n$$
$$\cdots$$
$$\mathbf{x}_k = \mathbf{A}^k\mathbf{x}_0 = c_1\lambda_1^k\mathbf{v}_1 + c_2\lambda_2^k\mathbf{v}_2 + \cdots + c_n\lambda_n^k\mathbf{v}_n$$

Dividing the last equation by λ_1^k

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

$\mathbf{x}_0$ 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 gives

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

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

$$\frac{\mathbf{x}_k \cdot \mathbf{y}}{\lambda_1^k} = \frac{1}{\lambda_1^k}\mathbf{A}^k\mathbf{x}_0 \cdot \mathbf{y} = c_1\mathbf{v}_1 \cdot \mathbf{y}$$

$$\frac{\mathbf{x}_{k+1} \cdot \mathbf{y}}{\lambda_1^{k+1}} = \frac{1}{\lambda_1^{k+1}}\mathbf{A}^{k+1}\mathbf{x}_0 \cdot \mathbf{y} = c_1\mathbf{v}_1 \cdot \mathbf{y}$$

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

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

Convergence to true λ_1 depends on how $(\lambda_2/\lambda_1)^k$ is decreasing with k.

Eigenvector $\mathbf{v}_1$ calculation: generally sequence of vectors given by

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

approaches a multiple of dominant evector of $\mathbf{A}$.

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

Choosing $\mathbf{y} = \mathbf{A}^k\mathbf{x}_0$ then from Rayleigh quotient formula,

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

Hence, $\mathbf{A}^k\mathbf{x}_0$ is approximately $\mathbf{v}_1$ corresponding to λ_1.

Although not recommended, but one can attempt to find non-dominant evalues by method of deflation,

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

Evalues of $\mathcal{A}$ are $0, \lambda_2, \lambda_3, \ldots$ and evectors of $\mathcal{A}$ are evectors of $\mathbf{A}$.

Depending on accuracy achieved for λ_1, large errors can creep into λ_2.

Jacobi method / Givens rotation

When all or almost all evalues but no evectors are required, then the approach is repeated similarity transformations on $\mathbf{A}$ to render it diagonal or tridiagonal. Assuming $\mathbf{A}$ real and symmetric

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

Similarity transformation does not change evalues of a matrix but does change its evectors to $\boldsymbol{\mathcal{S}}^T \mathbf{x}$. Idea is to apply series of $\boldsymbol{\mathcal{S}}$ to render $\mathbf{A}$ diagonal

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

This is achieved by Givens rotation. Consider an $n \times n$ orthogonal transformation matrix

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

$\boldsymbol{\mathcal{S}}$ is a rotation matrix in n-dimensional plane. The $\sin\theta$ and $\cos\theta$ appear at the intersections of k-th and l-th row and column, where $k > l$.

Non-zero elements of Givens matrix are

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

Similarity transformation with $\boldsymbol{\mathcal{S}}$ resulting in $\mathbf{B} = \boldsymbol{\mathcal{S}}^T \mathbf{A} \boldsymbol{\mathcal{S}}$,

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

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

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

Trivial demonstration of Givens rotation with a 2×2 matrix,

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

where $c = \cos\theta$, $s = \sin\theta$, $t = \tan\theta$ and $a_{12} = a_{21}$. To render $\mathbf{A}$ diagonal, choose θ accordingly,

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

For example,

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

Givens rotation :

- reliable, simple to code, relatively accurate
- for sparse matrices low arithmetics operations
- converges very slowly, difficult to extend beyond symmetric matrices

One source of inefficiency is that previously annihilated entries can subsequently become nonzero again because of rotation and hence require repeated annihilation. For example,

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

Repeated application of $\boldsymbol{\mathcal{S}}$, using the θ's above, yields

$$\begin{pmatrix} 1 & 0 & 2 \\ 0 & 2 & 1 \\ 2 & 1 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 3 & 0.707 & 0 \\ 0.707 & 2 & 0.707 \\ 0 & 0.707 & -1 \end{pmatrix} \rightarrow \begin{pmatrix} 3.366 & 0 & 0.325 \\ 0 & 1.634 & 0.628 \\ 0.325 & 0.628 & -1 \end{pmatrix}$$

$$\rightarrow \begin{pmatrix} 3.366 & 0.072 & 0.317 \\ 0.072 & 1.776 & 0 \\ 0.317 & 0 & -1.142 \end{pmatrix} \rightarrow \begin{pmatrix} 3.388 & 0.072 & 0 \\ 0.072 & 1.776 & -0.005 \\ 0 & -0.005 & -1.164 \end{pmatrix}$$

$$\rightarrow \quad \cdots$$

the process continues till the off-diagonals are small enough. To preserve the zero entries introduced into matrix, use QR algorithm.

QR algorithm

Recall : QR factorization of a matrix is a product of orthonormal $\mathbf{Q}$ and upper triangular $\mathbf{R}$, which in k-th step

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

Successive matrices $\mathbf{A}_k$ and $\mathbf{A}_{k+1}$ are unitarily similar,

$$\mathbf{A}_{k+1} = \mathbf{R}_k\mathbf{Q}_k = \mathbf{Q}_k^{-1}\mathbf{Q}_k\mathbf{R}_k\mathbf{Q}_k = \mathbf{Q}_k^{-1}\mathbf{A}_k\mathbf{Q}_k = \mathbf{Q}_k^T\mathbf{A}_k\mathbf{Q}_k$$
$$\Rightarrow \quad \mathbf{A}_k = \mathbf{Q}_k^T\mathbf{A}_{k-1}\mathbf{Q}_k$$

and similar matrices have same evalues. On continuing QR iteration,

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

Assuming eigenvalues are mutually different in magnitude such that,

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$$

$\Rightarrow$ $\mathbf{A}_k$ below diagonal converge to 0 and $\mathbf{A}$ converges to upper triangular matrix having evalues on the diagonal – Schur form.

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, eigenvalues of $\mathbf{A}_k$ approach eigenvalues of $\mathbf{A}$.

If $\mathbf{A}$ is symmetric, then symmetry is preserved by QR iterations. Hence, $\mathbf{A}_k$ converges to both triangular and symmetric *i.e.* diagonal matrix.

Product of orthogonal matrices $\mathbf{Q}_k$ converges to matrix of corresponding eigenvectors $\Rightarrow$ eigenvectors of $\mathbf{A}$ are the columns of Q. Writing $\mathcal{Q} = \mathbf{Q}_1 \dots \mathbf{Q}_{k-1}\mathbf{Q}_k$, then

$$\mathbf{A}_k = \mathcal{Q}^T \mathbf{A}_0 \mathcal{Q}$$

If $\mathbf{A}_k$ becomes diagonal, supposing $\mathbf{A}$ is symmetric, then eigenvectors are $\{\hat{e}_1, \hat{e}_2, \dots, \hat{e}_n\}$. Since $\mathbf{A}_0$ and $\mathbf{A}$ are similar, the eigenvector od $\mathbf{A}_0$ are $\mathcal{Q}\hat{e}$ *i.e.* columns of $\mathcal{Q}$.

Matrix close to an upper triangular matrix and is preserved by QR algorithm is the Hessenberg form.

Matrix $\mathbf{H}$ is (upper) Hessenberg if its elements below the lower sub-diagonal are zero.

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

Hessenberg form is preserved by QR

$$\text{If } \mathbf{H} = \mathbf{QR} \quad \text{then} \quad \overline{\mathbf{H}} = \mathbf{RQ} \quad \text{is also Hessenberg}$$

Householder transformation generates $\mathbf{H}$, which is further reduced to Schur form through QR.

Sounds complicated but reducing $\mathbf{A}$ to $\mathbf{H}$ and then to Schur economizes arithmetic involved in determining evalues.

Convergence rate of QR iteration is accelerated by incorporating shifts,

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

μ_k is approximation to eigenvalues.

Remembering $\mathbf{A}_{k+1} = \mathbf{Q}_k^T\mathbf{A}_k\mathbf{Q}_k$, during each iteration we have

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

Krylov subspace methods

Krylov subspace methods are among the most successful methods employed in numerical linear algebra. Matrix inverters like Conjugate Gradient and GMR are based on it. For linear equation $\mathbf{Ax} = \mathbf{b}$,

$$\text{Krylov sequence} : \left\{\mathbf{b}, \mathbf{Ab}, \mathbf{A}^2\mathbf{b}, \cdots, \mathbf{A}^{i-1}\mathbf{b}\right\}$$

$$\text{Krylov subspace} : \text{span}\left\{\mathbf{b}, \mathbf{Ab}, \mathbf{A}^2\mathbf{b}, \cdots, \mathbf{A}^{i-1}\mathbf{b}\right\} = \mathcal{K}_i$$

$$\text{Krylov matrix} : \left(\mathbf{b}\middle|\mathbf{Ab}\middle|\mathbf{A}^2\mathbf{b}\middle|\cdots\middle|\mathbf{A}^{i-1}\mathbf{b}\right) = K_{n\times i}$$

Lanczos algorithm builds an orthonormal basis for Krylov subspace for Hermitian matrices and Arnoldi algorithm generalizes this to non-Hermitian.

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

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

$\mathbf{T}$ is tridiagonal matrix for Lanczos and upper Hessenberg for Arnoldi.

Let $(\lambda, \mathbf{y})$ be evalue and evector of $\mathbf{T}$. The λ is called Ritz value of $\mathbf{A}$ that provides an approximation for evalues of $\mathbf{A}$. The $\mathbf{x} = \mathbf{Q}\mathbf{y}$ is called Ritz vector that provides a approximation for evectors.

If $\mathbf{A}$ is symmetric / Hermitian, the tridiagonal matrix $\mathbf{T}$ is also symmetric.

$$\mathbf{AQ} = \mathbf{QT} \quad \Rightarrow$$

$$\mathbf{A}\left(\mathbf{q}_1 \middle| \mathbf{q}_2 \middle| \cdots \middle| \mathbf{q}_i\right) = \left(\mathbf{q}_1 \middle| \mathbf{q}_2 \middle| \cdots \middle| \mathbf{q}_i\right) \begin{pmatrix} \alpha_1 & \beta_2 & & & & 0 \\ \beta_2 & \alpha_2 & \beta_3 & & & 0 \\ 0 & \beta_3 & \alpha_3 & \ddots & & \\ & & \ddots & \ddots & \beta_{i-1} & \\ & & & \beta_{i-1} & \alpha_{i-1} & \beta_i \\ 0 & & & & \beta_i & \alpha_i \end{pmatrix}$$

Equating columns of either sides,

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

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, $\mathbf{q}_{j+1}$ are updated as three term recurrence, as long as $\beta_j \neq 0$. Usually $\beta_j = 0$ happens long before $j \to n$.

$$\mathbf{q}_j^T \mathbf{A}\mathbf{q}_j = \alpha_j \mathbf{q}_j^T \mathbf{q}_j$$

$$\mathbf{r}_j \equiv \beta_j \mathbf{q}_{j+1} = \mathbf{A}\mathbf{q}_j - \alpha_j \mathbf{q}_j - \beta_{j-1}\mathbf{q}_{j-1} \quad \Rightarrow \quad \beta_j = ||\mathbf{r}_j||^2$$

$$\mathbf{q}_{j+1} = \frac{\mathbf{r}_j}{\beta_j}$$

This is Lanczos iteration and works wonderfully well for sparse system and where memory is a premium.

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.