Eigenvalues and eigenvectors

- especially in quantum mechanics, we often need to find eigenvalues and eigenstates of a system, by discretization or expansion to a basis, we obtain from Schrödinger equation $H\psi = E\psi$ a system of linear equations of the type $Ax = \lambda x$ which we usually solve numerically and necessarily iteratively
  (for $n > 4$, there are no closed-form solutions for finding roots of polynomial of order n)

- summary from algebra:
  - in $Ax = \lambda x$ we call λ eigenvalue and x (right) eigenvector if $x \neq 0$; λ i degenerated if there is more than one corresponding eigenvector (linearly independent) ← these form eigenspace for λ
  - all $\{\lambda_i\}_{i=1}^{n_\lambda}$ form the spectrum of A
  - λ's satisfies $\det(\lambda I - A) = P_A(\lambda) = 0$ where P_A is the characteristic polynomial
    - even for real matrices, λ's can be complex (and in general are)
  - defective matrices:
    algebraic multiplicity > geometric multiplicity
    (multiplicity of a root of $P_A(\lambda)$) (number of linearly independent eigenvectors for λ)

    example $A = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}$ 3 eigenvectors

    $B = \begin{pmatrix} 2 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{pmatrix}$ ← here only one non-zero eigenvector $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$
  - matrix is diagonalizable iff it is not defective
  - similarity transformations $B = X^{-1}AX$ do not change eigenvalues
    ($\det(B - \lambda I) = \det X^{-1}(A - \lambda I)X = \det(A - \lambda I)$)

    ⇒ we can transform A to a diagonal matrix Λ (with λ_i) using similarity transf. if A is not defective
    if $A = X\Lambda X^{-1}$ then X contains eigenvectors
  - moreover, shift by a constant (of all eigenvalues) does not change eigenvectors
    i.e. A and $A + \mu I$ have the same eigenvectors

- real symmetric ($A=A^T$) and hermitian ($A^\dagger=A$) matrices have real eigenvectors

- <u>normal matrices</u> ($[A,A^\dagger]=0$), including hermitian and unitary matrices, are diagonalizable by unitary transformations i.e. $A=Q\Lambda Q^\dagger$, where $Q^\dagger=Q^{-1}$, thus eigenvectors can be chosen to be all orthogonal

- determinant and trace of a matrix are invariants under similarity transformations $\Rightarrow \det A=\prod_{j=1}^{n}\lambda_j$, $\mathrm{Tr}\,A=\sum_{j=1}^{n}\lambda_j$

- <u>unitary triangularization</u> (called <u>Schur factorization</u>) $A=QTQ^\dagger$ exists always but T is upper triangular with λ's on the diagonal

- <u>numerical solution of the eigenvalue problem</u>

  - as for $Ax=b$, also here exist quite efficient algorithms implemented in libraries such as LAPACK etc.
  - how the problem is solved depends on the type of a matrix
  - methods for symmetric (hermitian) matrices are much more efficient than for general matrices
  - different methods were devised if we need only a few (usually smallest) eigenvalues and corresponding eigenvectors
  - often, also generalized eigenvalue problem $Ax=\lambda Bx$ is solved in the libraries, which is important for example in quantum chemistry (B is an overlap matrix if we have a non-orthogonal basis)

- the most efficient methods to find all eigenvalues and eigenvectors are based on two steps:

  1) transforming A with similarity transformations to a special form by a direct method with $\mathcal{O}(n^3)$ operations
     symmetric (hermitian) A → tridiagonal
     general A → Hessenberg matrix → $\begin{pmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{pmatrix}$
  2) application of very efficient algorithms for these special types of matrices

Jacobi method for eigenvalue problem

- simple method for real symmetric matrices
- basic idea: we zero off-diagonal elements using rotations, i.e. using orthogonal transformations

Example: a matrix 2×2 can be diagonalized exactly:

having $A = \begin{pmatrix} a & d \\ d & b \end{pmatrix}$, we search for $J = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$ with $c = \cos\varphi$, $s = \sin\varphi$

such that $J^T A J = \begin{pmatrix} x & 0 \\ 0 & y \end{pmatrix}$

we set the condition $d(c^2 - s^2) - cs(b-a) = 0$

thus $\frac{2d}{b-a} = \frac{2cs}{c^2-s^2} = \frac{\sin 2\varphi}{\cos 2\varphi} = \operatorname{tg} 2\varphi$

- in larger matrices, we zero elements below and above the diagonal using matrices of the form

$$P = \begin{pmatrix} 1 & & & & & 0 \\ & \ddots & & & & \\ & & c & & s & \\ & & & \ddots & & \\ & & -s & & c & \\ & & & & & \ddots \\ 0 & & & & & 1 \end{pmatrix} \begin{matrix} \\ \\ i \\ \\ j \\ \\ \end{matrix}$$

to eliminate elements a_{ij} and a_{ji} but we can destroy other zeroes on i-th and j-th rows and columns, but it works anyway if we repeat this zeroing systematicly for all off-diagonal elements

- to zero elements a_{pq} and a_{qp}, we change the following elements (rows and colouns p, q):

$$\left.\begin{aligned} a'_{rp} &= a'_{pr} = c\,a_{rp} - s\,a_{rq} \\ a'_{rq} &= a'_{qr} = c\,a_{rq} + s\,a_{rp} \end{aligned}\right\} \text{ for } r \neq p \text{ and } r \neq q$$

$$a'_{pp} = c^2 a_{pp} + s^2 a_{qq} - 2cs\,a_{pq}$$

$$a'_{qq} = s^2 a_{pp} + c^2 a_{qq} + 2cs\,a_{pq}$$

$$a'_{pq} = a'_{qp} = 0 = (c^2 - s^2)\,a_{pq} + cs(a_{pp} - a_{qq})$$

with $$\operatorname{cotg}(2\varphi) = \frac{c^2 - s^2}{2cs} = \frac{a_{qq} - a_{pp}}{2a_{pq}}$$

but this basic version is not used because of instabilities due to round-off errors

- we can rewrite the method using

$$tg\varphi = t = \frac{\sin\varphi}{\cos\varphi} = \frac{s}{c} \quad \text{and} \quad \theta = \operatorname{cotg}(2\varphi) = \frac{c^2-s^2}{2cs} = \frac{1-t^2}{2t} = \frac{a_{qq}-a_{pp}}{2a_{pq}}$$

we can express t from θ:

$$t^2 + 2t\theta - 1 = 0 \quad \Rightarrow \quad t = -\theta \pm \sqrt{\theta^2+1}$$

and more stable choice is a smaller root

$$t = \begin{cases} -\theta + \sqrt{\theta^2+1} = \dfrac{1}{\theta+\sqrt{\theta^2+1}} & \text{for } \theta > 0 \\ -\theta - \sqrt{\theta^2+1} = \dfrac{-1}{-\theta+\sqrt{\theta^2+1}} & \text{for } \theta < 0 \end{cases} \qquad \frac{sign(\theta)}{|\theta| + \sqrt{\theta^2+1}}$$

↑ to avoid subtracting two close numbers

if θ is too large, we can avoid evaluating θ^2 simply by taking $t = \frac{1}{2\theta}$

s and c can be now expressed as

$$t^2 = \frac{1-c^2}{c^2} \Rightarrow c = \frac{1}{\sqrt{t^2+1}}, \quad s = tc$$

and we finally set

$$a'_{pq} = a'_{qp} = 0 \qquad a'_{rp} = a'_{pr} = a_{rp} - s(a_{rq} + \tau a_{rp})$$

$$a'_{pp} = a_{pp} - t a_{pq} \qquad a'_{rq} = a'_{qr} = a_{rq} + s(a_{rp} - \tau a_{rq})$$

$$a'_{qq} = a_{qq} + t a_{pq} \qquad \text{with} \quad \tau = \frac{s}{1+c}$$

where all elements are rewritten as modification of the previous values

- one can show that the sum of all off-diagonal elements

$$S = \sum_{r \neq s} |a_{rs}|^2 \text{ is changed to } S' = S - 2|a_{pq}|^2 \quad (\text{it will be smaller!})$$

when a_{pq} is eliminated

thus S is decreasing and the method will be more efficient when „small" a_{pq} are skipped

- if we would choose the largest $a_{pq} > \frac{S}{n^2}$ then

$$S' < \left(1 - \frac{1}{n^2}\right) S \Rightarrow S^{(k)} < \left(1 - \frac{1}{n^2}\right)^k S_0$$

and we would need $O(n^2)$ iterations each with $O(n)$ operations

$\Rightarrow$ estimate $O(n^3)$ operations

- eigenvectors: $V = \mathbb{1} \cdot \underbrace{P_1 \cdot P_2 \cdots P_k}_{\text{subsequent rotations}} \Rightarrow$ we modify $\mathbb{1}_{n\times n} = V$ using

$$v'_{rs} = v_{rs}, \quad s \neq p, \ s \neq q$$

$$v'_{rp} = c v_{rp} - s v_{rq} = v_{rp} - s(v_{rq} + \tau v_{rp})$$

$$v'_{rq} = s v_{rp} + c v_{rq} = v_{rq} + s(v_{rp} - \tau v_{rq})$$

Methods for one or a few specific eigenvalues

- **power iteration** - converges to the largest eigenvalue λ_1 and the corresponding eigenvector q_1 if the initial guess is not perpendicular to the eigenvector q_1

algorithm:

pick $v^{(0)}$ ← some normalized initial guess

for $k = 1, 2, \ldots$

$w = A v^{(k-1)}$

$v^{(k)} = w / \|w\|$

$\lambda^{(k)} = (v^{(k)})^T A v^{(k)}$

- for $q_1^T v^{(0)} \neq 0$, convergence can be shown

by expanding $v^{(0)} = a_1 q_1 + \ldots + a_n q_n$, q_i being eigenvectors of A

thus $v^{(k)} = \underset{\text{normalization}}{C_k} A^k v^{(0)} =$

$= C_k (a_1 \lambda_1^k q_1 + \ldots + a_n \lambda_n^k q_n) =$

$= C_k \lambda_1^k \left(a_1 q_1 + a_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k q_2 + \ldots + a_n \left(\frac{\lambda_n}{\lambda_1}\right)^k q_n \right)$

and assuming $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \ldots \geq |\lambda_n| \geq 0$

the rate of convergence is

$$\| v^{(k)} - (\pm q_1) \| = \sigma\left(\left| \frac{\lambda_2}{\lambda_1} \right|^k \right)$$

$$|\lambda^{(k)} - \lambda_1| = \sigma\left(\left| \frac{\lambda_2}{\lambda_1} \right|^{2k} \right)$$

- **inverse iteration** - it used to find eigenvectors if we know eigenvalues

- to find other eigenvalues and eigenvectors for which we have a good estimate we can use the matrix $(A - \mu I)^{-1}$ instead of A
- this matrix has the same eigenvectors and the eigenvalues are $(\lambda_j - \mu)^{-1}$

- algorithm of inverse iteration

pick $v^{(0)}$ and μ ← μ should be close to requiered eigenvalue

for $k = 1, 2, \ldots$

solve $(A - \mu I)\, w = v^{(k-1)}$

$v^{(k)} = w / \|w\|_2$

$\lambda^{(k)} = (v^{(k)})^T A v^{(k)}$

it converges to λ_j and corresponding q_j to which μ is closest if $(v^{(0)})^T q_j \neq 0$

- why not to replace μ with $\lambda^{(k)}$ in each iteration?

this idea leads to <u>Rayleigh quotient iteration</u>

algorithm:

pick $v^{(0)}$ and $\lambda^{(0)}$

for $k = 1, 2, \ldots$

solve $(A - \lambda^{(k-1)} I)\, w = v^{(k-1)}$

$v^{(k)} = w / \|w\|_2$

$\lambda^{(k)} = (v^{(k)})^T A v^{(k)}$

- this algorithm converges very fast (cubically) to λ_j and q_j if close enough

---

- there are specialized algorithms to find several lowest (or highest) eigenvalues and eigenvectors

- an example is <u>Davidson</u> method inspired by large eigenvalue problems in quantum chemistry where typically only a few lowest states are required

- it iteratively projects the matrix on suitable subspaces and diagonalizes it there

- details can be found elsewhere

When all eigenvalues (and eigenvectors) are needed

1) Reduction of the matrix A to

a) Hessenberg form for a general A

b) or tridiagonal form for a symmetric (real) or Hermitian (complex) matrix A

a) we use Householder reflections or Givens rotations (for banded matrices) to get Hessenberg form (upper triangular with one non-zero subdiagonal)

$$A \xrightarrow{\begin{pmatrix}1&0\\0&F\end{pmatrix}} Q_1^\dagger A \qquad Q_1^\dagger A Q_1$$

$$\begin{pmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \end{pmatrix} \xrightarrow{Q_1^\dagger} \begin{pmatrix} x & x & x & x \\ y & y & y & y \\ 0 & y & y & y \\ 0 & y & y & y \end{pmatrix} \xrightarrow{Q_1} \begin{pmatrix} x & z & z & z \\ y & z & z & z \\ 0 & z & z & z \\ 0 & z & z & z \end{pmatrix}$$

algorithm for $k = 1, \dots, n-2$

$$x = A_{k+1:n,\,k}$$

$$v_k = \operatorname{sign}(x_1)\,\|x\|_2 e_1 + x$$

as in QR factorization $\quad v_k = v_k / \|v_k\|_2$

volume ⬡ $\Rightarrow \sim 4 \cdot \frac{1}{3} n^3$ operations $\longrightarrow$ $A_{k+1:n,\,k:n} = A_{k+1:n,\,k:n} - 2 v_k \left(v_k^\dagger A_{k+1:n,\,k:n}\right)$

volume ⬢ $\Rightarrow \sim 4 \cdot \frac{1}{2} n^3$ operations $\longrightarrow$ $A_{1:n,\,k+1:n} = A_{1:n,\,k+1:n} - 2 \left(A_{1:n,\,k+1:n} v_k\right) v_k^\dagger$

together $\sim \frac{10}{3} n^3$ operations

- as in QR factorization, here it is also a backward stable algorithm

i.e. $\tilde{Q} \tilde{H} \tilde{Q}^\dagger = A + \delta A$ where $\frac{\|\delta A\|}{\|A\|} = \mathcal{O}(\varepsilon_m)$

for some $\delta A \in \mathbb{C}^{n \times n}$

b) for Hermitian matrices this algorithm leads to tridiagonal matrices because if A is Hermitian, then $Q^\dagger A Q$ is also Hermitian and any Hermitian Hessenberg matrix is tridiagonal

but we can of course skip computation of zero elements

$\Rightarrow$ applying $Q_k^\dagger$ needs the same number of operations as Q_k

and thanks to symmetry, we finally get $\sim \frac{4}{3} n^3$ operations

2) calculation of Schur factorization in a general case

or diagonalization in a hermitian (real sym.) case

a) in a general case, the standard method to get Schur factorization is QR algorithm with shifts

basic idea without shifts:

$A^{(0)} = A$

for $k = 1, 2, \dots$

find QR factor. of $A^{(k-1)}$ → $Q^{(k)} R^{(k)} = A^{(k-1)}$

this gives → $A^{(k)} = R^{(k)} Q^{(k)}$

$A^{(k)} = Q^{(k)\dagger} A^{(k-1)} Q^{(k)}$

similarity transformation of $A^{(k-1)}$

algorithm with shifts in practice

$Q^{(0)\dagger} A^{(0)} Q^{(0)} = A$ ← Hessenberg or tridiagonal result in $A^{(0)}$

for $k = 1, 2, \dots$

pick a shift $\mu^{(k)}$

$Q^{(k)} R^{(k)} = A^{(k-1)} - \mu^{(k)} I$

$A^{(k)} = R^{(k)} Q^{(k)} + \mu^{(k)} I$

if off-diagonal $A^{(k)}_{j+1,j}$ is close to zero, set it to 0 and continue separately with A_1 and A_2 in

$$\begin{matrix} & & j \\ & & \downarrow \\ j+1 \rightarrow & \left(\begin{array}{c|c} A_1 & X \\ \hline 0 & A_2 \end{array} \right) & = A^{(k)} \end{matrix}$$

- in the algorithm with shifts we also get similarity transformation:

$$A^{(k)} = R^{(k)} Q^{(k)} + \mu^{(k)} I = Q^{(k)\dagger} Q^{(k)} R^{(k)} Q^{(k)} + \mu^{(k)} Q^{(k)\dagger} Q^{(k)} =$$

$$= Q^{(k)\dagger} \left(Q^{(k)} R^{(k)} + \mu^{(k)} I \right) Q^{(k)} = Q^{(k)\dagger} A^{(k-1)} Q^{(k)}$$

for an arbitrary shift

- standard choices of shifts are

Rayleigh quotient shift

$$\mu^{(k)} = A^{(k)}_{n,n}$$

Wilkinson shift

$$\mu^{(k)} = A^{(k)}_{n,n} - \mathrm{sign}(\delta) \, A^{(k)2}_{n,n-1} \Big/ \sqrt{|\delta| + \sqrt{\delta^2 + A^{(k)2}_{n,n-1}}}$$

where $\delta = (A^{(k)}_{n-1,n-1} - A^{(k)}_{n,n})/2$ (if $\delta = 0$, sign = 1)

this method is always convergent and avoids situations like in $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ when $Q^{(1)} R^{(1)} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

and thus $A^{(1)} = R^{(1)} Q^{(1)} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = A$ and shift by $A_{2,2} = 0$ does not help!

b) in a hermitian case, we can use either QR alg.

or there are more efficient algorithms as Divede-and-Conquer

(see Trefethen or Demmel for details)