Auxiliary notes on Computational Linear Algebra

Daniel Lin; Professor: Colin Cotter

January 18, 2023

---

1 Outer product

Proposition 1.1. *The product of an outer product to itself is simply a matrix scaling. (And the scale is the inner product)*

$$(uv^*)(uv^*) = v^*u(uv^*)$$

note $v^*u \in \mathbb{C}$

Proof. Perform matrix multiplication element-wise, first note $[\boldsymbol{uv}^*]_{i,j} = u_i\overline{v_j}$.

$$[\boldsymbol{uv}^*\boldsymbol{uv}^*]_{i,j} = \sum_{k=1}^{m} u_i\overline{v_k}u_k\overline{v_j} = u_i\overline{v_j}\sum_{k=1}^{m}\overline{v_k}u_k = \boldsymbol{v}^*\boldsymbol{u}(u_i\overline{v_j}) = \boldsymbol{v}^*\boldsymbol{u}[\boldsymbol{uv}^*]_{i,j}$$

□

Solutions to Exercise 1.13
According to Proposition 1.1, and the definition that $A^{-1}A = AA^{-1} = I$.

$$(I + uv^*)(I + \alpha uv^*) = I + (1+\alpha)uv^* + \alpha(uv^*)(uv^*) = I \Rightarrow (1 + \alpha + \alpha v^*u)uv^* = 0$$

If $uv^* = 0$, $A = I$. Trivial case.
Otherwise, $1 + \alpha + \alpha v^*u = 0$, so $\alpha = -\frac{1}{1+v^*u}$.

Proposition 1.2 (Useful identities for outer product)**.** .

- *For any* $x \in \mathbb{C}^n$, $uv^*x = (v^*x)u$
- $u_1v_1^*u_2v_2^* = v_1^*u_2(u_1v_2^*)$. *Proposition 1.1 is a corollary of this.*

Proposition 1.3 (Sum of outer products)**.** *Given* k *linearly independent vectors* $(u_i)_{i=1,2,\dots,k} \subset \mathbb{C}^n$ *(*$k \leq n$*), and other vectors* $(v_i)_{i=1,2,\dots,k} \subset \mathbb{C}^n$ *the sum of outer product*

$$\sum_{i=1}^{k} u_iv_i^*$$

has rank k.
Further, $\sum_{i=1}^{k} u_iv_i^* = UV^*$ *where*

$$U = \begin{pmatrix} | & & | \\ u_1 & \cdots & u_k \\ | & & | \end{pmatrix} \qquad V = \begin{pmatrix} | & & | \\ v_1 & \cdots & v_k \\ | & & | \end{pmatrix}$$

Proof. Use proposition 1.2. □

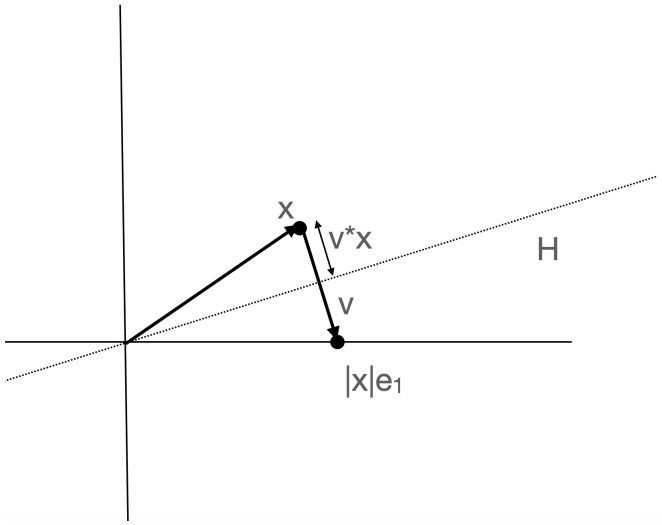

Figure 1: Householder Reflection

2 Householder reflection

This is a quick explanation of how the transformation F taking x to $\|x\|e_1$ is constructed. See figure 1

Starting from x, we first want to reach the hyperplane H orthogonal to $v = |x|e_1 - x$. Since H crosses the origin, the length along v that we should move is

$$\frac{v^*x}{v^*v} = \frac{\langle v, x\rangle}{\|v\|^2}$$

as the inner product is essentially calculating a scaled projection length. So

$$x - \frac{v^*x}{v^*v}v = x - \frac{vv^*}{v^*v}x$$

takes us to the hyperplane (equality due to the first identity in proposition 1.2) By symmetry, we should move further along v by the same distance to reach the axis, so

$$|x|e_1 = x - 2\frac{vv^*}{v^*v}x = (I - 2\frac{vv^*}{v^*v})x$$

.

3 Faulhaber's Formulae and operation count

When counting operations$(+, -, \times, \div)$ you may often encounter sums like $\sum_{k=1}^{n}(n-k)^2$ and inevitably you have to find $\sum_{k=1}^{n} k^2$ or even higher powers. The precise formulae, due to Faulhaber, is rather complicated.

$$\sum_{k=1}^{n} k^a = \frac{1}{a+1}\sum_{j=0}^{a}(-1)^j \binom{a+1}{j} B_j n^{a+1-j}$$

Where B_j are Bernoulli numbers. The leading term n^{a+1}, corresponds to $j = 0$ in the summation. Since operation counts only care about dominating terms, finding the coefficient for n^{a+1} is enough. $B_0 = 1$ so the coefficient is simply $1/(a+1)$. Conclusion:

$$\sum_{k=1}^{n} k^a \sim \frac{n^{a+1}}{a+1}$$

Sometimes, you may have to sum a "triangle". i.e. if outer loop i is from 0 to n, but inner loop is $0 : i$ or $i : n$. Instead of calculating this annoying sum, you can approximate it by integrals:

$$\int_0^n \int_0^x 1 \, \mathrm{dx}' \, \mathrm{dx}, \quad \int_0^n \int_x^n 1 \, \mathrm{dx}' \, \mathrm{dx}$$

Don't worry whether you are summing from $i : (n-1)$ or $(i-1) : n$. Use interval (x, n) for integral as these edge terms will not affect the final result.

3.1 Counting techniques

Warning: conjugation does not count. It is just a change of sign of the imagery part.

Inner Product given $u, v \in \mathbb{C}^m$, operation count for u^*v is

$$N_{\text{FLOP}} = 2m$$

m multiplications and $m-1$ additions included. $m + (m-1) \sim 2m$.

left Matrix-vector Multiplication given $v \in \mathbb{C}^m$, $A \in \mathbb{C}^{m \times n}$, v^*A can be viewed as n inner products in $\mathbb{C}^m$. So

$$N_{\text{FLOP}} = 2mn$$

right Matrix-vector Multiplication given $v \in \mathbb{C}^n$, $A \in \mathbb{C}^{m \times n}$, Av can be viewed as finding linear combination of columns of A.

$$N_{\text{FLOP}} = 2mn$$

There are n scalar multiplications in $\mathbb{C}^m$ taking mn operations and adding n vectors in $\mathbb{C}^m$ taking mn operations.

Matrix Multiplication Given $A \in \mathbb{C}^{m \times r}, B \in \mathbb{C}^{r \times n}$, AB can be viewed as applying A to columns of B, so

$$N_{\text{FLOP}} = n(2mr) = 2mnr$$

Let us also recall the big O notation: $f(x) = O(g(x))$ if there exists $M > 0$ s.t.

$$|f(x)| < Mg(x)$$

g is usually called polynomials or $log(x), e^x$.
The symbol $\sim$ used above is defined as: $f(x) \sim g(x)$ iff

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} = 1$$

Big O and $\sim$ should be distinguished: $x \not\sim 2x$ but $2x = O(x)$.

4 Matrix Norms

Recall matrix norm is a function $\|\cdot\|$: $K^{m \times n} \to \mathbb{R}$ where K is a field s.t.

- $\|A\| \geq 0$ and $\|A\| = 0$ iff $A = 0$.
- For $\alpha \in K$, $\|\alpha A\| = |\alpha| \|K\|$.
- $\|A + B\| \leq \|A\| + \|B\|$

actually, most matrix norms in our course satisfy multiplicative inequality:

$$\|AB\| \leq \|A\| \|B\|$$

For example, for Frobenius norm (L_2 norm of the vector form by flattening out the matrix): $\|AB\|_F \leq \|A\|_F \|B\|_F$

Proof. Suppose $A \in K^{m\times n}, B \in K^{n\times p}$

$$\|AB\|_F = \sum_{i=1}^{m}\sum_{j=1}^{p}\left(\sum_{k=1}^{n} A_{ik}B_{kj}\right)^2$$

by Cauchy-Schwarz:

$$(\sum_{k=1}^{n} A_{ik}B_{kj})^2 \leq \left(\sum_{k=1}^{n} A_{ik}^2\right)\left(\sum_{k=1}^{n} B_{jk}^2\right) = \sum_{k=1}^{n}\sum_{l=1}^{n} A_{ik}^2 B_{lj}^2$$

where the equality holds by multiplying terms in two brackets. Then

$$\|AB\|_F^2 \leq \sum_{i=1}^{m}\sum_{j=1}^{p}\sum_{k=1}^{n}\sum_{l=1}^{n} A_{ik}^2 B_{lj}^2 = \left(\sum_{i=1}^{m}\sum_{k=1}^{n} A_{ik}^2\right)\left(\sum_{j=1}^{p}\sum_{l=1}^{n} B_{lj}^2\right) = \|A\|_F\|B\|_F$$

□

Proposition 4.1. *Induction 2-norm is exactly $\sqrt{\sigma_{\max}(A)}$, where $\sigma_{\max}(A)$ is the largest singular value. (See definition of induction and Frobenius norms in official notes) Note: singular values are always non-negative.*

Proof. Finding $\|A\|^2$ (without explicit note, $\|A\|$ means induced 2-norm) is equivalent to maximising $\|A\boldsymbol{x}\|_2^2$ with constraint $\|\boldsymbol{x}\|_2^2 = 1$. So using Lagrange multiplier λ, maximum point $\boldsymbol{x}_{\max}^*$ will be one of the stationary points of the following function

$$F(\boldsymbol{x}) = \|A\boldsymbol{x}\|_2^2 - \lambda(\|\boldsymbol{x}\|_2^2 - 1)$$

Note $\|A\boldsymbol{x}\|^2 = \langle A\boldsymbol{x}, A\boldsymbol{x}\rangle = \langle A^*A\boldsymbol{x}, \boldsymbol{x}\rangle$ as A^* is the adjoint of A. It is an exercise to show that $\nabla\langle T\boldsymbol{x}, \boldsymbol{x}\rangle = 2T\boldsymbol{x}$. So if

$$\nabla F = 2A^*Ax - 2\lambda x = 0$$

then $A^*Ax = \lambda x$, which means λ is Eigenvalue of A^*A (or singular value of A) and x^* is the corresponding Eigenvector. Plugging $\boldsymbol{x}^*$ back into $\|A\boldsymbol{x}\|_2^2$ gives

$$\langle A^*Ax^*, x^*\rangle = \lambda\|x^*\|^2 = \lambda$$

last equality holds as $\|x\|^2 = 1$ for any stationary point of F. So the λ giving maximum value is the largest Eigenvalue of A^*A i.e. largest singular value, $\sigma_{\max}(A)$. □

Corollary.

$$\inf_{\|x\|_2=1} \|Ax\|^2 = \sigma_{\min}(A)$$

Proof. Since minimum point $\boldsymbol{x}_{\min}^*$ will also be hidden in stationary points of $F(\boldsymbol{x})$, and minimum Eigenvalue of A^*A gives this minimum value of $\|A\boldsymbol{x}\|^2$. □

Corollary. *Assume A is invertible, $\|A^{-1}\|^2 = 1/\sigma_{\min}(A)$*

Proof. By the previous corollary,

$$\sup_{\|x\|\neq 0} \|x\|/\|Ax\| = 1/\sqrt{\sigma_{\min}(A)}$$

so

$$\sup_{\|A^{-1}y\|\neq 0} \|A^{-1}y\|/\|y\| = \sqrt{1/\sigma_{\min}(A)}$$

where we made substitution $x = A^{-1}y$. And since $A^{-1}y \neq 0 \Leftrightarrow y \neq 0$, LHS of above equation is exactly the definition of $\|A^{-1}\|$. □

Corollary. $\kappa(A) = \sqrt{\frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}}$

Proof. Simply use definition $\kappa(A) = \|A\|\|A^{-1}\|$ □

Though we already have an explicit evaluation of operator norm, for some special cases, operator norm can be found using the definition.

Proposition 4.2 (Operator norm for outer product). *Let $A = uv^*$ where $u, v \in \mathbb{R}^m$, then $\|A\|_2 = \|u\|\|v\|$*

Proof. Note $Ax = uv^*x = (v^*x)u$ by Proposition 1.2. So

$$\|Ax\| = |v^*x|\|u\| \leq \|v\|\|x\|\|u\|$$

now we have to pick x to attain this upper bound. By equality condition for Cauchy-Schwartz, $x = v/\|v\|$ works. □

5 Accuracy, Stability

Recall absolute error and relative errors for approximating x as $\tilde{x}$ are defined as follows.

$$E_{\text{abs}}(\tilde{x}) = \|x - \tilde{x}\|, \quad E_{\text{rel}}(\tilde{x}) = \frac{\|x - \tilde{x}\|}{\|x\|}$$

When it comes to algorithms, we need to analyse how accurate estimation $\tilde{f}(x)$ would be for $f(x)$, where $\tilde{f}$ is the algorithm. (Floating point rounding error for x is ignored, as it is accurate to machine epsilon) Developers care about $E_{\text{abs}}(\tilde{f}(x)) = \|\tilde{f}(x) - f(x)\|$ or the relative error version, which are called forward errors. But it is easier to analyse input space. One can look for a $\tilde{x} = x + \Delta x$ near x that corresponds to the value of $\tilde{f}(x)$ i.e.

$$f(x + \Delta x) = \tilde{f}(x)$$

$\|\Delta x\|$ is called (absolute) *backward error*. There might be many Δx satisfying this condition. The smallest one (with the smallest absolute value) is taken. An algorithm is *backward stable* if, for all x, estimated $\tilde{f}(x)$ has a small backward error. Basic operations like addition and subtraction are backward stable.

Algorithms for computing $\cos x$ are not backward stable but rather satisfy a milder condition:

$$f(x + \Delta x) = \tilde{f}(x) + \Delta y$$

where we require $|\Delta x| < \epsilon|x|, |\Delta y| < \epsilon|y|$ for small $\epsilon > 0$. As you may have guessed, this error has the name *forward-backwards error*. An algorithm is *(numerically) stable* if for all x, estimated $\tilde{f}(x)$ has a small forward-backwards error.

You may recognise that Δy in the above definition is roughly the forward error. So if you want to obtain forward error from backward error, multiplying $\Delta y/\Delta x$ would be a good estimate.

Assume f is twice differentiable,

$$f(x + \Delta x) - f(x) = f'(x)\Delta x + O((\Delta x)^2)$$

So indeed $f'(x)$ measures how a small perturbation to x would change the value of $f(x)$, $|f'(x)|$ is called (absolute) condition number κ_{abs}. If x, f are vectors and vector functions instead, a similar concept of "maximum relative change" is defined:

$$\kappa_{\text{abs}} = \sup_{\Delta x \neq 0} \frac{\|f(\boldsymbol{x} + \Delta x) - f(\boldsymbol{x})\|}{\|\Delta x\|}$$

There is another version of condition number: if relative perturbation of x is considered instead, Maclaurin expansion above can be modified to:

$$\frac{f(x + \Delta x) - f(x)}{f(x)} = \frac{f'(x)x}{f(x)} \frac{\Delta x}{x} + O((\Delta x)^2)$$

so $f'(x)x/f(x)$ measures how change of x relative to x affects relative change of $f(x)$. Therefore, we define relative condition number as

$$\kappa_{\text{rel}} := \left| \frac{f'(x)x}{f(x)} \right|$$

vector version is given below

$$\kappa_{\text{rel}} := \sup_{\Delta x \neq 0} \frac{\|f(\boldsymbol{x} + \Delta x) - f(\boldsymbol{x})\|\|\boldsymbol{x}\|}{\|f(\boldsymbol{x})\|\|\Delta x\|}$$

Condition number gives a relation between errors: forward error is roughly bounded above by condition number × backward error. So algorithm with a large condition number (ill-conditioned) may have a large forward error even if the backward error is small.

5.1 Cancellation

Catastrophic cancellation occurs when two very close numbers are subtracted. The absolute error of subtraction is the same as addition, but the relative error is huge when your actual result is close to 0. For example, if $a = 1.1122, b = 1.11$, but your computer only has 2 decimal places of precision, $\tilde{a} - \tilde{b} = 0$! All accurate digits are cancelled.

Using mathematical language, assume $x = a - b$ and estimated $\tilde{x} = \tilde{a} - \tilde{b}$, where $\tilde{a} = a(1 + \Delta a), \tilde{b} = b(1 + \Delta b)$, then

$$\left|\frac{x - \hat{x}}{x}\right| = \left|\frac{-a\Delta a + b\Delta b}{a - b}\right| \leq \max(|\Delta a|, |\Delta b|)\frac{|a|+|b|}{|a - b|}$$

when $a \approx b$, the bound is large, so the relative error can be huge.

If you only want $a - b$, the cancellation would not cause any problems. Or, for expressions like $x + (y - z)$ where $y \approx z$, you don't have to do anything about this cancellation problem. But this effect is amplified when $a - b$ is used in other calculations with larger condition numbers (or less numerically stable), for example, multiplication. Consider $x^2 - y^2 = (x - y)(x + y)$ with $x \approx y$, $x \ominus y = (x - y)(1 + \epsilon_l)$ where ϵ_l is large. Then

$$(x \ominus y)(x \oplus y) \approx (x + y)(x - y) + \epsilon_l(x + y)$$

(ignore $\oplus$ error as it is tiny) so absolute error is $|(x + y)\epsilon_l|$.

One possible solution to this cancellation is to avoid subtraction $a - b$ where $a \approx b$, especially when a, b have large errors. (See Exercise 3.20 as an example) Other solutions are improving precision, or simply accept this as part of the algorithm.

5.2 Linear equation solving

For matrix-vector systems(general linear systems) like solving linear systems or matrix decomposition, backward error can easily found. For example, if we are finding approximate solution $\tilde{x}$ to $Ax = b$ with b fixed, the residual $r := b - A\tilde{x}$, or strictly speaking its norm $\|b - A\tilde{x}\|$, is the main error term we care. But since scaling A, b will scale this difference, the relative residual

$$\rho(\tilde{x}) = \frac{\|b - A\tilde{x}\|}{\|A\|\|\tilde{x}\|}$$

is considered instead.

Proposition 5.1 (Relative residual and backward error)**.**

$$\rho(\tilde{x}) = \min\left\{\frac{\|\Delta A\|_2}{\|A\|_2} \,:\, (A + \Delta A)\tilde{x} = b\right\}$$

note LHS is exactly the (relative) backwards error.

Proof. If $(A + \Delta A)\tilde{x} = b$, then $r = \Delta A\tilde{x}$ so $\|r\| \leq \|\Delta A\|\|\tilde{x}\|$. That means

$$\frac{\|\Delta A\|}{\|A\|} \geq \frac{\|r\|}{\|A\|\|\tilde{x}\|} = \rho(\tilde{x})$$

To prove this minimum is attained, we need to find ΔA s.t. $\|r\| = \|\Delta A\|\|\tilde{x}\|$. Try the outer product $r\tilde{x}^*$, we have $\|\Delta A\| = \|r\|\|\tilde{x}\|$(by proposition 4.2), only differs in a norm. So let $\Delta A = r\tilde{x}^*/(\|\tilde{x}\|^2)$. And it satisfies $(A+\Delta A)\tilde{x} = b$. So the minimum is attained. □

6 LU Decomposition

Recall that when using Gaussian elimination to solve a system of linear equations, the matrix of coefficients of equations is transformed into an upper triangular matrix by elementary row operations. Upper triangular systems

are easier to solve, and the inverse of upper triangular matrices is easier to compute. It turns out that these row operations can be represented by lower triangular matrices. For example,

$$\underbrace{\begin{pmatrix} 1 & 0 \\ -2 & 1 \end{pmatrix}}_{L_1}\underbrace{\begin{pmatrix} 2 & -1 \\ 4 & 3 \end{pmatrix}}_{A} = \begin{pmatrix} 2 & -1 \\ 0 & 5 \end{pmatrix}$$

left multiplication by L_1 reads as: keep the first row, substitute the second row by $r_2 - 2r_1$ where r_i are rows of A. This clears everything in the first column below the first entry. To understand this firmly, the following is a multiplication of the lower triangular matrix at the second step of Gaussian elimination

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3 & 1 \end{pmatrix}}_{L_2}\underbrace{\begin{pmatrix} 1 & 2 & 3 \\ 0 & 2 & 2 \\ 0 & 6 & 4 \end{pmatrix}}_{A_1} = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 2 & 2 \\ 0 & 0 & -2 \end{pmatrix}$$

Now it is clear that at iteration k, the lower triangular matrix of the form

$$L_k := \begin{pmatrix} 1 & & \text{column } k & & \\ & \ddots & & & \\ & & 1 & & \\ & & -\boldsymbol{l}_k & \ddots & \\ & & & & 1 \end{pmatrix}, \quad \boldsymbol{l}_k := \begin{pmatrix} a_{k+1,k}/a_{k,k} \\ a_{k+2,k}/a_{k,k} \\ \vdots \\ a_{m,k}/a_{k,k} \end{pmatrix}$$

should be used, where $a_{i,j}$ are taken from the the result matrix A_{k-1} from iteration $k-1$.

Collecting all the information above, we have the result after $m-1$ iterations ($m-1$'th iteration leaves submatrix of size 1×1, the matrix is already upper triangular)

$$L_{m-1}L_{m-2}\cdots L_1 A = U$$

note L_i are invertible as $\det L_i = \prod_i L_{ii} = 1 \neq 0$, so

$$A = (L_1^{-1}\cdots L_{m-1}^{-1})U$$

that completes the LU decomposition. Thinking of what L_i does to the rows, it is easy to see that the inverse of L_k is

$$L_k^{-1} = \begin{pmatrix} 1 & & \text{column } k & & \\ & \ddots & & & \\ & & 1 & & \\ & & \boldsymbol{l}_k & \ddots & \\ & & & & 1 \end{pmatrix}, \quad \boldsymbol{l}_k := \begin{pmatrix} a_{k+1,k}/a_{k,k} \\ a_{k+2,k}/a_{k,k} \\ \vdots \\ a_{m,k}/a_{k,k} \end{pmatrix}$$

we add the row k, scaled by $a_{j,k}/a_{k,k}$, back to row j. And because L_k^{-1} will not affect the first $k-1$ rows, when $s > k$,

$$L_k^{-1}L_s^{-1} = \begin{pmatrix} 1 & & \text{column } k & & \text{column } s & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & | & \ddots & & & \\ & & \boldsymbol{l}_k & & 1 & & \\ & & | & & \boldsymbol{l}_s & \ddots & \\ & & & & & & 1 \end{pmatrix}$$

Hence, $L := L_1^{-1}\cdots L_{m-1}^{-1}$ can be formed by filling vectors $\boldsymbol{l}_k$ to identity matrix in each operation below the main diagonal.

You may noticed that if $a_{k,k} = 0$, construction of $\boldsymbol{l}_k$ fails. Pivoting (exchanging rows) is required here, which means an additional permutation matrix P_k will be multiplied. And this ultimately results in the so-called *PLU decomposition*.

More surprisingly, the LU decomposition of positive definite Hermitian matrix A takes the form U^*U where U is upper triangular. This is the *Cholesky decomposition*.

7 Eigenvalues

The analytical method of deciding the Eigenvalues of a matrix is to find roots to the characteristic polynomial. Due to the high numerical instability of algorithms for finding roots, this may not be a practical route.

Iterative methods are devised to find the Eigenvalues; the first step is to find the regions where Eigenvalues lie roughly. One famous method is the Geršgorin circle

Theorem 7.1. *Given matrix $A = (a_{ij})$, if circles R_i in complex plane are defined by*

$$R_i := \left\{ z \in \mathbb{C} \,:\, |z - a_{ii}| \leq \sum_{j=1, j\neq i}^{n} |a_{ij}| \right\}$$

then Eigenvalues of A are all in $\bigcup_{i=1}^{n} R_i$ and union of any k circles that do not intersect the remaining $n-k$ circles contains precisely k(counting up to multiplicity) of the Eigenvalues.

Recall some properties related to Eigenvalues and Eigenvectors from linear algebra

- Eigenvectors of DISTINCT Eigenvalues are linearly independent
- orthogonal set of non-zero vectors is linearly independent
- Gram-Schmidt algorithm takes linearly independent vectors and turns them into orthogonal vectors.
- Eigenvalues of similar matrices are the same, and if $A = S^{-1}BS$, then eigenvectors of B are $S\boldsymbol{x}$ where $\boldsymbol{x}$ is an eigenvector of A with the same Eigenvalue.
- matrix is similar to a diagonal matrix $\Leftrightarrow$ the matrix has n linearly independent Eigenvectors.

7.1 Schur Decomposition

Schur decomposition is a powerful triangularisation tool which applies to all complex square matrices:

Theorem 7.2. *For any matrix $A \in \mathbb{C}^{n\times n}$, exists unitary U s.t. $U^{-1}AU$ is upper triangular.*

Proof. Recall Jordan normal form,

$$P^{-1}AP = J$$

where P is non-singular. If QR decomposition of P is $P = UR$ (U is unitary), then

$$R^{-1}U^{-1}AUR = J \quad \Rightarrow \quad U^{-1}AU = RJR^{-1}$$

where RJR^{-1} is upper triangular because R, J are both upper triangular. □

It can be shown that when A is normal, i.e. $A^*A = AA^*$, then Schur decomposition gives a diagonal matrix. Hermitian matrices are normal. Recall that the Eigenvalues of Hermitian matrices are real.

7.2 Power Iteration/Power method

Assumption: A has n eigenvalues with associated linear independent Eigenvectors, and there is a unique eigenvalue λ_1 satisfying

$$|\lambda_1| > |\lambda_i| \quad \forall i \neq 1$$

Remark. even if the assumption is not satisfied, there is still a chance that the power method works.

The idea is given any chosen initial vector $\boldsymbol{x}$, it can be expressed using Eigenvectors mentioned in the assumption because n LI vectors form a basis,

$$\boldsymbol{x} = \sum_{j=1}^{n} \beta_j \boldsymbol{v}_j$$

iteratively multiplying A gives

$$A^k \boldsymbol{x} = \sum_{j=1}^{n} \beta_j \lambda_j^k \boldsymbol{v}_j$$

factorising λ_1^k yields

$$A^k \boldsymbol{x} = \lambda_1^k \sum_{j=1}^{n} \beta_j \left(\frac{\lambda_j}{\lambda_1}\right)^k \boldsymbol{v}_j$$

by the fact that $|\lambda_1| > |\lambda_j|$ for $j \neq 1$, $(\lambda_j/\lambda_1)^k \to 0$, so

$$\lim_{k\to\infty} A^k \boldsymbol{x} = \lim_{k\to\infty} \lambda_1^k \beta_1 \boldsymbol{v}_1$$

If $\beta_1 \neq 0$, we obtain an eigenvector. However, if $|\lambda_1| < 1$, $\lambda_1^k \to 0$ and may result in underflow. Overflow may happen if $|\lambda_1| > 1$. The simplest scaling is to normalise the vector at every iteration. Some other scaling methods can accelerate convergence, but they will not be discussed here.

Drawbacks: no guarantee that dominant eigenvalue (the eigenvalue with the largest magnitude) is unique; slow convergence when eigenvalues are too close, because convergence is related to $(\lambda_j/\lambda_1)^k$; not known whether $\beta_1 \neq 0$ for chosen initial vector $\boldsymbol{x}$.

7.2.1 Rayleigh Iteration

Power iteration can be altered to find other Eigenvectors: if $\{\lambda_i\}$ are Eigenvectors of A, then for any $\mu \neq \lambda_i$, eigenvalues of $(A-\mu I)^{-1}$ are $\{1/(\lambda_i - q)\}$. So power iteration on $(A-\mu I)^{-1}$ yields the eigenvector associated to the largest $1/(\lambda_i - q)$, i.e. the eigenvalue λ_i closest to q. To determine the vector for the next step, $(A - \mu I)\boldsymbol{x}^{(m)} = \boldsymbol{x}^{(m-1)}$ is solved using Gaussian elimination with pivoting (or any other linear equation solver)

The algorithm will be accelerated if the choice of μ is close to the true Eigenvalue. This is done by setting μ as the Rayleigh quotient $(\boldsymbol{x}^T A\boldsymbol{x})/(\boldsymbol{x}^T\boldsymbol{x})$ each time: if $\boldsymbol{x}$ is indeed an Eigenvector, i.e. $A\boldsymbol{x} = \lambda\boldsymbol{x}$, then

$$\frac{\boldsymbol{x}^T A\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}} = \frac{\lambda \boldsymbol{x}^T\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}} = \lambda$$

and it can be shown that if $\boldsymbol{x}$ is close to an Eigenvector, the Rayleigh quotient is close to the Eigenvalue.

7.2.2 Deflation

Given matrix A with eigenvalues $\lambda_1, \cdots, \lambda_n$(WOLG assume eigenvalues are in descending order) and associated Eigenvectors $\boldsymbol{v}^{(1)}, \cdots, \boldsymbol{v}^{(n)}$, where multiplicity of λ_1 is 1. One can construct matrix B with Eigenvectors same as A but the dominating eigenvalue λ_1 deflated to 0 by picking $\boldsymbol{x}$ s.t. $\boldsymbol{x}^T\boldsymbol{v}^{(1)} = 1$, and letting

$$B := A - \lambda_1 \boldsymbol{v}^{(1)}\boldsymbol{x}^T$$

Suppose the Eigenvectors of B are $\boldsymbol{v}^{(1)}, \boldsymbol{w}^{(2)}, \cdots, \boldsymbol{w}^{(n)}$, we can get back $\boldsymbol{v}^{(i)}$ by

$$\boldsymbol{v}^{(i)} = (\lambda_i - \lambda_1)\boldsymbol{w}^{(i)} + \lambda_1(\boldsymbol{x}^T\boldsymbol{w}^{(i)})\boldsymbol{v}^{(1)} \qquad (*)$$

(the proof can be found in *The algebraic eigenvalue problem* by Wilkinson, J. H. on page 596)

$\boldsymbol{x}$ in the above procedure can be chosen by Wielandt deflation: choose i s.t. $v_i^{(1)}$ is nonzero, let

$$\boldsymbol{x} := \frac{1}{\lambda_1 v_i^{(1)}} \begin{pmatrix} a_{i1} \\ \vdots \\ a_{in} \end{pmatrix}$$

where a_{ij} are entries of matrix A. Verify:

$$\begin{aligned} \boldsymbol{x}^T\boldsymbol{v}^{(1)} &= \frac{1}{\lambda_1 v_i^{(1)}} \begin{pmatrix} a_{i1} & \cdots & a_{in} \end{pmatrix} \begin{pmatrix} v_1^{(1)} \\ \vdots \\ v_n^{(1)} \end{pmatrix} \\ &= \frac{1}{\lambda_1 v_i^{(1)}} \sum_{j=1}^{n} a_{ij} v_j^{(1)} \\ &= \frac{1}{\lambda_1 v_i^{(1)}} [A\boldsymbol{v}^{(1)}]_i \\ &= \frac{1}{\lambda_1 v_i^{(1)}} \lambda_1 v_i^{(1)} = 1 \end{aligned}$$

Note that with this choice of $\boldsymbol{x}$, ith row of B will be a zero row. If $B\boldsymbol{w} = \lambda\boldsymbol{w}$, then $w_i = 0$ and therefore ith column of B can be removed, together with the useless zero row to form $B' \in \mathbb{C}^{(n-1)\times(n-1)}$. We still have $B'\boldsymbol{w} = \lambda\boldsymbol{w}$ if $\lambda \neq 0$. Therefore, Eigenvalues of B' are $\lambda_2, \cdots, \lambda_n$. Applying power method to B' yields Eigenvector $(\boldsymbol{w}^{(2)})'$ with Eigenvalue λ_2. Inserting the 0 back to $(\boldsymbol{w}^{(2)})'$ gives $\boldsymbol{w}^{(2)}$, the Eigenvector of B with Eigenvalue λ_2. And using the formulae (*), $\boldsymbol{v}^{(2)}$ can be found.

Due to this algorithm's round-off errors, the deflation iteration result will not be used directly. Still, it can be used as an initial value for inverse power iteration on the original matrix using the estimated $\boldsymbol{v}^{(2)}$.

7.3 QR algorithm

Section 5.3 *Similarity transformation to upper Hessenberg form* in lecture notes explains how to find a matrix in upper Hessenberg form similar to given matrix A using a modified version of the QR factorisation using householder reflection. When this algorithm is applied to a real symmetric matrix, the result is a symmetric Hessenberg matrix, which is symmetric and tridiagonal. QR algorithm further refines the result, finding a diagonal matrix similar to the given tridiagonal matrix so we can directly read the eigenvalues from the diagonal.

If 0 is observed on the sub-diagonals at some step, say

$$A = \begin{pmatrix} a_1 & b_2 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ b_2 & a_2 & b_3 & & & & & & & \\ 0 & b_3 & a_3 & \ddots & & & & & & \\ \vdots & \ddots & \ddots & \ddots & b_{j-1} & & & & & \\ \vdots & & \ddots & b_{j-1} & a_{j-1} & 0 & & & & \\ \vdots & & & \ddots & 0 & a_j & b_{j+1} & & & \\ \vdots & & & & \ddots & b_{j+1} & a_{j+1} & \ddots & & \\ \vdots & & & & & \ddots & \ddots & \ddots & \ddots & \\ \vdots & & & & & & \ddots & \ddots & \ddots & b_n \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & b_n & a_n \end{pmatrix}$$

i.e. $b_j = 0$. Then clearly, the matrix can be divided into two submatrices that greatly simplify the algorithm. Further, if $b_2 = 0$, a_1 is already an Eigenvalue. (try calculated Ae_1, you will see it is $a_1 e_1$)

QR algorithm modifies $A^{(0)} := A$ forming a sequence $\{A^{(k)}\}$. At each iteration, $A^{(i)}$ is factorised as $A^{(i)} = Q^{(i)}R^{(i)}$ where $Q^{(i)}$ is orthogonal, $R^{(i)}$ is upper triangular, and the next matrix is determined by $A^{(i+1)} := R^{(i)}Q(i)$. Note $R^{(i)} = (Q^{(i)})^T A^{(i)}$ so

$$A^{(i+1)} = (Q^{(i)})^T A^{(i)} Q(i)$$

i.e. $A^{(i+1)}$ is similar to $A^{(i)}$. Therefore, Eigenvalues are preserved at each iteration. By design of QR factorisation, $Q^{(i)}$ will be a lower banded matrix with lower bandwidth 1 and $R^{(i)}$ will be an upper banded matrix with upper bandwidth 2. So $A^{(i+1)}$ will also be a tridiagonal matrix.

7.3.1 Essence of QR

The reason QR works is because it is equivalent to simultaneous iteration with initial basis chosen as the canonical basis. Simultaneous iteration is an algorithm that performs power algorithm to a given orthonormal basis $\{v_1, \cdots, v_n\}$, i.e. the process is

$$\text{(iteration 0)} \quad \{v_1, \cdots, v_n\}$$

$$\text{(iteration 1)} \quad \{v_1^{(1)} := Av_1, \cdots, v_n^{(1)} := Av_n\}$$

$$\cdots$$

$$\text{(iteration k)} \quad \{v_1^{(k)} := A^k v_1, \cdots, v_n^{(k)} := A^k v_n\}$$

but all vectors will converge to the same Eigenvector q_1 (the one corresponding to the largest eigenvalue λ_1), in order to obtain different Eigenvectors, we need to orthogonalise at each iteration. This can be performed by Gram-Schmidt, in the first step, pull the component of $A^k v_1$ (which is close to q_1) from the rest, so that q_2 becomes dominant among $v_2^{(k)}, \cdots, v_n^{(k)}$, then the second step pulls out the dominant q_2 from $v_3^{(k)}, \cdots, v_n^{(k)}$. This would end up in the sequence converging to an orthonormal set of Eigenvectors. The rigorous proof will not be given here.

In practice, simultaneous iteration is implemented by filling the initial orthonormal basis $\{v_1, \cdots, v_n\}$ into a matrix,

- for $i = 1, 2, \cdots$:
  - $\hat{Q}^{(0)}$[:, i] $\leftarrow v_i$
- for $k = 1, 2, \cdots$:
  - $Z \leftarrow A\hat{Q}^{(k-1)}$ (this is essentially performing power iteration to the n vectors)
  - do QR decomposition on Z, i.e. $\hat{Q}^{(k)}, \_\_ \leftarrow \text{qr}(Z)$ (this is essentially Gram-Schmidt, but it is implemented using the more stable householder QR decomposition)

.

.

In order to show QR algorithm is equivalent to simultaneous iteration with $\hat{Q}^{(0)} := I$, we first need to note that $\hat{Q}^{(k)}$ in simultaneous iterations and $Q^{(k)}$ in QR are DIFFERENT matrices. In short, $Q^{(k)}$ in QR algorithm only represents information of $A^{(k)}$ (recall $A^{(k)}$ a matrix similar to A produced in the kth iteration of QR algorithm), but $\hat{Q}^{(k)}$ in simultaneous iteration stores all the the information about A. By design of simultaneous iteration, $\hat{Q}^{(k)}$ will gradually converge to an orthonormal basis consisting of Eigenvectors of A at a rate of $C := \max_k |\lambda_{k+1}|/|\lambda_k|$. (we assume $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$, see the proof of this convergence rate on (page 214, Trefethen and Bau, 1997)) For this reason, two different symbols are used. Later we will show that $\hat{Q}^{(k)}$ from simultaneous iteration and $Q^{(k)}$ from QR algorithm are related by

$$\hat{Q}^{(k)} = Q^{(1)} \cdots Q^{(k)} \tag{1}$$

Conversely, the matrix $A^{(k)}$ that is kept similar to matrix A in QR algorithm is not present in simultaneous equation. $A^{(k)}$ from QR algorithm is related to $\hat{Q}^{(k)}$ produced from simultaneous iterations by

$$A^{(k)} = (\hat{Q}^{(k)})^T A \hat{Q}^{(k)} \tag{2}$$

You can see that this guarantees $A^{(k)} \sim A$. Again, see the complete proof later.

A final remark is that $Q^{(k)}, R^{(k)}$ in the QR algorithm can be used to find A^k. Consider the first iteration,

$$A = Q^{(1)} R^{(1)}$$

$$A^{(1)} = R^{(1)} Q^{(1)} = Q^{(2)} R^{(2)}$$

So

$$A^2 = (Q^{(1)} R^{(1)})(Q^{(1)} R^{(1)}) = Q^{(1)} (R^{(1)} Q^{(1)}) R^{(1)} = Q^{(1)} Q^{(2)} R^{(2)} R^{(1)}$$

Proceeding in similar way, for any integer k,

$$A^k = Q^{(1)} \cdots Q^{(k)} R^{(k)} \cdots R^{(1)}$$

therefore, keeping a running product of $R^{(k)}$ would be helpful to simplify this expression, i.e. let $\hat{R}^{(k)} := R^{(k)} \cdots R^{(1)}$, then

$$A^k = \hat{Q}^{(k)} \hat{R}^{(k)} \tag{3}$$

we will also calculate this running product in the simultaneous iteration and later prove that equation (3) is also satisfied.

To prove the relations (1), (2), (3), we need to keep record of of $A^{(k)}$, $\hat{Q}^{(k)}, \hat{R}^{(k)}$ for both QR and simultaneous iteration by:

- adding definitions $A^{(k)} := (\hat{Q}^{(k)})^T A \hat{Q}^{(k)},\ \hat{R}^{(k)} := R^{(k)} \cdots R^{(1)}$ to simultaneous iteration
- adding definitions $\hat{Q}^{(k)} := Q^{(1)} \cdots Q^{(k)},\ \hat{R}^{(k)} := R^{(k)} \cdots R^{(1)}$ to QR algorithm

The initial terms of the running products, $\hat{Q}^{(0)}, \hat{R}^{(0)}$, are defined as I, to guarantee that $\hat{Q}^{(k)} = \hat{Q}^{(k-1)} Q^{(k)},\ \hat{R}^{(k)} = R^{(k)} \hat{R}^{(k-1)}$ are satisfied even when $k = 1$. Just like you will defined $a^0 := 1$. And we define $A^{(0)} := A$ in simultaneous iteration.

The full process of simultaneous iteration and QR algorithm with the additional definitions are given below as a reference while you go through the proof of theorem.

Simultaneous iteration

- $\hat{Q}^{(0)} \leftarrow I,\ \hat{R}^{(0)} \leftarrow I,\ A^{(0)} \leftarrow A$
- for $k = 1, 2, \cdots$
  - $Z \leftarrow A \hat{Q}^{(k-1)}$
  - $\hat{Q}^{(k)}, R^{(k)} \leftarrow \mathrm{qr}(Z)$
  - (ADDITIONAL) $A^{(k)} := (\hat{Q}^{(k)})^T A \hat{Q}^{(k)}$
  - (ADDITIONAL) $\hat{R}^k := R^{(k)} \cdots R^{(1)}$

QR algorithm

- $\hat{Q}^{(0)} \leftarrow I,\ \hat{R}^{(0)} \leftarrow I,\ A^{(0)} \leftarrow A$
- for $k = 1, 2, \cdots$
  - $Q^{(k)}, R^{(k)} \leftarrow \mathrm{qr}(A^{(k-1)})$
  - $A^{(k)} \leftarrow R^{(k)} Q^{(k)}$
  - (ADDITIONAL) $\hat{Q}^{(k)} := Q^{(1)} \cdots Q^{(k)}$
  - (ADDITIONAL) $\hat{R}^{(k)} := R^{(k)} \cdots R^{(1)}$

Theorem 7.3. *With the above new definition, $A^{(k)}$, $\hat{Q}^{(k)}, \hat{R}^{(k)}$ obtained from QR algorithm and simultaneous iteration with $\hat{Q}^{(0)} := I$ are the same. Further, we have relationships*

$$A^k = \hat{Q}^{(k)}\hat{R}^{(k)}, \quad A^{(k)} := (\hat{Q}^{(k)})^T A\hat{Q}^{(k)}$$

Proof. Use proof by induction. $A^0 = \hat{R}^{(0)} = \hat{Q}^{(0)} = I$ for both algorithms, so $A^k = \hat{Q}^{(k)}\hat{R}^{(k)}$ is satisfied for $k = 0$. $A^{(k)} := (\hat{Q}^{(k)})^T A\hat{Q}^{(k)}$ is also trivially satisfied by both algorithms when $k = 0$ because $\hat{Q}^{(0)} = I$.

Inductive step – simultaneous iteration: we first prove the two relationships for simultaneous iteration. $A^{(k)} := (\hat{Q}^{(k)})^T A\hat{Q}^{(k)}$ by our arbitrary definition on simultaneous iteration.

$$\begin{aligned}
A^k &= AA^{k-1} \\
&= A\hat{Q}^{(k-1)}\hat{R}^{(k-1)} \quad \text{by inductive hypothesis} \\
&= Z\hat{R}^{(k-1)} = \hat{Q}^{(k)}R^{(k)}\hat{R}^{(k-1)} \\
&= \hat{Q}^{(k)}\hat{R}^{(k)} \quad \text{because} \hat{R}^{(k)} = R^{(k)}\hat{R}^{(k-1)}
\end{aligned}$$

Inductive step – QR algorithm:
First prove the formulae for A^k,

$$\begin{aligned}
A^k &= AA^{k-1} \\
&= A\hat{Q}^{(k-1)}\hat{R}^{(k-1)} \quad \text{by inductive assumption} \\
&= \hat{Q}^{(k-1)}A^{(k-1)}\hat{R}^{(k-1)} \quad \text{because by another inductive assumption, } A^{(k-1)} = (\hat{Q}^{(k-1)})^T A\hat{Q}^{(k-1)} \\
&= \hat{Q}^{(k-1)}Q^{(k)}R^{(k)}\hat{R}^{(k-1)} \quad \text{by design of QR algorithm} \\
&= \hat{Q}^{(k)}\hat{R}^{(k)} \quad \text{by definitions of } \hat{Q}^{(k)}, \hat{R}^{(k)}
\end{aligned}$$

Now recall that each iteration of QR is similarity transformation of the previous step,

$$\begin{aligned}
A^{(k)} &= R^{(k)}Q^{(k)} \quad \text{by design of QR algorithm} \\
&= IR^{(k)}Q^{(k)} = (Q^{(k)})^T Q^{(k)}R^{(k)}Q^{(k)} \quad \text{as } Q^{(k)} \text{ is orthogonal} \\
&= (Q^{(k)})^T A^{(k-1)}Q^{(k)} \quad \text{by design of QR algorithm} \\
&= (Q^{(k)})^T (\hat{Q}^{(k-1)})^T A\hat{Q}^{(k-1)}Q^{(k)} \quad \text{by inductive assumption} \\
&= (\hat{Q}^{(k)})^T A\hat{Q}^{(k)} \quad \text{by definition of } \hat{Q}^{(k)}
\end{aligned}$$

.
Using the relation $A^{(k)} := (\hat{Q}^{(k)})^T A\hat{Q}^{(k)}$, we only need to show $\hat{Q}^{(k)}$ in two algorithms are the same. Again using induction:
Base step. $\hat{Q}^{(0)}$ are both I.
Inductive step. suppose $\hat{Q}^{(k-1)}$ are the same, in simultaneous iteration, $Z = A\hat{Q}^{(k-1)} = \hat{Q}^{(k)}R^{(k)}$ by design of algorithm, and $A\hat{Q}^{(k-1)} = AQ^{(1)}\cdots Q^{(k-1)}$ (here, $Q^{(j)}$ are taken from QR algorithm) by inductive assumption. Therefore,

$$\hat{Q}^{(k)}R^{(k)} = AQ^{(1)}\cdots Q^{(k-1)} \tag{4}$$

The aim is to prove $\hat{Q}^{(k)} = Q^{(1)}\cdots Q^{(k)}$, using QR algorithm,

$$\begin{aligned}
R^{(k)}Q^{(k)} &= A^{(k)} \\
&= (\hat{Q}^{(k)})^T A\hat{Q}^{(k)} \quad \text{by relation deduced above}
\end{aligned}$$

Therefore, $A = \hat{Q}^{(k)}R^{(k)}Q^{(k)}(\hat{Q}^{(k)})^T$. Substituting this to equation (4) yields:

$$\begin{aligned}
&\hat{Q}^{(k)}R^{(k)} = \hat{Q}^{(k)}R^{(k)}Q^{(k)}(\hat{Q}^{(k)})^T Q^{(1)}\cdots Q^{(k-1)} \\
&(\Rightarrow) I = Q^{(k)}(\hat{Q}^{(k)})^T Q^{(1)}\cdots Q^{(k-1)} \\
&(\Rightarrow)\hat{Q}^{(k)}(Q^{(k)})^T = Q^{(1)}\cdots Q^{(k-1)} \\
&(\Rightarrow)\hat{Q}^{(k)} = Q^{(1)}\cdots Q^{(k)}
\end{aligned}$$

Now we can prove $\hat{R}^{(k)}$ taken from two algorithms aligns. From $A^k = \hat{Q}^{(k)}\hat{R}^{(k)}$, we have $(\hat{Q}^{(k)})^T A^k = \hat{R}^{(k)}$, which finishes the proof. □

Corollary. *$A^{(k)}$ in QR algorithm converges to diagonal matrix D (whose entries are Eigenvalues), and $\hat{Q}^{(k)}$ converges to Q, a matrix consisting of Eigenvectors corresponding to diagonal of D. Both convergence rates are $C := \max_k |\lambda_{k+1}|/|\lambda_k|$*

7.3.2 Rate of Convergence

Rate of convergence of $b_{j+1}^{(i+1)}$, the $j+1$th sub-diagonal element of $A^{(i+1)}$, to 0 is determined by the rate that $a_j^{(i+1)}$ converges to λ_j, which is related to $|\lambda_{j+1}/\lambda_j|$. The basic QR algorithm is computationally expensive because in each iteration there are operations on the whole matrices and convergence may be slow due to Eigenvalues of matrix clustering together. We already discussed an improvement: reduce matrix to tridiagonal form instead of performing QR on the original matrix. There is another one: giving a shift.

If the Eigenvalues have distinct moduli, i.e.

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

recall the shifting technique used in inverse power iteration, i.e. do QR decomposition on $A^{(i)} - \mu^{(i)}I$ instead. Now one iteration becomes

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

and the convergence rate becomes $|(\lambda_{j+1} - \mu^{(j)})/(\lambda_j - \mu^{(j)})|$. If $\mu^{(j)}$ is chosen to be close to λ_{j+1} but not λ_j.

When we can assume Eigenvalues have distinct modulus, the rate of convergence of $b_n^{(i+1)}$ is faster than $b_j^{(i+1)}$ for any other j. So this algorithm is continued until $b_n^{(i+1)} \approx 0$, in which case λ_n is approximated by $a_n^{(i+1)}$ and nth row and column are deleted. The same procedures continue on the reduced matrix to produce λ_{n-1}. A first thought of choice of shift parameter may be the Rayleigh's quotient, but it does not work for matrices like

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

a good choice for the shift $\mu^{(i)}$ is the Eigenvalue of a 2×2 slice of the whole matrix:

$$E^{(i)} := \begin{pmatrix} a_{n-1}^{(i)} & b_n^{(i)} \\ b_n^{(i)} & a_n^{(i)} \end{pmatrix}$$

that is closer to $a_n^{(i)}$.

If some Eigenvalues of A have the same modulus, $b_j^{(i+1)}$ may converge to 0 faster than $b_n^{(i+1)}$, in which case we can split the matrix into two sub-matrices, and continue finding Eigenvalues for the two smaller sub-matrices.

Another possible choice of shift parameter is Rayleigh's quotient.

With shift, relations (2) still holds, but relation (3) becomes

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

Therefore, if the shift parameters are good approximation to Eigenvalues, columns of $\hat{Q}^{(k)}$ quickly converges to Eigenvectors of A.

7.4 Non-symmetric matrices

QR can also be performed on upper Hessenberg matrix H (so even A is non-symmetric, its Eigenvalue can be found using QR). The shifts are more complicated due to possible complex-valued Eigenvalues, but QR is still a very robust choice. (More details can be found in *The algebraic eigenvalue problem* by Wilkinson, J. H. and *Handbook for automatic computation, Vol. 2: Linear Algebra* by Wilkinson, J.H. and C. Reinsch)

In summary, QR is powerful in finding Eigenvalues, but if the associated Eigenvectors are required, the inverse power method may be more suitable.

8 Krylov Methods

There are many implementations based on the idea of using Krylov subspace to solve linear systems, and most of them can be found in the LAPACK(Linear Algebra PACKage). The one we learnt in lectures is designed by Arnoldi. Other methods include conjugate gradient(CG), conjugate Residual(CR), general CR(GCR), biconjugate gradient(BiCG), Lanczo-type product methods(LTPM), ORTHOMIN.

Definition 1 (Krylov subspace)**.** Given non-singular matrix $A \in \mathbb{C}^{m\times m}$ and $\mathbf{0} \neq \boldsymbol{y} \in \mathbb{C}^m$, the nth Krylov subspace is

$$\mathcal{K}_n := \text{span}(\boldsymbol{y}, A\boldsymbol{y}, \cdots, A^{n-1}\boldsymbol{y})$$

There are few important results about Krylov subspace:

- $\mathcal{K}_1 \subseteq \mathcal{K}_2 \subseteq \mathcal{K}_3 \subseteq \cdots$
- there is a positive integer $\nu := \nu(\boldsymbol{y}, A)$ s.t.

$$\dim(\mathcal{K}_n) = \begin{cases} n & \text{if } n \leq \nu \\ \nu & \text{if } n \geq \nu \end{cases}$$

  and this integer is given by

$$\nu(\boldsymbol{y}, A) = \min\left\{n \,:\, A^{-1}\boldsymbol{y} \in \mathcal{K}_n\right\}$$

Solvers using Krylov subspace basically aims to find a good polynomial $p(x)$ s.t. $p(A)v$ is almost $A^{-1}b$, where v is usually chosen directly as b, or residual $r_0 = b - Ax_0$ where x_0 is some approximation of the solution. To illustrate why this is possible, suppose we choose $v = b$, if A is invertible, the set $\{b, Ab, A^2b, \cdots, A^nb\}$ with $n+1$ vectors must be linearly dependent, and so

$$\alpha_0 b + \alpha_1 Ab + \cdots + \alpha_n A^n b = 0$$

for some coefficients α_i. If k is the smallest integer s.t. $\alpha_k = 0$, then $A^{-1}b = -\frac{1}{\alpha_k}\left(\alpha_{k+1}b + \cdots + \alpha_n A^{n-k-1}b\right)$ This is called the *weak Cayley-Hamilton Theorem.*

8.1 Arnoldi iteraitons

To construct Krylov subspace, one just need to repeatedly multiply A to v and orthogonalise this set of vectors $\{v, Av, A^2v, \cdots\}$ to get basis. This can be done by performing Gram-Schmidt at each step: unify v getting v_1, find Av_1 and remove the component of v_1 to get v_2, repeat. This is the Arnoldi iteration. If this algorithm is written using matrix,

$$A\hat{Q}_n = \hat{Q}_{n+1}\tilde{H}_n$$

where $\tilde{H}_n \in \mathbb{C}^{(n+1)\times n}$ is upper Hessenberg and $\hat{Q}_n \in \mathbb{C}^{m\times n}$ has orthonormal columns. Because if we take out the nth column of this equation,

$$Aq_n = h_{1n}q_1 + \cdots + h_{nn}q_n + hn + 1, nq_{j+1}$$

where RHS is linear combination of columns of $\hat{Q}_{n+1}$.

If we define nth Krylov matrix to be

$$K_n = \begin{pmatrix} | & | & & | \\ b & Ab & \cdots & A^{n-1}b \\ | & | & & | \end{pmatrix}$$

then the reduced QR factorisation is $K_n = Q_n R_n$ where Q_n will be the same as $\hat{Q}_n$ obtained during Arnoldi iterations. But due to possible overflow while computing K_n, and failure of QR factorisation when matrix is ill-conditioned, directly doing QR factorisation to Krylov matrix is numerical unstable.

Arnoldi iterations can be viewed as projections. As proved in the lectures, $H_n = \hat{Q}_n^* A \hat{Q}_n$, and so H_n is representation in basis $\{q_1, \cdots, q_n\}$ of orthogonal projection of A onto the nth Krylov subspace $\mathcal{K}_n$. This projection method described is called *Rayleigh-Ritz procedure*.

8.2 GMRES

With the basis obtained from Arnoldi iteration, there are many ways to utilise this to solve linear systems. The algorithm GMRES we learnt aims to minimises the norm of residual $r = Ax - b$. At each iteration, it finds the vector in $x^{(n)} \in \mathcal{K}_n$ that minimises $\|Ax^{(n)} - b\|$. With the orthonormal basis obtained from Arnoldi iterations, we know $x^{(n)} = \hat{Q}_n y$ for some $y \in \mathbb{C}^n$. Then the tricks mentioned in lecture notes find an equivalent problem to the minimisation problem above:

$$\min_{y \in \mathbb{R}^n} \|\tilde{H}_n y - \|b\| e_1\|, \qquad \text{where } e_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{R}^n$$

GMRES is a special case of Petrov-Galerkin-Krylov(PGK) Methods.