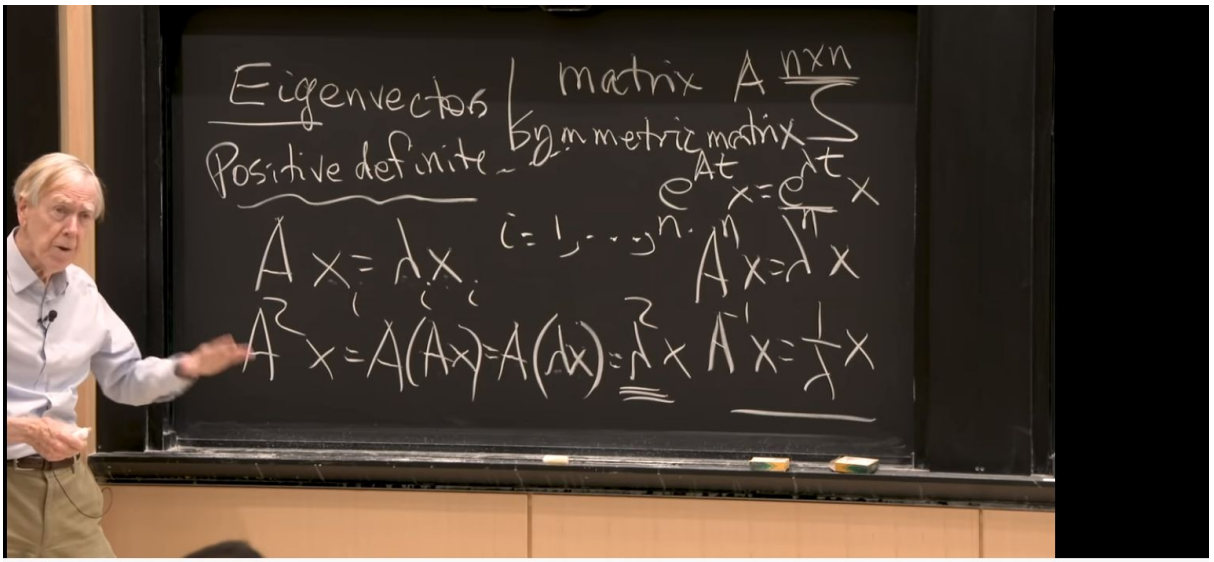
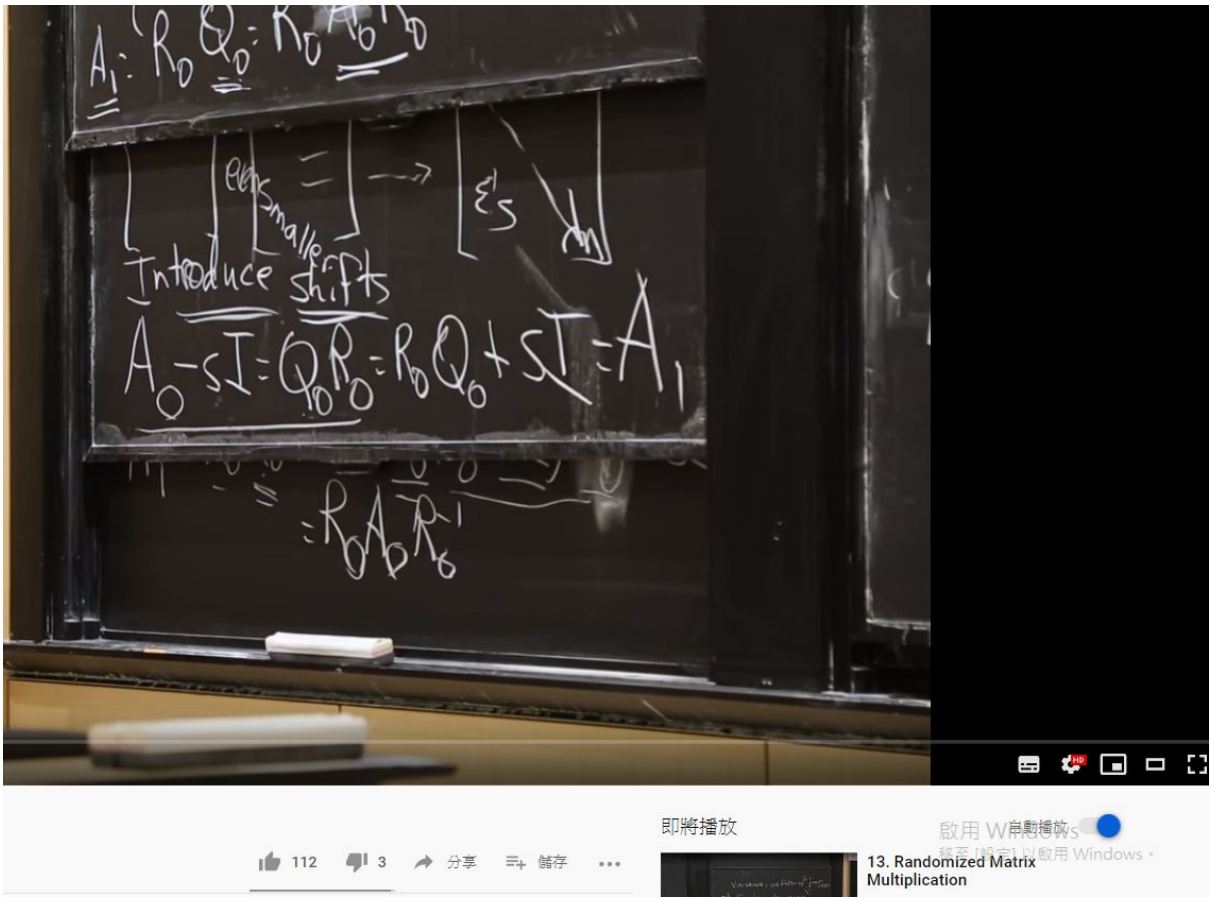
PMS final prepare

About Power Method, QR decomposition

- Week 13: 2019.12.3(二)
  - 課程進度
    - Ch.9. Approximating Eigenvalues
      - 簡介 9.1-9.6 (PDF)
      - 9.3. The Power Method (PDF)
      - 9.5. The QR Algorithm (PDF)
      - 9.6. Singular Value Decomposition
- Week 14: 2019.12.10(二)
  - 課程進度
    - Ch.9. Approximating Eigenvalues
      - 9.1. Linear Algebra and Eigenvalues
      - 9.2. Orthogonal Matrices and Similarity Transformations
      - 9.3. The Power Method (PDF)
      - 9.4. Householder's Method
      - 9.5. The QR Algorithm (PDF)
- Week 14: 2019.12.13(五)
  - 隨堂檢測.7
- Week 15: 2019.12.17(二)
  - 課程進度
    - Ch.9. Approximating Eigenvalues
      - 9.5. The QR Algorithm (PDF)
      - 9.4. Householder's Method
- Week 16: 2019.12.24(二)
  - 課程進度
    - Ch.9. Approximating Eigenvalues
      - 9.4. Householder's Method
      - 9.5. The QR Algorithm (PDF)
- Week 16: 2019.12.27(五)
  - 隨堂檢測.8: Q.1-Q.3
    - the Gram-Schmidt process
    - the Householder transformation
- Week 17: 2019.12.31(二)
  - 隨堂檢測.8: Q.4-Q.7
    - the Jacobi's method like (use the Gram-Schmidt process to replace the rotation matrix vs. sec.9.5 #15, p.623)
    - the QR method (for the symmetric tridiagonal matrix)
    - the single shift QR method (for the symmetric tridiagonal matrix)
    - the variant single shift QR method (for the symmetric tridiagonal matrix)
- Week 18: 2020.1.7(二) 期末考





https://www.youtube.com/watch?v=d32WV1rKoVk

https://www.youtube.com/watch?v=k095NdrHxY4&list=PLUl4u3cNGP63oMNUHXqIUcrkS2PivhN3k&index=7&t=0s
https://www.youtube.com/watch?v=dkT8yuI2d50&list=PLO7mwuD-OlqlKfzBjIGL-u4ci-KfHO68L

https://www.youtube.com/watch?v=1kw8bpA9QmQ

//power
http://math.ntnu.edu.tw/~min/matrix_computation/power_method.pdf
http://math.ntnu.edu.tw/~min/matrixcontent.html
https://www.youtube.com/watch?v=g3-oinWys8I&list=PLO7mwuD-OlqlKfzBjIGL-u4ci-KfHO68L&index=17

http://math.ntnu.edu.tw/~min/Hands_on_power_Lanczos_all.html

Power Method

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

Inverse Shifted Power Method (A-Mu0*I) -1

$$b_{k+1} = \frac{(A - \mu I)^{-1} b_k}{C_k},$$

where C_k are some constants usually chosen as $C_k = \|(A - \mu I)^{-1} b_k\|$.

Rayleigh Quotient iteration(Variant Inverse Shifted Power Method) cubic/O(n^3)step for recomputing (A-Mu*I)-1

$$b_{i+1} = \frac{(A - \mu_i I)^{-1} b_i}{\|(A - \mu_i I)^{-1} b_i\|},$$

where I is the identity matrix,

$$\mu_{i+1} = \frac{b_{i+1}^* A b_{i+1}}{b_{i+1}^* b_{i+1}}.$$

Speed of convergence [edit]

Let us analyze the rate of convergence of the method.

The power method is known to converge linearly to the limit, more precisely:

$$\text{Distance}(b^{\text{ideal}}, b^k_{\text{Power Method}}) = O\left(\left|\frac{\lambda_{\text{subdominant}}}{\lambda_{\text{dominant}}}\right|^k\right),$$

hence for the inverse iteration method similar result sounds as:

$$\text{Distance}(b^{\text{ideal}}, b^k_{\text{Inverse iteration}}) = O\left(\left|\frac{\mu - \lambda_{\text{closest to } \mu}}{\mu - \lambda_{\text{second closest to } \mu}}\right|^k\right).$$

Tridiagonalization [edit]

Main article: Tridiagonal matrix

This procedure is taken from the book: Numerical Analysis, Burden and Faires, 8th Edition. In the firs

$$\alpha = -\operatorname{sgn}(a_{21})\sqrt{\sum_{j=2}^{n} a_{j1}^2};$$

$$r = \sqrt{\frac{1}{2}\left(\alpha^2 - a_{21}\alpha\right)};$$

From α and r, construct vector v:

$$v^{(1)} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix},$$

where $v_1 = 0$, $v_2 = \frac{a_{21}-\alpha}{2r}$, and

$v_k = \frac{a_{k1}}{2r}$ for each $k = 3, 4 \ldots n$

Then compute:

$$P^1 = I - 2v^{(1)}\left(v^{(1)}\right)^{\mathsf{T}}$$

$$A^{(2)} = P^1 A P^1$$

Having found P^1 and computed $A^{(2)}$ the process is repeated for $k = 2, 3, \ldots, n-2$ as follows:

$$\alpha = -\operatorname{sgn}\left(a_{k+1,k}^k\right)\sqrt{\sum_{j=k+1}^{n}\left(a_{jk}^k\right)^2}$$

$$r = \sqrt{\frac{1}{2}\left(\alpha^2 - a_{k+1,k}^k\alpha\right)}$$

$$v_1^k = v_2^k = \cdots = v_k^k = 0$$

$$v_{k+1}^k = \frac{a_{k+1,k}^k - \alpha}{2r}$$

$$v_j^k = \frac{a_{jk}^k}{2r} \text{ for } j = k+2,\ k+3,\ \ldots,\ n$$

$$P^k = I - 2v^{(k)}\left(v^{(k)}\right)^{\mathsf{T}}$$

$$A^{(k+1)} = P^k A^{(k)} P^k$$

Continuing in this manner, the tridiagonal and symmetric matrix is formed.

Tridiagonization

- Reduction to Hessenberg form

Take

$$A = \begin{pmatrix} \alpha_{11} & a_{12}^* \\ a_{21} & A_{22} \end{pmatrix}.$$

Let $\hat{H}_1$ be a Householder transformation such that

$$\hat{H}_1 a_{21} = v_1 e_1.$$

Set $H_1 = diag(1, \hat{H}_1)$. Then

$$H_1 A H_1 = \begin{pmatrix} \alpha_{11} & a_{12}^* \hat{H}_1 \\ \hat{H}_1 a_{21} & \hat{H}_1 A_{22} \hat{H}_1 \end{pmatrix} = \begin{pmatrix} \alpha_{11} & a_{12}^* \hat{H}_1 \\ v_1 e_1 & \hat{H}_1 A_{22} \hat{H}_1 \end{pmatrix}$$

For the general step, suppose $H_1, \cdots, H_{k-1}$ are Householder transformation such that

$$H_{k-1} \cdots H_1 A H_1 \cdots H_{k-1} = \begin{pmatrix} A_{11} & a_{1,k} & A_{1,k+1} \\ 0 & \alpha_{kk} & a_{k,k+1}^* \\ 0 & a_{k+1,k} & A_{k+1,k+1} \end{pmatrix},$$

$$\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix} \xrightarrow{H_1} \begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix}$$

$$\xrightarrow{H_2} \begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix}$$

$$\xrightarrow{H_3} \begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{pmatrix}$$

Method	Applies to	Produces	Cost without similarity matrix	Cost with similarity matrix	Description
Householder transformations	General	Hessenberg	$2n^3/3 + O(n^2)$[9](p474)	$4n^3/3 + O(n^2)$[9](p474)	Reflect each column through a subspace to zero out its lower entries.
Givens rotations	General	Hessenberg	$4n^3/3 + O(n^2)$[9](p470)		Apply planar rotations to zero out individual entries. Rotations are ordered so that later ones do not cause zero entries to become non-zero again.
Arnoldi iteration	General	Hessenberg			Perform Gram–Schmidt orthogonalization on Krylov subspaces.
Lanczos algorithm	Hermitian	Tridiagonal			Arnoldi iteration for hermitian matrices, with shortcuts.

Jacobi for Eigen

A Givens rotation is represented by a matrix of the form

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

where $c = \cos\theta$ and $s = \sin\theta$ appear at the intersections i

$$\begin{aligned} g_{kk} &= 1 \qquad \text{for } k \neq i,\ j \\ g_{kk} &= c \qquad \text{for } k = i,\ j \\ g_{ji} &= -g_{ij} = -s \end{aligned}$$

Let S be a symmetric matrix, and $G = G(i,j,\theta)$ be a Givens rotation matrix. Then:

$$S' = GSG^{\mathsf{T}}$$

is symmetric and similar to S.

Furthermore, S' has entries:

$$\begin{aligned} S'_{ii} &= c^2 S_{ii} - 2sc S_{ij} + s^2 S_{jj} \\ S'_{jj} &= s^2 S_{ii} + 2sc S_{ij} + c^2 S_{jj} \\ S'_{ij} &= S'_{ji} = (c^2 - s^2) S_{ij} + sc (S_{ii} - S_{jj}) \\ S'_{ik} &= S'_{ki} = c S_{ik} - s S_{jk} && k \neq i,j \\ S'_{jk} &= S'_{kj} = s S_{ik} + c S_{jk} && k \neq i,j \\ S'_{kl} &= S_{kl} && k,l \neq i,j \end{aligned}$$

where $s = \sin(\theta)$ and $c = \cos(\theta)$.

Since G is orthogonal, S and S' have the same Frobenius norm $||\cdot||_F$ (the square-root sum of squares of all components), however we can choose θ such that $S'_{ij} = 0$, in which case S' i

$$S'_{ij} = \cos(2\theta) S_{ij} + \tfrac{1}{2}\sin(2\theta)(S_{ii} - S_{jj})$$

Set this equal to 0, and rearrange:

$$\tan(2\theta) = \frac{2S_{ij}}{S_{jj} - S_{ii}}$$

if $S_{jj} = S_{ii}$

$$\theta = \frac{\pi}{4}$$

In order to optimize this effect, S_{ij} should be the off-diagonal element with the largest absolute value, called the *pivot*.

The Jacobi eigenvalue method repeatedly performs rotations until the matrix becomes almost diagonal. Then the elements in the diagonal are approximations of the (real) eigenvalues of S.

In summary, here are the steps in each iteration of the Jacobi algorithm:

- Find the off-diagonal element a_{ij} $(i \neq j)$ of the greatest absolute value, the *pivot*, and find $w = (a_{jj} - a_{ii})/2a_{ij}$.
- Solve the quadratic equation $t^2 + 2wt - 1 = 0$ for $t = \tan\theta$.
- Obtain $c = 1/\sqrt{1+t^2}$ and $s = t/\sqrt{1+t^2}$.
- Update all elements in the ith and jth rows and columns.

QR

form $A_{k+1} = R_k Q_k$. Note that

$$A_{k+1} = R_k Q_k = Q_k^{-1} Q_k R_k Q_k = Q_k^{-1} A_k Q_k = Q_k^{\mathsf{T}} A_k Q_k,$$

so all the A_k are similar and hence they have the same eigenvalues. The algorithm is numerically stable because it proceeds by *orthogonal* similarity transforms.

Using the Gram–Schmidt process

Advantages and disadvantages[edit]

The Gram-Schmidt process is inherently **numerically unstable**. While the application of the projections has an appealing geometric analogy to orthogonalization, the orthogonalization itself is prone to numerical error. A significant advantage, however, is the **ease of implementation**, which makes this a useful algorithm to use for prototyping if a pre-built linear algebra library is unavailable.

Householder Reflection

Advantages and disadvantages[edit]

The use of Householder transformations is inherently **the most simple of the numerically stable QR decomposition** algorithms due to the use of reflections as the mechanism for producing zeroes in the *R* matrix. However, the Householder reflection algorithm is **bandwidth-heavy and not parallelizable,** as every reflection that produces a new zero element changes the entirety of both *Q* and *R* matrices.

Given Rotation

Advantages and disadvantages[edit]

The QR decomposition via Givens rotations is the most **involved to implement,** as the ordering of the rows required to fully exploit the algorithm is not trivial to determine. However, it has a significant advantage in that each new zero element affects only the row with the element to be zeroed (i) and a row above (j). This makes the Givens rotation algorithm more **bandwidth-efficient and parallelizable** than the Householder reflection technique.

Advantages and disadvantages [edit]

The use of Householder transformations is inherently the most simple of the numerically stable QR decomposition algorithms due to the use of reflections as the mechanism for producing zeroes in the *R* matrix. However, the Householder reflection algorithm is bandwidth heavy and not parallelizable, as every reflection that produces a new zero element changes the entirety of both *Q* and *R* matrices.

Advantages and disadvantages [edit]

The QR decomposition via Givens rotations is the most involved to implement, as the ordering of the rows required to fully exploit the algorithm is not trivial to determine. However, it has a significant advantage in that each new zero element a_{ij} affects only the row with the element to be zeroed (i) and a row above (j). This makes the Givens rotation algorithm more bandwidth efficient and parallelisable than the Householder reflection technique.

Iterative algorithms [edit]

Iterative algorithms solve the eigenvalue problem by producing sequences that converge to the eigenvalues. Some algorithms also produce sequences of vectors that converge to the eigenvectors. Most commonly, the eigenvalue sequences are expressed as s
of similar matrices which converge to a triangular or diagonal form, allowing the eigenvalues to be read easily. The eigenvector sequences are expressed as the corresponding similarity matrices.

Method	Applies to	Produces	Cost per step	Convergence	Description
Power iteration	general	eigenpair with largest value	$O(n^2)$	linear	Repeatedly applies the matrix to an arbitrary starting vector and renormalizes.
Inverse iteration	general	eigenpair with value closest to μ		linear	Power iteration for $(A-\mu I)^{-1}$
Rayleigh quotient iteration	hermitian	any eigenpair		cubic	Power iteration for $(A-\mu_i I)^{-1}$, where μ_i for each iteration is the Rayleigh quotient of the previous iteration.
Preconditioned inverse iteration[11] or LOBPCG algorithm	positive-definite real symmetric	eigenpair with value closest to μ			Inverse iteration using a preconditioner (an approximate inverse to A).
Bisection method	real symmetric tridiagonal	any eigenvalue		linear	Uses the bisection method to find roots of the characteristic polynomial, supported by the Sturm sequence.
Laguerre iteration	real symmetric tridiagonal	any eigenvalue		cubic[12]	Uses Laguerre's method to find roots of the characteristic polynomial, supported by the Sturm sequence.
QR algorithm	Hessenberg	all eigenvalues	$O(n^2)$	cubic	Factors $A = QR$, where Q is orthogonal and R is triangular, then applies the next iteration to RQ.
		all eigenpairs	$6n^3 + O(n^2)$		
Jacobi eigenvalue algorithm	real symmetric	all eigenvalues	$O(n^3)$	quadratic	Uses Givens rotations to attempt clearing all off-diagonal entries. This fails, but strengthens the diagonal.
Divide-and-conquer	Hermitian Tridiagonal	all eigenvalues	$O(n^2)$		Divides the matrix into submatrices that are diagonalized then recombined.
		all eigenpairs	$(4/3)n^3 + O(n^2)$		
Homotopy method	real symmetric tridiagonal	all eigenpairs	$O(n^2)$[13]		Constructs a computable homotopy path from a diagonal eigenvalue problem.
Folded spectrum method	real symmetric	eigenpair with value closest to μ			Preconditioned inverse iteration applied to $(A-\mu I)^2$
MRRR algorithm[14]	real symmetric tridiagonal	some or all eigenpairs	$O(n^2)$		"Multiple relatively robust representations" – performs inverse iteration on a LDL^T decomposition of the shifted matrix.

To accelerate this convergence, a shifting technique is employed similar to that used with the Inverse Power method in Section 9.3. A constant σ is selected close to an eigenvalue of A. This modifies the factorization in Eq. (9.16) to choosing $Q^{(i)}$ and $R^{(i)}$ so that

$$A^{(i)} - \sigma I = Q^{(i)}R^{(i)}, \tag{9.18}$$

and, correspondingly, the matrix $A^{(i+1)}$ is defined to be

$$A^{(i+1)} = R^{(i)}Q^{(i)} + \sigma I. \tag{9.19}$$

With this modification, the rate of convergence of $b_{j+1}^{(i+1)}$ to 0 depends on the ratio $|(\lambda_{j+1} - \sigma)/(\lambda_j - \sigma)|$. This can result in a significant improvement over the original rate of convergence of $a_j^{(i+1)}$ to λ_j if σ is close to λ_{j+1} but not close to λ_j.

We change σ at each step so that when A has eigenvalues of distinct modulus, $b_n^{(i+1)}$ converges to 0 faster than $b_j^{(i+1)}$ for any integer j less than n. When $b_n^{(i+1)}$ is sufficiently small, we assume that $\lambda_n \approx a_n^{(i+1)}$, delete the nth row and column of the matrix, and proceed in the same manner to find an approximation to λ_{n-1}. The process is continued until an approximation has been determined for each eigenvalue.

The shifting technique chooses, at the ith step, the shifting constant σ_i, where σ_i is the eigenvalue of the matrix

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

that is closest to $a_n^{(i)}$. This shift translates the eigenvalues of A by a factor σ_i. With this shifting technique, the convergence is usually cubic. (See [WR], p. 270.) The method accumulates these shifts until $b_n^{(i+1)} \approx 0$ and then adds the shifts to $a_n^{(i+1)}$ to approximate the eigenvalue λ_n.

If A has eigenvalues of the same modulus, $b_j^{(i+1)}$ may tend to 0 for some $j \neq n$ at a faster rate than $b_n^{(i+1)}$. In this case, the matrix-splitting technique described in (9.14) can be employed to reduce the problem to one involving a pair of matrices of reduced order.

the matrix, and a shifting procedure is used similar to that in the QR method. However, the shifting is somewhat more complicated for nonsymmetric matrices since complex eigenvalues with the same modulus can occur. The shifting process modifies the calculations in Eqs. (9.20), (9.21), and (9.22) to obtain the double QR method

$$H^{(1)} - \sigma_1 I = Q^{(1)}R^{(1)}, \qquad H^{(2)} = R^{(1)}Q^{(1)} + \sigma_1 I,$$
$$H^{(2)} - \sigma_2 I = Q^{(2)}R^{(2)}, \qquad H^{(3)} = R^{(2)}Q^{(2)} + \sigma_2 I,$$

where σ_1 and σ_2 are complex conjugates and $H^{(1)}, H^{(2)}, \ldots$ are real upper-Hessenberg matrices.

§9.2.2. Disadvantages of Jacobi's method

(i) When the process is truncated it may introduce errors which seriously affect the solutions.

(ii) The elements that are reduced to zero may not remain so during subsequent transformations.

(iii) This process is very slow and the number of iterations increase if the matrix is large.

(iv) The minimum number of operations to make a matrix into a diagonal form is $\frac{n(n-1)}{2}$.

Although the method has many disadvantages, the advantage of this method is that it gives all the eigen values and eigen vectors of a real symmetric matrix at a time.

Advantages of the Jacobi method. It is simple and numerically robust. Each iteration is quite fast. Notice that (14) can be done very efficiently because of the special nature of the matrix M.

Disadvantages. It might require many iterations. More important, we are forced to estimate all eigenvalues of Q, whereas for our factor decomposition we only want the r largest.

Parallel Jacobi

The primary advantage of the Jacobi method over the symmetric QR algorithm is its parallelism. As each Jacobi update consists of a row rotation that affects only rows p and q, and a column rotation that effects only columns p and q, up to $n/2$ Jacobi updates can be performed in parallel. Therefore, a sweep can be efficiently implemented by performing $n-1$ series of $n/2$ parallel updates in which each row i is paired with a different row j, for $i \neq j$.

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

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

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

The Modified QR Method:

Step 1: Choose s_1 be an approximation to the jth eigenvalue of the $n \times n$ matrix A. Let$A_1 = A - s_1 I = Q_1 R_1$ be a QR factorization of $A - s_1 I$; create $A_2 = R_1 Q_1 + s_1 I$

Step 2: Choose s_2 to be be an approximation to the jth eigenvalue of A. Let $A_2 - s_2 I = Q_2 R_2$ be a QR factorization of $A_2 - s_2 I$; create $A_3 = R_2 Q_2 + s_2 I$

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

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

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

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

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

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

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

3 Pros/cons of the power method

- Our analysis shows that the power method can converge very slowly if $|\lambda_2/\lambda_1|$ is close to 1. And if two eigenvalues have *equal* magnitude, the method may not converge *at all*.
- Also, it only gives us x_1. What if we want x_2, or in general a *few* of the biggest-$|\lambda|$ eigenvectors?

Still, the power method is the **starting point for many more sophisticated methods,** including the Arnoldi method (which gives a few of the biggest eigenvectors) or the QR algorithm which gives *all* of the eigenvectors.

And the power method *by itself* can still be a pretty good method if we know that one eigenvalue is bigger than all of the others, e.g. for Markov matrices. And because it is so simple, the power method is easy to apply in lots of different cases, especially since:

- The power method only requires you to supply a "black box" that **multiplies matrix $\times$ vector** This is a *huge* advantage for problems where the matrix is *mostly zeros* (or has some other special structure), in which you can multiply **matrix $\times$ vector much more quickly than for a generic matrix.**

In homework, you will look at how Markov matrices relate to the Google PageRank. Google actually runs this algorithm on a *huge* Markov matrix where rows/cols are *web pages*: the matrix is *over a billion by billion entries*. But since most web pages only link to a few other pages, the matrix is mostly zeros, and you can multiply it by a random vector in a few billion operation, rather than a billion^2 operations. (They don't even store the whole matrix: you only store the nonzero entries of such a sparse matrix.)