

Notes for Ma221 Lecture 10, Mar 29, 2022

Chapter 5 on symmetric eigenvalue problems and the SVD

Goals:

- Perturbation Theory
- Algorithms for the symmetric eigenproblem and SVD

Perturbation Theory

The theorems we present are useful not just for perturbation theory, but understanding why algorithms work. Everything from chapter 4 applies to symmetric matrices, but much more can be shown.

We consider only real, symmetric  $A = A^T$ . The case of complex Hermitian ( $A = A^H$ ) is similar. Then we know that we can write  $A = Q \Lambda Q^T$  where  $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$  with real eigenvalues, and  $Q = [q_1, \dots, q_n]$  consists of real orthonormal eigenvectors. We'll assume  $\lambda_1 \geq \dots \geq \lambda_n$ .

Note that the complex symmetric case is totally different:

$A = \begin{bmatrix} 1 & i \\ i & -1 \end{bmatrix}$  has 2 eigenvalues at 0, and is not diagonalizable.

Most of what we say will also be true for the SVD of an arbitrary matrix:

Recall that the SVD of  $A$  is simply related to the eigendecomposition of the symmetric matrix (see Thm 3.3 part 4)

$$B = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$$

with eigenvalues of  $B = \pm$  singular values of  $A$  (plus some zeros if  $A$  is

rectangular). So a small change from  $A$  to  $A+E$  is also a small symmetric change to  $B$ .

So if we show this doesn't change the eigenvalues of  $B$  very much (in fact by no more than  $\text{norm}(E)$ ), then this means the singular values of  $A$  can change just as little.

Def: Rayleigh quotient  $\rho(u, A) = u^T A u / u^T u$ .

Properties: if  $A q = \lambda q$ , then  $\rho(q, A) = \lambda$

if  $u = \sum_i b_i q_i = Q b$  where  $b = [b_1, \dots, b_n]$ , then

$$\begin{aligned} \rho(u, A) &= (Q b)^T A (Q b) / (Q b)^T (Q b) \\ &= b^T Q^T A Q b / (b^T Q^T Q b) \\ &= b^T \Lambda b / (b^T b) \\ &= \sum_i \lambda_i b_i^2 / \sum_i b_i^2 \\ &= \text{convex combination of } \{\lambda_1, \dots, \lambda_n\} \end{aligned}$$

so  $\lambda_1 \geq \rho(u, A) \geq \lambda_n$  for all  $u$

and  $\lambda_1 = \max_u \rho(u, A)$

$\lambda_n = \min_u \rho(u, A)$

In fact all the eigenvalues can be written using the Rayleigh quotient:

Thm (Courant-Fischer Minimax Thm): Let  $R^j$  denote a  $j$ -dimensional subspace of  $n$ -dimensional space, and  $S^{(n-j+1)}$  denote an  $n-j+1$  dimensional subspace. Then

$$\begin{aligned} & \max_{\{R^j\}} \min_{\{0 \neq r \text{ in } R^j\}} \rho(r,A) \\ &= \lambda_j \\ &= \min_{\{S^{(n-j+1)}\}} \max_{\{0 \neq s \text{ in } S^{(n-j+1)}\}} \rho(s,A) \end{aligned}$$

The max over  $R^j$  is attained when  $R^j = \text{span}(q_1, \dots, q_j)$ , and the min over  $r$  is attained for  $r = q_j$ .

The min over  $S^{(n-j+1)}$  is attained for  $S^{(n-j+1)} = \text{span}(q_j, q_{j+1}, \dots, q_n)$  and the max over  $s$  is attained for  $s = q_j$  too.

Proof:

For any pair  $R^j$  and  $S^{(n-j+1)}$ , since their dimensions add up to  $n+1$ , they must intersect in some nonzero vector  $x_{RS}$ . So

$$\begin{aligned} \min_{\{0 \neq r \text{ in } R^j\}} \rho(r,A) &\leq \rho(x_{RS},A) \\ &\leq \max_{\{0 \neq s \text{ in } S^{(n-j+1)}\}} \rho(s,A) \end{aligned}$$

Let  $R'$  maximize  $\min_{\{0 \neq r \text{ in } R^j\}} \rho(r,A)$  and

Let  $S'$  minimize  $\max_{\{0 \neq s \text{ in } S^{(n-j+1)}\}} \rho(s,A)$  so

$$\begin{aligned} & \max_{\{R^j\}} \min_{\{0 \neq r \text{ in } R^j\}} \rho(r,A) \\ &= \min_{\{0 \neq r \text{ in } R'^j\}} \rho(r,A) \\ &\leq \rho(x_{R'S'}) \\ &\leq \max_{\{0 \neq s \text{ in } S'^{(n-j+1)}\}} \rho(s,A) \\ &= \min_{\{S^{(n-j+1)}\}} \max_{\{0 \neq s \text{ in } S^{(n-j+1)}\}} \rho(s,A) \end{aligned}$$

But if we choose  $R^j = \text{span}(q_1, \dots, q_j)$  and  $r = q_j$ , we get

$$\min_{\{0 \neq r \text{ in } R^j\}} \rho(r,A) = \rho(q_j, A) = \lambda_j$$

and if we choose  $S_{(n-j+1)} = \text{span}(q_j, \dots, q_n)$  and  $s = q_j$ , we get

$$\max_{\{0 \neq s \text{ in } S^{(n-j+1)}\}} \rho(s,A) = \rho(q_j, A) = \lambda_j$$

so the above inequalities are bounded below and above by  $\lambda_j$ , and so must all equal  $\lambda_j$ .

(We pause the recorded lecture here.)

Weyl's Theorem: if  $A$  is symmetric, with eigenvalues  $\lambda_1 \geq \dots \geq \lambda_n$  and if  $A+E$  is symmetric, with eigenvalues  $\mu_1 \geq \dots \geq \mu_n$ , then  $|\lambda_j - \mu_j| \leq \|E\|_2$  for all  $j$

Corollary: If  $A$  general, with singular values  $\sigma_1 \geq \dots \geq \sigma_n$ , and  $A+E$  has singular values  $\tau_1 \geq \dots \geq \tau_n$ , then  $|\sigma_j - \tau_j| \leq \|E\|_2$  for all  $j$ .

Proof of Weyl:

$$\begin{aligned} \mu_j &= \min_{\{S^{(n-j+1)}\}} \max_{\{0 \neq s \text{ in } S^{(n-j+1)}\}} \rho(s, A+E) \\ &= \min_{\{S^{(n-j+1)}\}} \max_{\{0 \neq s \text{ in } S^{(n-j+1)}\}} s^T (A+E) s / s^T s \\ &= \min_{\{S^{(n-j+1)}\}} \max_{\{0 \neq s \text{ in } S^{(n-j+1)}\}} [ s^T A s / s^T s + s^T E s / s^T s ] \end{aligned}$$

$$\leq \min_{\{S^{(n-j+1)}\}} \max_{\{0 \neq s \in S^{(n-j+1)}\}} [s^T A s / s^T s + \|E\|_2] \\ = \lambda_j + \|E\|_2$$

Swapping roles of  $\mu_j$  and  $\lambda_j$ , we get  $\lambda_j \leq \mu_j + \|E\|_2$ , or what we want.

Def:  $\text{Inertia}(A) = (\#\text{negative evals}(A), \#\text{zero evals}(A), \#\text{positive evals}(A))$

Sylvester's Theorem: If  $A$  is symmetric and  $X$  nonsingular, then  $\text{Inertia}(A) = \text{Inertia}(X^T A X)$

Fact: Suppose we do  $A = L^* D^* L^T$  (Gaussian elimination with symmetric or no pivoting). Then  $A$  and  $D$  have the same Inertia, namely  $(\#D(i,i) < 0, \#D(i,i) = 0, \#D(i,i) > 0)$ , which is easy to compute. Now suppose we do  $A - x^* I = L^* D^* L^T$ ; then

$$\#D(i,i) < 0 = \#\text{eigenvalues of } A - x^* I < 0 = \#\text{eigenvalues of } A < x$$

Similarly suppose we do  $A - y^* I = L^* D^* L^T$  for some  $y > x$ , then we can similarly compute  $\#\text{eigenvalues of } A \leq y$ . Then

$$\#\text{eigenvalues in } [x, y] = (\#\text{evals} \leq y) - (\#\text{evals} < x)$$

so we can count the  $\#$  eigenvalues in any interval  $[x, y]$ .

Now suppose we count  $\# \text{evals} < (x+y)/2$ ; we can get the number number of evals in each half of the interval. By repeatedly bisecting the interval, we can compute any subset of the eigenvalues as accurately as we like. We say more on how to do this cheaply later.

Proof of Sylvester's Theorem:

Suppose  $\#\text{evals of } A < 0$  is  $m$ , and  $\#\text{evals of } X^T A X < 0$  is  $m'$ , but  $m' < m$ ; let's find a contradiction. Let  $N$  be the  $m$ -dimensional subspace of eigenvectors for the  $m$  negative eigenvalues of  $A$ , so  $x$  in  $N$  means  $x^T A x < 0$ . and let  $P$  be the  $n-m'$  dimensional subspace of nonnegative eigenvalues of  $X^T A X$ , so  $x$  in  $P$  means  $x^T X^T A X x = (X^* x)^T A (X^* x) \geq 0$ , or  $y$  in  $X^* P$  means  $y^T A y \geq 0$ . But  $\text{dimension}(X^* P) = \text{dimension}(P) = n-m'$ , and  $\text{dimension}(N) + \text{dimension}(X^* P) = n-m'+m > n$ , so they intersect, i.e. there is some nonzero  $x$  in both spaces  $N$  and  $X^* P$ , i.e.  $x^T A x < 0$  and  $x^T A x \geq 0$ , a contradiction.

(We pause the recorded lecture here.)

Now a little about eigenvectors:

Thm: Let  $A = Q^* \Lambda^* Q^T$  be the eigendecomposition of  $A$ , with  $\Lambda^* = \text{diag}(\lambda_i)$ , and  $Q = [q_1, \dots, q_n]$ , and  $A+E = Q'^* \Lambda'^* Q'^T$  be the eigendecomposition of  $A+E$ , with  $\Lambda'^* = \text{diag}(\lambda'_i)$ , and  $Q' = [q'_1, \dots, q'_n]$ .

Let  $\theta_i = \text{angle between } q_i \text{ and } q'_i$ . We want to bound  $\theta_i$ .

Let  $\text{gap}(i, A) = \min_{\{j \neq i\}} |\lambda_j - \lambda_i|$ . then

$$|.5 \sin(2\theta_i)| \leq \|E\|_2 / \text{gap}(i, A)$$

Note that when  $\theta_i$  is small,  $.5 \sin(2\theta_i) \sim \theta_i$   
 In other words, if the perturbation  $\|E\|_2$  is small compared to the distance  $\text{gap}(i,A)$  to the nearest other eigenvalue, the change in direction of the eigenvector will be small.

Proof of a weaker result: write eigenvector  $i$  of  $A+E$  as  $q_i + d$  where  $d$  is perpendicular to  $q_i$  (so  $q'_i = (q_i+d)/\|q_i+d\|_2$ ). Then

$$(A+E)(q_i+d) = (\lambda'_i)(q_i+d).$$

Ignoring second order terms we get

$$Aq_i + E q_i + A d = \lambda'_i q_i + \lambda'_i d$$

or

$$(A+E-\lambda'_i I)q_i = (\lambda'_i - A)d$$

or

$$(\lambda_i I + E - \lambda'_i I)q_i = (\lambda'_i - A)d$$

Write  $d = \sum_{j \neq i} d_j q_j$  yielding

$$(\lambda_i I + E - \lambda'_i I)q_i = \sum_{j \neq i} d_j (\lambda'_i - \lambda_j) q_j$$

Taking 2-norms, the norm(LHS)  $\leq 2\|E\|_2$  by Weyl's Theorem

Taking 2-norms, the norm(RHS)  $\geq (\text{gap}(i,A) - \|E\|) \|d\|_2$

So  $2\|E\|_2 / (\text{gap}(i,A) - \|E\|) \geq \|d\|_2 = |\tan(\theta_i)|$ .

See the text for the full proof.

(We pause the recorded lecture here.)

Important fact about the Rayleigh Quotients: The Rayleigh Quotient is a best approximation to an eigenvalue in a certain sense:

Thm: Suppose  $x$  is a unit vector and  $\beta$  a scalar. Then  $A$  has an eigenvalue

$\alpha$  such that  $|\alpha - \beta| \leq \|A^*x - \beta x\|_2$ . Given only  $x$ ,

the choice  $\beta = \rho(x,A)$  minimizes  $\|A^*x - \beta x\|_2$ . So given any

unit vector  $x$ , there is always an eigenvalue within distance

$\|A^*x - \rho(x,A)x\|_2$  of  $\rho(x,A)$ , and  $\rho(x,A)$  minimizes this bound.

Furthermore, let  $\lambda_i$  be the eigenvalue of  $A$  closest to  $\rho(x,A)$ , and

$$\text{gap} = \min_{j \neq i} |\lambda_j - \rho(x,A)|$$

be the distance to the next closest eigenvalue. Then

$$|\lambda_i - \rho(x,A)| \leq \|A^*x - \rho(x,A)x\|_2^2 / \text{gap}$$

i.e. the error in  $\rho(x,A)$  as an approximate eigenvalue

is proportional to the square of the norm of the residual  $A^*x - \rho(x,A)x$ .

Proof of part 1: If  $x$  is a unit vector

$$1 = \|x\| = \|\text{inv}(A - \beta I)(A - \beta I)x\|$$

$$\text{so } 1 \leq \|\text{inv}(A - \beta I)\|_2 \cdot \|A^*x - \beta x\|_2$$

$$\text{or } 1 \leq 1/\min_i |\lambda_i - \beta| \cdot \|A^*x - \beta x\|_2$$

To show that  $\beta = \rho(x,A)$  minimizes  $\|A^*x - \beta x\|_2$ ,

write  $A^*x - \beta x = [A^*x - \rho(x,A)x] + [\rho(x,A)x - \beta x]$

$$= y + z$$

If we show  $z$  is orthogonal to  $y$ , then by the Pythagorean Theorem we are done:

$$\begin{aligned} z^T y &= (\rho(x,A) - \beta) * x^T * (A * x - \rho(x,A) * x) \\ &= (\rho(x,A) - \beta) * (x^T * A * x - \rho(x,A) * x^T * x) \\ &= 0 \end{aligned}$$

Partial Proof of Part 2: We do just the special case of a 2x2 diagonal matrix, which seems very special, but has all the ingredients of the general case.

So we assume  $A = \text{diag}(\lambda_1, \lambda_2)$ , and  $x = [c; s]$  where  $c^2 + s^2 = 1$ , so  $\rho = c^2 * \lambda_1 + s^2 * \lambda_2$ . We'll assume  $c > s$ , so  $\rho$  is closer to  $\lambda_1$  than  $\lambda_2$ , so

$$\begin{aligned} |\lambda_1 - \rho| &= |\lambda_1 - c^2 * \lambda_1 - s^2 * \lambda_2| \\ &= s^2 * |\lambda_1 - \lambda_2| \end{aligned}$$

$$\begin{aligned} \text{gap} &= |\lambda_2 - \rho| = |\lambda_2 - c^2 * \lambda_1 - s^2 * \lambda_2| \\ &= c^2 * |\lambda_1 - \lambda_2| \end{aligned}$$

$$r = A * x - \rho * x = \begin{bmatrix} \lambda_1 * c \\ \lambda_2 * s \end{bmatrix} - (c^2 * \lambda_1 + s^2 * \lambda_2) * \begin{bmatrix} c \\ s \end{bmatrix}$$

$$= \begin{bmatrix} c * s^2 * (\lambda_1 - \lambda_2) \\ s * c^2 * (\lambda_2 - \lambda_1) \end{bmatrix}$$

$$\begin{aligned} \text{and so } \|r\|_2^2 &= s^2 * c^2 * (\lambda_1 - \lambda_2)^2 * (s^2 + c^2) \\ &= s^2 * c^2 * (\lambda_1 - \lambda_2)^2 \end{aligned}$$

$$\begin{aligned} \text{and } \|r\|_2^2 / \text{gap} &= s^2 * |\lambda_1 - \lambda_2| \\ &= |\lambda_1 - \rho| \end{aligned}$$

exactly (whereas the theorem only claims an upper bound, in the general case).

This result will be important later to help show that the QR algorithm from Chapter 4 converges cubically when applied to symmetric matrices (instead of "just" quadratically).

(We pause the recorded lecture here.)

Overview of Algorithms: There are several approaches, depending on what one wants to compute:

- (1) "Usual accuracy": backward stable in the sense that you get the exact eigenvalues and eigenvectors for  $A + E$  where  $\|E\| = O(\text{macheps}) * \|A\|$ 
  - (1.1) Get all the eigenvalues (with or without the corresponding eigenvectors)
  - (1.2) Just get all the eigenvalues in some interval  $[x, y]$  (with or without the corresponding eigenvectors)
  - (1.3) Just get  $\lambda_i$  through  $\lambda_j$  (with or without the corresponding vectors)
    - Ex: get the 10 largest eigenvalues ( $\lambda_1$  through  $\lambda_{10}$ )
- (2) "High accuracy": compute tiny eigenvalues and their eigenvectors more accurately than the "usual" accuracy guarantees.

Ex: If  $A$  well-conditioned, so all singular values large, then error bound  $O(\text{macheps} \cdot \text{norm}(A))$  implies we can compute them with small relative errors. Now consider  $B = D \cdot A$  where  $D$  is diagonal. If some entries of  $D$  are very small, and some  $O(1)$ ,  $B$  will be ill-conditioned, and so have some tiny singular values, and the usual error bound  $O(\text{macheps} \cdot \text{norm}(B))$  implies they will not be accurate. But in this case, there is a tighter perturbation theory, and an algorithm, that will still compute the tiny singular values of  $B$  with about as many correct digits as for  $A$ . For a survey of cases like this when high relative accuracy in tiny eigenvalues and singular values is possible, see the papers "Accurate and efficient expression evaluation and linear algebra," "New fast and accurate Jacobi SVD algorithm," "Computing the SVD with high relative accuracy," and "Jacobi's method is more accurate than QR," linked on the class web page.

(3) Updating - given the eigenvalues and eigenvectors of  $A$ , find them for  $A$  with a "small change", small in rank, eg  $A + x \cdot x^T$ .

All the above options also apply to computing the SVD, with the additional possibility of computing the left and/or the right singular vectors.

Algorithms and their costs:

(1) We begin by reducing  $A$  to  $Q^T \cdot A \cdot Q = T$  where  $Q$  is orthogonal and  $T$  is tridiagonal, costing  $O(n^3)$  flops. This uses the same approach as in chapter 4, where  $Q^T \cdot A \cdot Q$  was upper Hessenberg, since a matrix that is both symmetric and upper Hessenberg must be tridiagonal. This is currently done by LAPACK routine `ssytrd`. All subsequent algorithms operate on  $T$ , which is much cheaper.

We have recently identified an algorithm for tridiagonal reduction that moves only  $O(n^3 / \sqrt{\text{fast\_memory\_size}})$  words between fast and slow memory, attaining the lower bound. See "Avoiding Communication in Successive Band Reduction" linked on the class webpage for details.

When  $A$  is banded, a different algorithm (in `ssbtrd`) takes advantage of the band structure to reduce to tridiagonal form in just  $O(n^2 \cdot \text{bandwidth})$  operations. This can also be done while doing much less communication (the lower bound itself is an open question).

In the case of the SVD, we begin by reducing the general nonsymmetric (even rectangular)  $A$  to  $U^T \cdot A \cdot V = B$  where  $U$  and  $V$  are orthogonal and  $B$  is bidiagonal, i.e. nonzero on the main diagonal and right above it. All subsequent SVD algorithms operate on  $B$ . This is currently done by LAPACK routine `sgebrd`. All the ideas for minimizing communication mentioned above generalize to the SVD.

(1.1) Given  $T$  we need to find all its eigenvalues (and possibly vectors)

There are a variety of algorithms; all cost  $O(n^2)$  just for eigenvalues, but anywhere from  $O(n^2)$  to  $O(n^3)$  for eigenvectors. Also some have better numerical stability properties than others. We summarize their properties here, and describe a few in more detail below.

(1.1.1) Oldest is QR iteration, as in chapter 4, but for tridiagonal  $T$ .

Thm (Wilkinson): With the right shift, tridiagonal QR is globally convergent, and usually cubically convergent (# correct digits triples at each step!). (We sketch a partial proof later.)

It costs  $O(n^2)$  to get all the eigenvalues, but costs  $O(n^3)$  to get the vectors, unlike later routines. Since it only multiplies orthogonal matrices, it is backward stable by the same analysis as Chap 4. It is used by LAPACK routine `ssyev`. This was the standard approach for many years because of its reliability.

LAPACK routine `sgesvd` uses a variant of QR iteration for the SVD, which has the additional property of guaranteed high relative accuracy for all singular values, no matter how small, if the input is bidiagonal (see "Accurate singular values of bidiagonal matrices" on the class webpage).

(1.1.2) Another approach, which is much faster for computing eigenvectors ( $O(n^2)$  instead of  $O(n^3)$ ) but does not guarantee that they are orthogonal, works as follows:

(1) compute the eigenvalues alone (`sstebz` in LAPACK)

(2) compute the eigenvectors using inverse iteration (`sstein` in LAPACK)

$$x_{(i+1)} = \text{inv}(T - \lambda(j)I) * x_{(i)}$$

Since  $T$  is tridiagonal, one steps of inverse iteration costs  $O(n)$ , and since  $\lambda(j)$  is an accurate eigenvalue, it should converge in very few steps. The trouble is that when  $\lambda(j)$  and  $\lambda(j+1)$  are very close, and so the eigenvectors are very sensitive, they may not be computed to be orthogonal, since there is nothing in the algorithm that explicitly enforces this (imagine what would happen if  $\lambda(j)$  and  $\lambda(j+1)$  were so close that they rounded to the same floating point number). Still, the possibility of having an algorithm that ran in  $O(n^2)$  time but guaranteed orthogonality was a goal for many years, with the eventual algorithm discussed below (MRRR).

(1.1.3) Next is "divide-and-conquer", which is faster than QR, but not as fast as inverse iteration.

It is used in LAPACK routine `ssyevd` (`sgesdd` for SVD).

Its speed is harder to analyze, but empirically it is like  $O(n^g)$  for some  $2 < g < 3$ .

The idea behind it is used for the updating problem, i.e. getting the eigenvalues and vectors of  $A + x x^T$  given those of  $A$ .

(1.1.4) Most recent is MRRR = Multiple Relatively Robust Representations. which can be thought as a version of inverse iteration that does guarantee orthogonal eigenvectors, and still cost just  $O(n^2)$ . It was developed by Prof. Parlett and former student Inderjit Dhillon, (see "Orthogonal Eigenvectors and Relative Gaps" on the class webpage). Since the output eigenvector matrix is of size  $n^2$ , it is optimal (but see below). It is implemented in LAPACK routine `ssyevr`. The adaptation of this algorithm for the SVD appeared in the prize-winning thesis of Paul Willems, but some examples of matrices remain where it does not achieve the desired accuracy, so it is still an open problem to make this routine reliable enough for general use.

In theory, there is an even faster algorithm than  $O(n^2)$ , based on divide-and-conquer, by representing the eigenvector matrix  $Z$  of  $T$  implicitly rather than as  $n^2$  explicit matrix entries, but since most users want explicit eigenvectors, and the cost of reduction to tridiagonal form  $T$  is already much larger, we do not usually use it:

Thm (Ming Gu): One can compute  $Z$  in  $O(n \log^p n)$  time, provided we represent it implicitly (so that we can multiply any vector by it cheaply). Here  $p$  is a small integer.

Thus  $A = Q^T Q^T A Q^T = Q^T Z \Lambda Z^T Q^T = (Q^T Z) \Lambda (Q^T Z)^T$ , so we need to multiply  $Q^T Z$  to get final eigenvectors (costs  $O(n^3)$ )

(1.2) or (1.3) Reduce  $A$  to  $Q^T A Q = T$  as above.

Use bisection (based on Sylvester's Thm) on  $T$  to get a few eigenvalues, and then inverse iteration to get their eigenvectors if desired.

This is cheap,  $O(n)$  per eigenvalue/vector of  $T$ , but does not guarantee orthogonality of eigenvectors of nearby eigenvalues `ssyevx` in LAPACK. MRRR (in `ssyevr`) could be used for this too, and guarantee orthogonality.

(2) Based on Jacobi's Method, the historically oldest algorithm. The modern version is by Drmac and Veselic, called `sgesvj` for the SVD (not yet for the eigenproblem).

(3) Use same idea as divide-and-conquer: assuming we have the eigenvalues and eigenvectors for  $A = Q \Lambda Q^T$ , it is possible to compute the eigenvalues of  $A + u u^T$  in  $O(n^2)$ , much faster than starting from scratch.

(We pause the recorded lecture here.)

We now illustrate important properties of some of these algorithms.

Matlab demo of QR: illustrating cubic convergence

```
n=6; A = randn(n,n), A = A+A', T = hess(A),
s = T(n,n); [Q,R]=qr(T-s*eye(n)); T = R*Q+s*eye(n);
off=triu(tril(T,-1),-1);T=off+off'+diag(diag(T))
```

... the last line just ensures that T is symmetric and tridiagonal

Cubic convergence will follow from analysis of a simpler algorithm called Rayleigh Quotient iteration:

```
choose unit vector x_0
i = 0
repeat
  s_i = rho(x_i,A) = x_i^T*A*x_i
  y = (A - s_i*I)^(-1) * x_i
  x_(i+1) = y / ||y||_2
  i = i+1
until convergence ( s_i and/or x_i stop changing )
```

This is what we called inverse iteration before, using the Rayleigh Quotient  $s_i$  as a shift, which we showed was the best possible eigenvalue approximation for the approximate eigenvector  $x_i$ .

Suppose that  $A*q = \lambda*q$ ,  $\|q\|_2 = 1$ , and  $\|x_i - q\| = e \ll 1$ .

We want to show that  $\|x_{i+1} - q\| = O(e^3)$ .

We need to bound  $|s_i - \lambda|$ ; a first bound is

$$\begin{aligned} s_i &= x_i^T A x_i = (x_i - q + q)^T A (x_i - q + q) \\ &= (x_i - q)^T A (x_i - q) + q^T A (x_i - q) + (x_i - q)^T A q + q^T A q \\ &= (x_i - q)^T A (x_i - q) + \lambda q^T (x_i - q) + (x_i - q)^T q \lambda + \lambda \end{aligned}$$

$$\text{so } s_i - \lambda = (x_i - q)^T A (x_i - q) + 2\lambda (x_i - q)^T q$$

$$\text{so } |s_i - \lambda| = O(\|x_i - q\|^2 + \|x_i - q\|) = O(\|x_i - q\|) = O(e)$$

A tighter bound is

$$\begin{aligned} |s_i - \lambda| &\leq \|A x_i - s_i x_i\| / \text{gap} \\ &\quad \dots \text{ where gap} = \text{distance from } s_i \text{ to next closest eigenvalue} \\ &\quad \dots \text{ we assume the gap is not too small} \\ &\quad \dots \text{ This is Thm 5.5 in the text, proof sketch above} \\ &= \|A(x_i - q + q) - s_i(x_i - q + q)\| / \text{gap} \\ &= \|(A - s_i I)(x_i - q) + (\lambda - s_i)q\| / \text{gap} \\ &\leq (\|(A - s_i I)(x_i - q)\| + \|(\lambda - s_i)q\|) / \text{gap} \\ &= O(e^2) \end{aligned}$$

Now we take one step of inverse iteration to get  $x_{i+1}$ ; from earlier analysis (Sec 4.4.2, discussed in Chap 4 lectures) we know

$$\begin{aligned} \|x_{i+1} - q\| &\leq \|x_i - q\| * |s_i - \lambda| / \text{gap} \\ &= e * O(e^2) / \text{gap} \\ &= O(e^3) \text{ as desired} \end{aligned}$$

To see that QR iteration is doing Rayleigh Quotient iteration in disguise, look at one step:

$$T - s(i)I = Q^*R \text{ so}$$

$$\begin{aligned} (T - s(i)I)^{-1} &= R^{-1}Q^{-1} \\ &= R^{-1}Q^* \text{ ... since } Q \text{ is orthogonal} \\ &= (R^{-1}Q^*)^T \text{ ... since the matrix is symmetric} \\ &= Q^*R^{-T} \end{aligned}$$

$$\text{so } (T - s(i)I)^{-1} * R^T = Q$$

$$\text{so } (T - s(i)I)^{-1} * e_n * R(n,n) = q_n = \text{last column of } Q$$

Since  $s(i) = T(n,n) = e_n^T * T * e_n$ ,  $s(i)$  is just the Rayleigh Quotient for  $e_n$ , and  $q_n$  is the result of one step of Rayleigh quotient iteration.

Then the updated  $T$  is  $R^*Q + s(i)I = Q^*T^*Q$ , so

(updated  $T$ )( $n,n$ ) =  $q_n^T * T * q_n$  is the next Rayleigh quotient as desired.

In practice QR iteration is implemented analogously to Chap 4, by "bulge chasing", at a cost of  $O(n)$  per iteration, or  $O(n^2)$  to find all the eigenvalues. But multiplying together all the Givens rotations still costs  $O(n^3)$ , so something faster is still desirable.

(We pause the recorded lecture here.)

Divide-and-conquer: The important part is how to cheaply compute the eigendecomposition of a rank-one update to a symmetric matrix  $A + \alpha * u * u^T$ , where we already have the eigendecomposition  $A = Q * \Lambda * Q^T$ . Then

$$\begin{aligned} A + \alpha * u * u^T &= Q * \Lambda * Q^T + \alpha * u * u^T \\ &= Q * (\Lambda + \alpha * (Q^T * u) * (Q^T * u)^T) * Q^T \\ &= Q * (\Lambda + \alpha * v * v^T) * Q^T \end{aligned}$$

so we only need to think about computing the eigenvalues and eigenvectors of  $\Lambda + \alpha * v * v^T$ . Let's compute its characteristic polynomial:

$$\text{Lemma (homework!)} \det(I + x * y^T) = 1 + y^T * x$$

Then the characteristic polynomial is

$$\begin{aligned} \det(\Lambda + \alpha * v * v^T - \lambda I) &= \det((\Lambda - \lambda I) * (I + \alpha * (\Lambda - \lambda I)^{-1} * v * v^T)) \\ &= \prod_i (\lambda_i - \lambda) * (1 + \alpha * \sum_i v(i)^2 / (\lambda_i - \lambda)) \\ &= \prod_i (\lambda_i - \lambda) * f(\lambda) \end{aligned}$$

So our goal is to solve the so-called secular equation  $f(\lambda) = 0$ .

Consider figure 5.2 in the text (Secular1.ps): this plot of

$f(\lambda) = 1 + .5/(1-\lambda) + .5/(2-\lambda) + .5/(3-\lambda) + .5/(4-\lambda)$  looks benign: we see there is one root of  $f(\lambda)$  between each pair  $[\lambda_i, \lambda_{i+1}]$  of adjacent eigenvalues, and in each such

interval  $f(\lambda)$  is monotonic, so some Newton-like method should work well.

But now consider figure 5.3 in the text (Secular2.ps):

$$f(\lambda) = 1 + .001/(1-\lambda) + .001/(2-\lambda) + .001/(3-\lambda) + .001/(4-\lambda)$$

Here  $f(\lambda)$  no longer looks so benign, and simple Newton would not work, taking a big step out of the bounding interval. But we can still use Newton, but instead of approximating the  $f(\lambda)$  by a straight line at each step, and getting the next guess by finding a zero of that straight line, we approximate  $f(\lambda)$  by a simple but non-linear function that better matches the graph, with poles at  $\lambda_i$  and  $\lambda_{i+1}$ :

$$g(\lambda) = c_1 + c_2/(\lambda_i - \lambda) + c_3/(\lambda_{i+1} - \lambda)$$

where  $c_1$ ,  $c_2$  and  $c_3$  are chosen as in Newton to match  $f$  and  $f'$  at the last approximate 0. Solving  $g(\lambda)=0$  is then solving a quadratic for  $\lambda$ . (picture).

It remains to compute the eigenvectors:

Lemma: If  $\lambda$  is an eigenvalue of  $\Lambda + \alpha v v^T$ , then

$(\Lambda - \lambda I)^{-1} v$  is its eigenvector. Since  $\Lambda$  is diagonal, this costs  $O(n)$  to compute.

Proof:  $(\Lambda + \alpha v v^T)(\Lambda - \lambda I)^{-1} v$   
 $= (\Lambda - \lambda I + \lambda I + \alpha v v^T)(\Lambda - \lambda I)^{-1} v$   
 $= v + \lambda (\Lambda - \lambda I)^{-1} v$   
 $+ v (\alpha v^T (\Lambda - \lambda I)^{-1} v)$   
 $= v + \lambda (\Lambda - \lambda I)^{-1} v - v$   
 since  $\alpha v^T (\Lambda - \lambda I)^{-1} v + 1 = f(\lambda) = 0$   
 $= \lambda (\Lambda - \lambda I)^{-1} v$  as desired

Unfortunately this simple formula is not numerically stable; the text describes the clever fix (due to Ming Gu and Stan Eisenstat, for which Prof. Gu won the Householder Prize for best thesis in Linear Algebra in 1996).

When two eigenvalues  $\lambda_i$  and  $\lambda_{i+1}$  are (nearly) identical, then we know there is a root between them without doing any work. Similarly when  $v(i)$  is (nearly) zero, we know that  $\lambda(i)$  is (nearly) an eigenvalue without doing any work. In practice a surprising number of eigenvalues can be found quickly ("deflated") this way, which speeds up the algorithm.

Write  $[Q', \Lambda'] = \text{Eig\_update}[Q, \Lambda, \alpha, u]$  as the function we just described that takes the eigendecomposition of  $A = Q \Lambda Q^T$  and updates it, returning the eigendecomposition of

$$A + \alpha u u^T = Q' \Lambda' Q'^T$$

We can also use  $\text{Eig\_update}$  as a building block for an entire eigensolver, using Divide-and-Conquer. To do so we need to see how to divide a tridiagonal matrix into two smaller ones of half the size, just



and eigenvectors are desired.

Recall Sylvester's Theorem: Suppose  $A$  is symmetric and  $X$  nonsingular. Then  $A$  and  $X^*A^*X^T$  have the same Inertia = (#evals<0,#evals=0,#evals>0). So  $A - \sigma I$  and  $X^*(A - \sigma I)^*X^T$  have the same Inertia, namely (#evals of  $A < \sigma$ , #evals of  $A = \sigma$ , #evals of  $A > \sigma$ ) So if  $X^*(A - \sigma I)^*X^T$  were diagonal, it would be easy to count the number of eigenvalues of  $A$  less than, equal to or greater than  $\sigma$ , for any  $\sigma$ . By doing this for  $\sigma_1$  and  $\sigma_2$ , we can count the number of eigenvalues in any interval  $[\sigma_1, \sigma_2]$ , and by repeatedly cutting intervals in half, we can compute intervals containing the eigenvalues that are as narrow as we like, or that only contain eigenvalues in regions of interest (eg the smallest).

But how do we cheaply choose  $X$  to make  $X^*(A - \sigma I)^*X^T$  diagonal? By starting with  $A = T$  tridiagonal, and doing Gaussian elimination without pivoting:  $T - \sigma I = L^*D^*L^T$ , so  $\text{inv}(L)^*(T - \sigma I)^*\text{inv}(L)^T$  and  $D$  have the same inertia.

However, LU without pivoting seems numerically dangerous. Fortunately, because of the tridiagonal structure, it is not, if done correctly:

```
function c = Negcount(T,s) ... count the number c of eigenvalues of  $T < s$ 
... assume  $T$  has  $a(1), \dots, a(n)$  on diagonal and  $b(1), \dots, b(n-1)$  off-diagonal
... only compute diagonal entries  $d(i)$  of  $D$  in  $T - sI = L^*D^*L^T$ 
c = 0
b(0) = 0
d(0) = 1
for i = 1 to n
    d(i) = (a(i) - s) - b(i-1)^2/d(i-1) ... need to obey parentheses!
    if (d(i)<0), c=c+1
end
```

We don't need  $l(i) = i$ -th off diagonal of  $L$ , because  $(T - sI)(i, i+1) = b(i) = (L^*D^*L^T)(i, i+1) = d(i)l(i)$  so we can replace the usual inner loop of  $LDL^T$  (analogous to LU)  $d(i) = a(i) - s - l(i-1)^2 * d(i-1)$  by  $d(i) = a(i) - s - b(i-1)^2 / d(i-1)$

Thm: Assuming we don't divide by zero, overflow or underflow, the computed  $c$  from  $\text{Negcount}(T, s)$  is exactly correct for a slightly perturbed  $T + E$ , where  $E$  is symmetric and tridiagonal,

$E(i,i) = 0$  (the diagonal is not perturbed at all) and  
 $|E(i,i+1)| \leq 2.5 \cdot \text{macheps} \cdot |T(i,i+1)|$

Proof: As before, we replace each operation like  $a+b$  in the algorithm with  $(a+b) \cdot (1+\delta)$  where  $|\delta| \leq \text{macheps}$ . If we obey the parentheses in the algorithm, so  $a(i) - s$  is subtracted first, then we can divide out by all the  $(1+\delta)$  factors multiplying  $a(i)-s$  to get

$$d(i) \cdot F(i) = a(i) - s - b(i-1)^2 \cdot G(i) / d(i-1)$$

where  $F(i)$  and  $G(i)$  are the product of 2 or 3  $(1+\delta)$  factors.

Now write this as

$$d(i) \cdot F(i) = a(i) - s - b(i-1)^2 \cdot G(i) \cdot F(i-1) / (d(i-1) \cdot F(i-1))$$

or

$$d'(i) = a(i) - s - b'(i-1)^2 / d'(i-1)$$

the exact recurrence for  $T+E$ . Since  $d(i)$  and  $d'(i) = d(i) \cdot F(i)$

have the same sign, we get the same exact Negcount for either.

In fact more is true: This works even if some pivot  $d(i-1)$  is exactly zero, and we divide by zero, so the next  $d(i)$  is infinite, because we end up dividing by  $d(i)$  in the next step, and the "infinity" disappears. But it does mean that to correctly compute Negcount, we need to count  $-0$  as negative, and  $+0$  as positive ( $-0$  is a feature of IEEE floating point arithmetic). Furthermore, the function  $g(\sigma) = \# \text{evals} < \sigma$  is computed monotonically increasing, as long as the arithmetic is correctly rounded (otherwise the computed number of eigenvalues in a narrow enough interval might be negative!).

The cost of Negcount is clearly  $O(n)$ . To find one eigenvalue with Negcount via bisection, Negcount needs to be called at most 64 (in double) or 32 (in single) times, since each time the interval gets half as big, and essentially determines one more bit in the floating point format. So the cost is still  $O(n)$ . So to find  $k$  eigenvalues using bisection, the cost is  $O(k \cdot n)$ .

(We pause the recorded lecture here.)

Given an accurate eigenvalue  $\lambda_j$  from bisection, we find its eigenvector by inverse iteration:

choose  $x(0)$ ,  $i=0$

repeat

$i = i+1$

solve  $(T - \lambda_j I) \cdot y = x(i-1)$  for  $y$

$x(i) = y / \|y\|_2$

until  $x(i)$  converges

which should converge in a few steps, and so at cost  $O(n)$  per eigenvector.

This seems like an optimal algorithm for all  $n$  eigenvalues and eigenvectors: at a cost of  $O(n)$  per eigenpair, the total cost should be  $O(n^2)$ . But it has a problem: the computed eigenvectors, while they individually very nearly satisfy  $A*x = \lambda(j)*x$  as desired, may not be orthogonal when  $\lambda(j)$  is very close to another  $\lambda(j+1)$ ; there is nothing in the algorithm to enforce this, and when  $\lambda(j)$  is close to  $\lambda(j+1)$ , solving with  $T - \lambda(j)*I$  is nearly the same as solving with  $T - \lambda(j+1)*I$ . Imagine the extreme case where  $\lambda(j)$  and  $\lambda(j+1)$  are so close that they round to the same floating point number! One could start with a random starting vector  $x(0)$  and hope for the best but there is no guarantee of orthogonality.

At first people tried to guarantee orthogonality by taking all the computed eigenvectors belonging to a cluster of nearby eigenvalues and running QR decomposition on them, replacing them by the columns of  $Q$ . This guarantees orthogonality, but has two problems:

- (1) if the computed vectors are not linearly independent, the columns of  $Q$  may not satisfy  $A*q = \lambda*q$  very well.
- (2) if the size of the cluster of close eigenvalues is  $s$ , the cost of QR decomposition is  $O(s^2*n)$ , so if  $s$  is large, the cost could go back up to  $O(n^3)$ .

The MRRR = Multiple Relatively Robust Representations algorithm was motivated by this problem: computing all the eigenvectors in  $O(n^2)$  time such that are also guaranteed orthogonal. It is the fastest algorithm available (for large enough problems; it defaults to other algorithms for small problems), but is rather complicated to explain. Two lectures explaining it in detail are available from the 2004 Ma221 web page, see the class web site for details.

The final algorithm of importance is Jacobi's method, whose classical (and so slow) implementation is described in section 5.3.5. Jacobi can be shown to be potentially the most accurate algorithm, with an error determined by a version of Weyl's Theorem for relative error (Thm 5.6 in the text), and so getting the tiny eigenvalues (and their eigenvectors) much more accurately. It was much slower than the other methods discussed so far, until work by Drmac and Veselic showed how it could be made much faster.

All the algorithms discussed so far extend to the SVD, typically by using the relationship between the SVD of  $A$  and the eigenvalues and eigenvectors of the symmetric matrix  $\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$ . The one exception is MRRR, where some open problems remain, addressed in the 2010 PhD dissertation of Paul Willems, although there are still examples where Willems' code loses significant accuracy.