Fundamentals

Norms

- Induced matrix norms
  - $||A||_1$: maximum column sum of A
    - pf. Show the norm is bounded above by max column sum, use e_j to show equality.
  - $||A||_2$: Largest singular value
  - $||A||_\infty$: maximum row sum of A
    - Same idea as with 1 norm
  - $||A||_F$: The square root of the sum of the squared singular values.
  - 2-norm and F-norm are invariant under unitary multiplication

Singular Value Decomposition (SVD)

Theorems

- Every matrix has an SVD, singular values are uniquely determined. If A is square and singular values are distinct, then left and right singular vectors are unique up to complex signs.
- The nonzero singular values of A are the square roots of the nonzero singular values of A^*A
- For square A, its determinant is the product of singular values (which is the product of eigenvalues)

Low Rank Approximations

Theorems

- $A = \sum_{j=1}^{r} \sigma_j u_j v_j^*$
- For any $0 \leq \nu \leq r$, def $A_\nu = \sum_{j=1}^{\nu} \sigma_j u_j v_j^*$, if ν = min(m,n), we can define $\sigma_{\nu+1} = 0$. Then we have $||A - A_\nu||_2 = \inf ||A - B||_2 = \sigma_{\nu+1}$
- A_ν minimizes the error in the Frobenius norm where the error is the sqrt of the sum of the remaining squared singular values.

QR Factorization and Least Squares

Projectors

- A square matrix P is a projector is $P^2 = P$. AKA Idempotent
- We can think of a projector as splitting the space into two.
- Orthogonal projectors
  - hermitian projectors
  - Given matrix with orthonormal columns $\hat{Q}_n$ we have P = $\hat{Q}_n \hat{Q}_n^*$
  - Rank 1 projector: P = qq*

QR Factorization

Consider m x n matrix A
Reduced QR: $\hat{Q} \in \mathbf{C}^{m \times n}$, $\hat{R} \in \mathbf{C}^{n \times n}$
Full QR: $Q \in \mathbf{C}^{m \times m}$, $R \in \mathbb{C}^{m \times n}$
Gram-Schmidt Orthogonalization presents a way to form reduced QR

- Like successive orthogonalization
- **Algorithm 7.1. Classical Gram–Schmidt (unstable)**

  for $j = 1$ **to** n
  $v_j = a_j$
  for $i = 1$ **to** $j - 1$
  $r_{ij} = q_i^* a_j$
  $v_j = v_j - r_{ij} q_i$
  $r_{jj} = \|v_j\|_2$
  $q_j = v_j / r_{jj}$
- Every matrix $A \in \mathbb{C}^{m \times n}$ where m > n has a full QR decomposition (hence reduced)
- Every full rank A has a unique QR factorization if positive diagonals.

Gram Schmidt Orthogonalization

We can think of GS as each iterations projecting the column a_j into the space orthogonal to the previous collected q's.

Algorithm 7.1 computes one large projection of rank m - (j-1).

For each value of j, Algorithm 7.1 computes a single orthogonal projection of rank $m-(j-1)$,

$$v_j = P_j a_j. \tag{8.4}$$

In contrast, the modified Gram–Schmidt algorithm computes the same result by a sequence of $j-1$ projections of rank $m-1$. Recall from (6.9) that $P_{\perp q}$ denotes the rank $m-1$ orthogonal projector onto the space orthogonal to a nonzero vector $q \in \mathbb{C}^m$. By the definition of P_j, it is not difficult to see that

$$P_j = P_{\perp q_{j-1}} \cdots P_{\perp q_2} P_{\perp q_1}, \tag{8.5}$$

again with $P_1 = I$. Thus an equivalent statement to (8.4) is

$$v_j = P_{\perp q_{j-1}} \cdots P_{\perp q_2} P_{\perp q_1} a_j. \tag{8.6}$$

The modified Gram–Schmidt algorithm is based on the use of (8.6) instead of (8.4).

Algorithm 8.1. Modified Gram–Schmidt

for $i = 1$ **to** n
    $v_i = a_i$
for $i = 1$ **to** n
    $r_{ii} = \|v_i\|$
    $q_i = v_i / r_{ii}$
    for $j = i+1$ **to** n
        $r_{ij} = q_i^* v_j$
        $v_j = v_j - r_{ij} q_i$

In MGS, you are projecting the rest of the vectors into the space orthogonal to the qs you have already formed.
This can be viewed as "triangular orthogonalization" where each iteration multiplies on the right by an upper triangular matrix to form Q. This bears a close resemblance to gaussian elimination.

Householder Triangularization

The aim of householder triangularization is to successively form an upper triangular matrix through a series of orthogonal transformations. At iteration k, we want to introduce zeros below the k-th diagonal entry.

Better of two reflectors: we want to reflect over the vector not too close to itself

At iteration k:
$x = A_{k:m,k}$ (the vector of the kth column from rows k:m)
$v = -sign(x_1)||x||e_1 - x$ or $v = sign(x_1)||x||e_1 + x$
$F = I - 2\frac{vv^*}{v^*v}$

Conditioning and Stability

Condition number of matrix vector product: $||A||\frac{||x||}{||Ax||}$
Condition number of Matrix: K(A) =$||A||||A^{-1}||$

Systems of Equations

Gaussian Elimination

LU Factorization:
A = LU by triangular triangularization

Let A be an m x m square matrix (can be done for nonsquare but rare)
$L_{m-1} \dots L_2 L_1 A = U$
Where L are lower triangular and U is upper triangular. Then our LU decomposition becomes
A = LU where L = $L_1^{-1} L_2^{-2} \dots L_{m-1}^{-1}$

At step k (starting at 1), we are trying to make the k-th column of U 0 below the diagonal

L_k must be chosen so that

$$x_k = \begin{bmatrix} x_{1k} \\ \vdots \\ x_{kk} \\ x_{k+1,k} \\ \vdots \\ x_{mk} \end{bmatrix} \quad \overset{L_k}{\longrightarrow} \quad L_k x_k = \begin{bmatrix} x_{1k} \\ \vdots \\ x_{kk} \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

To do this we wish to subtract ℓ_{jk} times row k from row j, where ℓ_{jk} is the *multiplier*

$$\ell_{jk} = \frac{x_{jk}}{x_{kk}} \qquad (k < j \le m). \tag{20.6}$$

The matrix L_k takes the form

$$L_k = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -\ell_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & -\ell_{mk} & & 1 \end{bmatrix},$$

Two strokes of luck:

1. $L_n^{-1} = L_n$ with sub diagonal entries negated
2. L is just the unit lower triangular matrix with the nonzero columns of $L_1^{-1} L_2^{-2} \dots L_{m-1}^{-1}$ inserted into the right places.

Algorithm 20.1. Gaussian Elimination without Pivoting

$U = A, \ L = I$
for $k = 1$ **to** $m - 1$
  for $j = k + 1$ **to** m
    $\ell_{jk} = u_{jk}/u_{kk}$
    $u_{j,k:m} = u_{j,k:m} - \ell_{jk} u_{k,k:m}$

Instability without pivoting
if a pivot has an extremely small number, dividing by this pivot can introduce a lot of numerical instability

Pivoting

Pivoting allows us to address the instability of GE.
We generally want to select pivots as the largest number from set of candidates
At iteration k

- We move pivot candidate x_{ij} into the (k, k) position by permuting the rows and columns

Partial pivoting

Select the largest subdiagonal entry in column k. This is equivalent to performing regular LU on a permutated matrix PA.
PA = LU
Why? Our third stroke of luck
Selecting the largest subdiagonal entry at interation k looks like

$$L_n P_n \dots L_1 P_1 A = U$$

To show an example of our third stroke of luck, consider

$$L_3 P_3 L_2 P_2 L_1 P_1 A = U$$

This can be reordered as

$$L_3' L_2' L_1' P_3 P_2 P_1 A = U$$

Where

$$L_n' = L_n,\ L_2' = P_3 L_2 P_3^{-1},\ L_1' = P_3 P_2 L_1 P_2^{-1} P_3^{-1}$$

In general, for an $m \times m$ matrix, the factorization (21.1) provided by Gaussian elimination with partial pivoting can be written in the form

$$(L_{m-1}' \cdots L_2' L_1')(P_{m-1} \cdots P_2 P_1) A = U, \tag{21.5}$$

where L_k' is defined by

$$L_k' = P_{m-1} \cdots P_{k+1} L_k P_{k+1}^{-1} \cdots P_{m-1}^{-1}. \tag{21.6}$$

Of course, since we do not know P ahead of time, we do not perform standard gaussian elimination on PA.

Algorithm 21.1. Gaussian Elimination with Partial Pivoting

$U = A,\ L = I,\ P = I$
for $k = 1$ **to** $m - 1$
  Select $i \geq k$ to maximize $|u_{ik}|$
  $u_{k,k:m} \leftrightarrow u_{i,k:m}$ (interchange two rows)
  $\ell_{k,1:k-1} \leftrightarrow \ell_{i,1:k-1}$
  $p_{k,:} \leftrightarrow p_{i,:}$
  for $j = k + 1$ **to** m
    $\ell_{jk} = u_{jk}/u_{kk}$
    $u_{j,k:m} = u_{j,k:m} - \ell_{jk} u_{k,k:m}$

Questions: How to recover the original LU after pivoting?

Stability of GE

While proven to be both forward and backward stable, the bounds that guarantee stability are batshit insane (2^m). It turns out in practice, the growth factors that contribute to this bound are never seen in reality. This requires a very special column space of A an L. Growth factor:

$$\rho = \frac{\max_{i,j} |u_{ij}|}{\max_{i,j} |a_{ij}|}.$$

Worst-Case Instability

For certain matrices A, despite the beneficial effects of pivoting, ρ turns out to be huge. For example, suppose A is the matrix

$$A = \begin{bmatrix} 1 & & & & 1 \\ -1 & 1 & & & 1 \\ -1 & -1 & 1 & & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix}. \tag{22.4}$$

At the first step, no pivoting takes place, but entries $2, 3, \ldots, m$ in the final column are doubled from 1 to 2. Another doubling occurs at each subsequent elimination step. At the end we have

$$U = \begin{bmatrix} 1 & & & & 1 \\ & 1 & & & 2 \\ & & 1 & & 4 \\ & & & 1 & 8 \\ & & & & 16 \end{bmatrix}. \tag{22.5}$$

The final $PA = LU$ factorization looks like this:

$$\begin{bmatrix} 1 & & & & 1 \\ -1 & 1 & & & 1 \\ -1 & -1 & 1 & & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ -1 & 1 & & & \\ -1 & -1 & 1 & & \\ -1 & -1 & -1 & 1 & \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & & & & 1 \\ & 1 & & & 2 \\ & & 1 & & 4 \\ & & & 1 & 8 \\ & & & & 16 \end{bmatrix}.$$

For this 5×5 matrix, the growth factor is $\rho = 16$. For an $m \times m$ matrix of the same form, it is $\rho = 2^{m-1}$. (This is as large as ρ can get; see Exercise 22.1.)

Cholesky Factorization

How about LU for symmetric matrices. Given hermitian, positive definite $A \in \mathbb{C}^{mxm}$, we want to find upper triangular matrix R s.t $A = R^*R$

Cholesky Factorization

In order for the symmetric triangular reduction to work in general, we need a factorization that works for any $a_{11} > 0$, not just $a_{11} = 1$. The generalization of (23.1) is accomplished by adjusting some of the elements of R_1 by a factor of $\sqrt{a_{11}}$. Let $\alpha = \sqrt{a_{11}}$ and observe:

$$A = \begin{bmatrix} a_{11} & w^* \\ w & K \end{bmatrix}$$

$$= \begin{bmatrix} \alpha & 0 \\ w/\alpha & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & K - ww^*/a_{11} \end{bmatrix} \begin{bmatrix} \alpha & w^*/\alpha \\ 0 & I \end{bmatrix} = R_1^* A_1 R_1.$$

This is the basic step that is applied repeatedly in Cholesky factorization. If the upper-left entry of the submatrix $K - ww^*/a_{11}$ is positive, the same formula can be used to factor it; we then have $A_1 = R_2^* A_2 R_2$ and thus $A = R_1^* R_2^* A_2 R_2 R_1$. The process is continued down to the bottom-right corner, giving us eventually a factorization

$$A = \underbrace{R_1^* R_2^* \cdots R_m^*}_{R^*} \underbrace{R_m \cdots R_2 R_1}_{R}. \tag{23.2}$$

This equation has the form

$$A = R^* R, \qquad r_{jj} > 0, \tag{23.3}$$

where R is upper-triangular. A reduction of this kind of a hermitian positive definite matrix is known as a *Cholesky factorization*.

Theorems:

- Every hermitian positive definite matric has a Cholesky factorization
- Always stable, subtleties in standard Gaussian elimination disappear.
  - matrix norm of R is bounded above by A
- Implies there is always a stable way to solve hermitian positive definite systems of equations $Ax = b$

Algorithm 23.1. Cholesky Factorization

$R = A$
for $k = 1$ **to** m
    for $j = k + 1$ **to** m
        $R_{j,j:m} = R_{j,j:m} - R_{k,j:m} R_{kj} / R_{kk}$
    $R_{k,k:m} = R_{k,k:m} / \sqrt{R_{kk}}$

Eigenvalues

Eigenvalue Problems

Unitary Diagonalization:

- A matrix A is unitarily diagonalizable iff it is normal
- A matrix is normal iff $A^*A = AA^*$

Schur Factorization:
THM: Every square matrix A has a schur factorization $A = QTQ^*$, where Q is orthogonal and T is upper triangular.
Steps to find Schur factorization:

Eigenvalue-Revealing Factorizations

In the preceding pages we have described three examples of *eigenvalue-revealing factorizations*, factorizations of a matrix that reduce it to a form in which the eigenvalues are explicitly displayed. We can summarize these as follows.

A diagonalization $A = X\Lambda X^{-1}$ exists if and only if A is nondefective.
A unitary diagonalization $A = Q\Lambda Q^*$ exists if and only if A is normal.
A unitary triangularization (Schur factorization) $A = QTQ^*$ always exists.

Overview of Eigenvalue Problems

The algebraic nature of eigenvalue problems (which can be represented as root finding problems and vice versa) tell us that there can be no finite method for finding exact solutions. Thus, we are left to the family of iterative methods.

Two phases of eigenvalue computation:

1. Reduction to Hessenburg form (zeros below first subdiagonal)
   - $O(m^3)$
2. Iterative portion
   - Generally convergence to machine precision is achieved in $O(m)$ iterations.
   - Each iteration is at cost $O(m^2)$
   - Total requirement is $O(m^3)$ flops

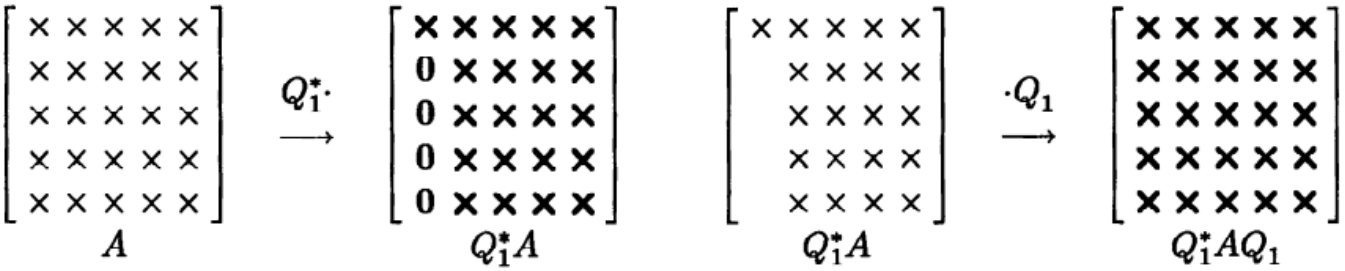
   Without reduction to hessenburg form, reach iteration would involve a full matrix, requiring $O(m^3)$ work. NOT GOOD

   If A is Hermitian, the Hessenburg form is tridiagonal: $O(m)$

Reduction to Hessenburg or Tridiagonal form

Our goal is to introduce zeros below the main subdiagonal. It is important to remember we are trying to construct a *similar matrix*, that is, the hessenburg form must have have the same eigenvalues as A if we want to find the eigenvalues of A. Thus, any transformation we apply to A on the left must also be applied on the right: $H = QAQ^*$

If we naively apply householder reflector Q_1 from QR, we lose all the zeros introduced by left multiplication when we apply Q_1 to the right. .



A good idea is to apply a reflector that leaves the first row/column unchanged.
We can think of our first reflector as the following block matrix:

$$\left[\begin{array}{c|ccc} 1 & 0 & 0 & 0 \\ \hline 0 & & & \\ 0 & & F_1 & \\ 0 & & & \end{array}\right]$$

where F_1 is the householder reflector for the (m-1) vector $A_{2:m,1}$
As we iterate, the identity matrix in the top left gets larger and our bottom right reflector gets smaller.

Algorithm 26.1. Householder Reduction to Hessenberg Form

for $k = 1$ **to** $m - 2$

$$x = A_{k+1:m,k}$$
$$v_k = \text{sign}(x_1)\|x\|_2 e_1 + x$$
$$v_k = v_k / \|v_k\|_2$$
$$A_{k+1:m,k:m} = A_{k+1:m,k:m} - 2v_k(v_k^* A_{k+1:m,k:m})$$
$$A_{1:m,k+1:m} = A_{1:m,k+1:m} - 2(A_{1:m,k+1:m} v_k)v_k^*$$

Stability: Backwards stable

Rayleigh Quotient, Inverse Iteration

IN THE FOLLOWING 4 LECTURES, WE RESTRICT OURSELF TO THE STUDY OF REAL, SYMMETIC MATRICES. $A \in \mathbb{R}^{m \times m}$

A priori, A has a complete set of orthogonal eigenvectors
Notation:

real eigenvalues: $\lambda_1, \ldots, \lambda_m$,

orthonormal eigenvectors: $q_1, \ldots, q_m$.

The *Rayleigh quotient* of a vector $x \in \mathbb{R}^m$ is the scalar

$$r(x) = \frac{x^T A x}{x^T x}. \tag{27.1}$$

We can think of the Rayleigh quotient as a continuous function over x.
It turns out (after analysis in Bau)

Thus the Rayleigh quotient is a *quadratically accurate* estimate of an eigenvalue. Herein lies its power.

This idea inspires the following algorithm
POWER ITERATION:

Algorithm 27.1. Power Iteration

$v^{(0)}$ = some vector with $\|v^{(0)}\| = 1$

for $k = 1, 2, \ldots$

$w = Av^{(k-1)}$	apply A
$v^{(k)} = w/\|w\|$	normalize
$\lambda^{(k)} = (v^{(k)})^T A v^{(k)}$	Rayleigh quotient

One way we can think of this is from a theorem that as $n \to \infty$, $A^n x$ approaches to the eigenvector with associated eigenvalue of largest magnitude. Applying the Raylaigh quotient retieves that eigenvalue.

On its own, power iteration is of limited use, for several reasons. First, it can find only the eigenvector corresponding to the largest eigenvalue. Second, the convergence is linear, reducing the error only by a constant factor $\approx |\lambda_2/\lambda_1|$ at each iteration. Finally, the quality of this factor depends on having a largest eigenvalue that is significantly larger than the others. If the largest two eigenvalues are close in magnitude, the convergence will be very slow.

Fortunately, there is a way to amplify the differences between eigenvalues.

This brings us to...
INVERSE ITERATION:

Inverse Iteration

For any $\mu \in \mathbb{R}$ that is not an eigenvalue of A, the eigenvectors of $(A - \mu I)^{-1}$ are the same as the eigenvectors of A, and the corresponding eigenvalues are $\{(\lambda_j - \mu)^{-1}\}$, where $\{\lambda_j\}$ are the eigenvalues of A. This suggests an idea. Suppose μ is close to an eigenvalue λ_J of A. Then $(\lambda_J - \mu)^{-1}$ may be much larger than $(\lambda_j - \mu)^{-1}$ for all $j \neq J$. Thus, if we apply power iteration to $(A - \mu I)^{-1}$, the process will converge rapidly to q_J. This idea is called *inverse iteration*.

Algorithm 27.2. Inverse Iteration

$v^{(0)}$ = some vector with $\|v^{(0)}\| = 1$

for $k = 1, 2, \ldots$

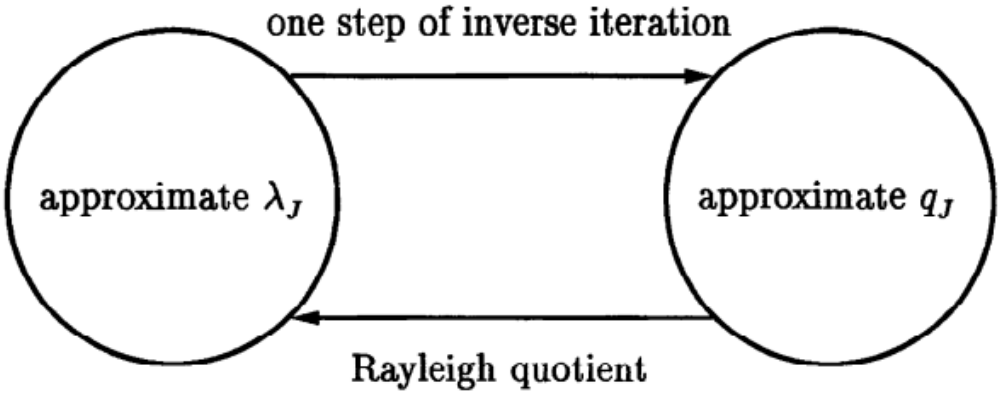
Solve $(A - \mu I)w = v^{(k-1)}$ for w	apply $(A - \mu I)^{-1}$
$v^{(k)} = w/\|w\|$	normalize
$\lambda^{(k)} = (v^{(k)})^T A v^{(k)}$	Rayleigh quotient

If we select μ smartly, with μ being closer to our desired eigenvalue that the other, we can expect rapid convergence.
It turns out that if μ is an eigenvalue, the illconditionedness of $(A - \mu I)^{-1}$ is not a problem.

Inverse iteration is one of the most valuable tools of numerical linear algebra, for it is the standard method of calculating one or more eigenvectors of a matrix if the eigenvalues are already known. In this case Algorithm 27.2 is applied as written, except that the calculation of the Rayleigh quotient is dispensed with.

RAYLEIGH QUOTIENT ITERATION:

So far in this lecture, we have presented one method for obtaining an eigenvalue estimate from an eigenvector estimate (the Rayleigh quotient), and another method for obtaining an eigenvector estimate from an eigenvalue estimate (inverse iteration). The possibility of combining these ideas is irresistible:



What is the idea? When we apply A to v in power iteration, we are getting a better eigenvalue estimate. With a better eigenvalue estimate, we do the inverse in inverse iteration to get a better eigenvector estimate. Boom boom pow we can combine the two.

Algorithm 27.3. Rayleigh Quotient Iteration

$v^{(0)}$ = some vector with $\|v^{(0)}\| = 1$
$\lambda^{(0)} = (v^{(0)})^T A v^{(0)}$ = corresponding Rayleigh quotient
for $k = 1, 2, \ldots$

Solve $(A - \lambda^{(k-1)} I) w = v^{(k-1)}$ for w	apply $(A - \lambda^{(k-1)} I)^{-1}$
$v^{(k)} = w / \|w\|$	normalize
$\lambda^{(k)} = (v^{(k)})^T A v^{(k)}$	Rayleigh quotient

The convergence of this algorithm is spectacular: each iteration triples the number of digits of accuracy.

Theorem 27.3. *Rayleigh quotient iteration converges to an eigenvalue/eigenvector pair for all except a set of measure zero of starting vectors $v^{(0)}$. When it converges, the convergence is ultimately cubic in the sense that if λ_J is an eigenvalue of A and $v^{(0)}$ is sufficiently close to the eigenvector q_J, then*

$$\|v^{(k+1)} - (\pm q_J)\| = O(\|v^{(k)} - (\pm q_J)\|^3) \tag{27.6}$$

and

$$|\lambda^{(k+1)} - \lambda_J| = O(|\lambda^{(k)} - \lambda_J|^3) \tag{27.7}$$

as $k \to \infty$. The $\pm$ signs are not necessarily the same on the two sides of (27.6).

Cubic convergence is FUCKING AWESOME

Operation Counts

We close this lecture with a note on the amount of work required to execute each step of the three iterations we have described.

First, suppose $A \in \mathbb{R}^{m \times m}$ is a full matrix. Then each step of power iteration involves a matrix-vector multiplication, requiring $O(m^2)$ flops. Each step of inverse iteration involves the solution of a linear system, which might seem to require $O(m^3)$ flops, but this figure reduces to $O(m^2)$ if the matrix is processed in advance by LU or QR factorization or another method. In the case of Rayleigh quotient iteration, the matrix to be inverted changes at each step, and beating $O(m^3)$ flops per step is not so straightforward.

These figures improve greatly if A is tridiagonal. Now, all three iterations require just $O(m)$ flops per step. For the analogous iterations involving nonsymmetric matrices, incidentally, we must deal with Hessenberg instead of tridiagonal structure, and this figure increases to $O(m^2)$.

QR Algorithm without Shifts

The most basic form of the QR algorithm is freakishly simple.

Algorithm 28.1. "Pure" QR Algorithm

$A^{(0)} = A$

for $k = 1, 2, \ldots$

$Q^{(k)}R^{(k)} = A^{(k-1)}$	QR factorization of $A^{(k-1)}$
$A^{(k)} = R^{(k)}Q^{(k)}$	Recombine factors in reverse order

We can get that sweet sweet cubic convergence with this simple little algorithm, but only when we make it a little less simple.

1. Before starting the iteration, A is reduced to tridiagonal form, as discussed in Lecture 26.
2. Instead of $A^{(k)}$, a shifted matrix $A^{(k)} - \mu^{(k)}I$ is factored at each step, where $\mu^{(k)}$ is some eigenvalue estimate.
3. Whenever possible, and in particular whenever an eigenvalue is found, the problem is "deflated" by breaking $A^{(k)}$ into submatrices.

A QR algorithm incorporating these modifications has the following outline.

Algorithm 28.2. "Practical" QR Algorithm

$(Q^{(0)})^T A^{(0)} Q^{(0)} = A$ $\quad$ $A^{(0)}$ is a tridiagonalization of A

for $k = 1, 2, \ldots$

Pick a shift $\mu^{(k)}$	e.g., choose $\mu^{(k)} = A_{mm}^{(k-1)}$
$Q^{(k)}R^{(k)} = A^{(k-1)} - \mu^{(k)}I$	QR factorization of $A^{(k-1)} - \mu^{(k)}I$
$A^{(k)} = R^{(k)}Q^{(k)} + \mu^{(k)}I$	Recombine factors in reverse order

If any off-diagonal element $A_{j,j+1}^{(k)}$ is sufficiently close to zero,

set $A_{j,j+1} = A_{j+1,j} = 0$ to obtain

$$\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} = A^{(k)}$$

and now apply the QR algorithm to A_1 and A_2.

Algorithm 28.1 converges to a Schur form for A (eigenvalue revealing facorization). We can get an intuition for this by looking at the final product in each iteration.

$$A^{(k)} = R^{(k)}Q^{(k)} = (Q^{(k)})^T A^{(k-1)} Q^{(k)}$$

Now this looks like a Rayleigh quotient matrix, so it makes sense that these iterations are

eigenvalue revealing. To make greater sense of what is occurring in the QR algorithm, let us first consider another method, simultaneous iteration.

SIMULTANEOUS ITERATION:
Simultaneous iteration builds on the idea of power iteration. Instead of starting with 1 vector, I want to start with n vectors and obtain n eigenvectors. The main idea is that if I have have n linearly independent vectors $v_1, \ldots, v_n$, then the space of $\langle A^{(k)}v_1, \ldots, A^{(k)}v_n \rangle$ should converge to the space $\langle q_1, \ldots, q_n \rangle$ spanned by the eigenvectors corresponding to the n largest eigenvalues. Greater mathematical justification of this can be found in the chapter.

As $k \to \infty$, the vectors $v_1^{(k)}, \ldots, v_n^{(k)}$ in the algorithm (28.1)–(28.3) all converge to multiples of the same dominant eigenvector q_1 of A. Thus, although the space they span, $\langle v_1^{(k)}, \ldots, v_j^{(k)} \rangle$, converges to something useful, these vectors constitute a highly ill-conditioned basis of that space. If we actually carried out simultaneous iteration in floating point arithmetic as just described, the desired information would quickly be lost to rounding errors.

The remedy is simple: one must orthonormalize at each step rather than once and for all. Thus we shall not construct $V^{(k)}$ as defined above, but a different sequence of matrices $Z^{(k)}$ with the same column spaces.

Algorithm 28.3. Simultaneous Iteration

Pick $\hat{Q}^{(0)} \in \mathbb{R}^{m \times n}$ with orthonormal columns.
for $k = 1, 2, \ldots$
$\quad Z = A\hat{Q}^{(k-1)}$
$\quad \hat{Q}^{(k)}\hat{R}^{(k)} = Z$ reduced QR factorization of Z

Some nice math tells us that not only does the column space of $\hat{Q}^{(k)}$ span the space of our n eigenvectors, but the columns actually converge to those eigenvectors. Makes sense.

NOW, we can understand the QR algorithm. Turns out, the QR algorithm is equivalent to simultaneous iteration applied to m initial vectors.

Here are the three formulas that define simultaneous iteration with $\underline{Q}^{(0)} = I$, followed by a fourth formula that we shall take as a definition of an $m \times m$ matrix $A^{(k)}$:

Simultaneous Iteration:

$$\underline{Q}^{(0)} = I, \tag{28.7}$$

$$Z = A\underline{Q}^{(k-1)}, \tag{28.8}$$

$$Z = \underline{Q}^{(k)} R^{(k)}, \tag{28.9}$$

$$A^{(k)} = (\underline{Q}^{(k)})^T A \underline{Q}^{(k)}. \tag{28.10}$$

And here are the three formulas that define the pure QR algorithm, followed by a fourth formula that we shall take as a definition of an $m \times m$ matrix $\underline{Q}^{(k)}$:

Unshifted QR Algorithm:

$$A^{(0)} = A, \tag{28.11}$$

$$A^{(k-1)} = Q^{(k)} R^{(k)}, \tag{28.12}$$

$$A^{(k)} = R^{(k)} Q^{(k)}, \tag{28.13}$$

$$\underline{Q}^{(k)} = Q^{(1)} Q^{(2)} \cdots Q^{(k)}. \tag{28.14}$$

Additionally, for both algorithms, let us define one further $m \times m$ matrix $\underline{R}^{(k)}$,

$$\underline{R}^{(k)} = R^{(k)} R^{(k-1)} \cdots R^{(1)}. \tag{28.15}$$

We can now exhibit the equivalence of these two algorithms.

Theorem 28.3. *The processes* (28.7)–(28.10) *and* (28.11)–(28.14) *generate identical sequences of matrices* $\underline{R}^{(k)}$, $\underline{Q}^{(k)}$, *and* $A^{(k)}$, *namely, those defined by the QR factorization of the* k*th power of* A,

$$A^k = \underline{Q}^{(k)} \underline{R}^{(k)}, \tag{28.16}$$

together with the projection

$$A^{(k)} = (\underline{Q}^{(k)})^T A \underline{Q}^{(k)}. \tag{28.17}$$

pf. A pretty simple induction. Can be found in the textbook.

Convergence:

QR Algorithm with Shifts

The main idea is that the same reasons that make shifting in inverse iteration converge so fast with good eigenvalue estimates applies here.

Other eigenvalue algorithms

Iterative Methods

Overview of Iterative Methods

Direct matrix algorithms where we cannot take advantage or sparsity are bottlenecks at $O(m^3)$. If we could somehow do this in $O(m^2)$, then we could solve problems with much larger matrices.

BIG IDEA
When we are doing a task with extremely large matrices, we aren't dilly dallying with random matrices. Perhaps we are working with the discretization of a differential or integral equation. In practice, large matrices are sparse, and we can abuse this sparsity. If we can limit our interaction with our large matrix to matrix multiplication, we never have to deal with the pain of a huge matrix, and instead can take advantage of the sparsity for $O(m)$ matrix multiplication.

Jacobi Iteration

Jacobi iteration is a classical iterative method for solving $Ax = b$.
Let x^* be the solution such that $Ax^* = b$. With Jacobi iteration, we want to create a sequence of iterates $x_1, \ldots, x_n$ that converge to x^*.
Idea: Split up A into easier sub components to work with i.e THE DIAGONAL.

$$A = A - D + D$$

Now, we are solving

$$(A - D + D)x = b$$

Define our iterates as follows:

$$(A - D)x^n + Dx^{n+1} = b$$

The idea is that the iterates should converge to the fixed point x^*, thus, our update is:

$$x^{(n+1)} = D^{-1}(b - (A - D)x^{(n)}) = D^{-1}b - D^{-1}(A - D)x^{(n)}$$

Analyzing convergence.
Lets $M = D^{-1}(A - D)$ be called our iteration matrix.
Trivially, we have

$$x^{(n+1)} = D^{-1}b - Mx^{(n)}$$

$$x^* = D^{-1}b - Mx^*$$

Defining our error $e^n = x^n - x^*$

$$x^{(n+1)} - x* = -M(x^{(n)} - x^*) \implies e^{(n+1)} = (-1)^n M^n e^{(0)}$$

To guarantee convergence, we must have M^n send all vectors to 0 as $n \to \infty$.
Thus, a necessary and sufficient condition for guaranteeing the convergence of Jacobi iteration is that the spectrum of M, $\rho(M) < 1$ where

$$\rho(M) = \max_{\lambda \in \Lambda(M)} |\lambda|$$

THM: If A is strictly diagonally dominant, then Jacobi iteration is guaranteed to converge.
pf. Easy to show that this causes the iteration matrix to have spectrum < 1.

THM: If A is 2x2, symmetric, positive definite, then Jacobi iteration converges.

To solve a system to a tolerance ϵ, Jacobi iteration needs $log(\frac{1}{\epsilon})\kappa_2(A)$ iteration
Per iteration cost is ~ n^2

Gauss-Seidel Iteration

The main idea behind Jacobi iteration is that the inverse matrix we calculate at each iteration must be easy. Since we only use the inverse of a diagonal matrix in Jacobi, it is easily computed by constructing a diagonal matrix with the inverse of the diagonal elements. In earlier chapters, we saw that is it easy to solve a system $Lx = b$ where L is lower triangular using forward substitution (similarly, we can use backward substitution for upper triangular matrix). Gauss-Seidel uses this idea.

Consider the system $Ax = b$
We can split up A as follows

$$A = L + R$$

where L is lower triangular and R is strictly upper triangular.
Like Jacobi iteration, we define our sequence of iterates as follows:

$$Lx^{(n+1)} = b - Rx^{(n)} \implies x^{(n+1)} = L^{-1}(b - Rx^{(n)})$$

The same error analysis shows us:

$$e^{(n)} = (-1)^n (L^{-1}R)^n e^{(0)}$$

Thus, Gauss-Seidel has the same convergence criteria where

$$\rho(L^{-1}R) < 1$$

THM: Gauss-Seidel converges on strictly diagonally dominant matrices.
pf. Bound eigenvalues with gershgorin circle theorem.

THM: Gauss-Seidel converges for symmetric, positive definite matrices.

Understanding Jacobi & Gauss-Seidel through the lens of optimization

Suppose A is symmetric, positive definite
We can view solving the system $Ax = b$ as the following optimization problem

$$\min_x f(x) = \frac{1}{2}x^T Ax - b^T x$$

where

$$\nabla f(x) = Ax - b$$

For Jacobi iteration, we can rewrite our update as:

$$x^{(n+1)} = x^{(n)} - D^{-1}(Ax^{(n)} - b)$$

This is precisely gradient descent with preconditioner D^{-1}:

$$x^{(n+1)} = x^{(n)} - D^{-1}\nabla f(x^{(n)})$$

For Gauss-Seidel, at iteration k+1, we update each coordinate i sequentially:

$$x_i^{(k+1)} = \arg\min_{x_i} f(x_1^{(k+1)}, \ldots, x_{i-1}^{(k+1)}, x_i, x_{i+1}^{(k)}, \ldots, x_n^{(k)})$$

This is exactly coordinate descent on our quadratic objective function. The update can be computed in closed form since:

$$\frac{\partial f}{\partial x_i} = \sum_{j=1}^{n} a_{ij}x_j - b_i$$

Setting this to zero and solving for x_i gives us:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}x_j^{(k+1)} - \sum_{j>i} a_{ij}x_j^{(k)}\right)$$

This optimization perspective provides several insights:

1. Both methods are minimizing the same quadratic objective function
2. Gauss-Seidel/coordinate descent often converges faster because it uses the most recent updates
3. The convergence rate depends on the conditioning of A, just as in optimization
4. The connection to optimization allows us to apply optimization theory to understand convergence properties

Krylov Subspace Methods

Here, we move on from classical iterative methods into the family of iterative methods known as Krylov Subspace Methods.

Arnoldi Iteration

The Arnoldi Iteration is a Gram-Schmidt style iteration for transforming a matrix to Hessenburg form. Why? Consider the analogy of QR factorization. We saw Householder reflectors which triangularize A through a series of orthogonal transformations, and Gram-Schmidt orthogonalization which orthogonalizes A through a series of triangular operations. The thing about Gram-Schmidt is that it can be stopped early to produce a reduce QR factorization. To construct a Hessenburg, we last saw the use of Householder reflectors operating on both sides of the matrix (similarity transformation). Thus, the Arnoldi iteration is the analogue of the Gram-Schmidt process for similarity transformations to Hessenburg form. Like Gram-Schmidt, we can stop early, leading to a partial reduction to Hessenburg form, which can be very useful.

$A \in \mathbb{R}^{m \times m}$, n < m. From Trefethen and Bau

A complete reduction of A to Hessenberg form by an orthogonal similarity transformation might be written $A = QHQ^*$, or $AQ = QH$. However, in dealing with iterative methods we take the view that m is huge or infinite, so that computing the full reduction is out of the question. Instead we consider the first n columns of $AQ = QH$. Let Q_n be the $m \times n$ matrix whose columns are the first n columns of Q:

$$Q_n = \left[\begin{array}{c|c|c|c} & & & \\ q_1 & q_2 & \cdots & q_n \\ & & & \end{array}\right]. \tag{33.1}$$

Here and in the lectures ahead, it would be consistent with our usage elsewhere in the book to put hats on the symbols Q_n, since these matrices are rectangular, but to keep the formulas uncluttered we do not do this.

Let $\tilde{H}_n$ be the $(n+1)\times n$ upper-left section of H, which is also a Hessenberg matrix:

$$\tilde{H}_n = \begin{bmatrix} h_{11} & \cdots & h_{1n} \\ h_{21} & h_{22} & \\ & \ddots & \ddots & \vdots \\ & & h_{n,n-1} & h_{nn} \\ & & & h_{n+1,n} \end{bmatrix}. \tag{33.2}$$

Then we have

$$AQ_n = Q_{n+1}\tilde{H}_n, \tag{33.3}$$

that is,

$$\begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix} \begin{bmatrix} q_1 & \cdots & q_n \end{bmatrix} = \begin{bmatrix} q_1 & \cdots & q_{n+1} \end{bmatrix} \begin{bmatrix} h_{11} & \cdots & h_{1n} \\ h_{21} & & \vdots \\ & \ddots & \\ & & h_{n+1,n} \end{bmatrix}.$$

The nth column of this equation can be written as follows:

$$Aq_n = h_{1n}q_1 + \cdots + h_{nn}q_n + h_{n+1,n}q_{n+1}. \tag{33.4}$$

In words, q_{n+1} satisfies an $(n+1)$-term recurrence relation involving itself and the previous Krylov vectors.

The Arnoldi iteration is simply the modified Gram–Schmidt iteration that implements (33.4). The following algorithm should be compared with Algorithm 8.1.

Algorithm 33.1. Arnoldi Iteration

$b =$ arbitrary, $q_1 = b/\|b\|$
for $n = 1, 2, 3, \ldots$
    $v = Aq_n$
    for $j = 1$ **to** n
        $h_{jn} = q_j^* v$
        $v = v - h_{jn} q_j$
    $h_{n+1,n} = \|v\|$ [see Exercise 33.2 concerning $h_{n+1,n} = 0$]
    $q_{n+1} = v/h_{n+1,n}$

GMRES

Generalized Minimal Residuals

Conjugate Gradients

Conjugate Gradients is an algorithm designed to minimize the A norm of the error as we expand our Krylov subspace.

Algorithm:
$x_0 = 0$, $r_0 = b$, $p_0 = r_0 = b$
At iteration k:
$\alpha_k = \frac{\langle r_{k-1}, r_{k-1} \rangle}{\langle p_{k-1}, Ap_{k-1} \rangle}$
$x_k = x_{k-1} + \alpha_k p_{k-1}$
$r_k = b - Ax_k = r_{k-1} - \alpha_k A p_{k-1}$
$\beta_k = \frac{\langle r_k, r_k \rangle}{\langle r_{k-1}, r_{k-1} \rangle}$
$p_k = r_k + b_k p_{k-1}$

So what is going on?

Important facts:

- Residuals r are orthogonal
- Search directions p are A orthogonal
- At each iteration k, CG minimizes $|e_k|_A$ where $x_k \in \mathcal{K}_k(A, b)$
- $\frac{|e_n|_A}{|e_0|_A} = (\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1})^n$

Fast Fourier Transform