CS 6210 notes

Shawn Ong

January 2022

1 Matrix Factorizations

(Reduced) SVD of $A \in \mathbb{R}^{m\times n}$ is $A = U\Sigma V^T = \sum_{i=1}^{\ell} \sigma_i u_i v_i^T$ where (let $\ell = \min(m,n)$):

- $U \in \mathbb{R}^{m\times \ell}$ with $U^TU = I$ and cols of U are left singular vectors
- $V \in \mathbb{R}^{n\times \ell}$ with $V^TV = I$ and cols of V are right singular vectors
- $\Sigma \in \mathbb{R}^{\ell\times\ell}$ diagonal $\begin{pmatrix}\sigma_1 & & \\ & \ddots & \\ & & \sigma_\ell\end{pmatrix}$ with $\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_\ell \geqslant 0$.

Key facts about SVD:

- Always exists
- $\|A\|_2 = \sigma_1$
- $\|A\|_F = (\sum_i \sigma_i^2)^{1/2}$
- $\operatorname{rank}(A) = \#\sigma_i > 0$
- $\varepsilon\text{-}\operatorname{rank}(A) = \#\sigma_i > \epsilon\sigma_1$
- SVD yields "best approximations" i.e. for any $i \leqslant k < \ell$, $A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$ is the best rank-k approximation of A in $\|\cdot\|_2$ or $\|\cdot\|_F$.

2 More Related Factorizations:

Often square $A \in \mathbb{R}^{n\times n}$ can be written as $A = X\Lambda X^{-1}$ where X invertible and Λ diagonal. If so, A is diagonalizable; columns of X are eigenvectors and entries of Λ are eigenvalues:

$$AX = X\Lambda$$

Diagonalizable matrices dense, but we won't work with JCF because not nice numerically - not continuous in entries.

If $A = A^T$, then A can always be written as $A = V\Lambda V^T$ with V orthogonal ($V^TV = I$) and Λ diagonal, real-valued.

For any square matrix A, Schur form is $A = UTU^*$ where $U^*U = I$, T is upper triangular and contains A's eigenvalues on its diagonal. We often use it to reason about convergence. For real A, real Schur form always exists – allow T to be block upper triangular; then version with all real entries exists.

2 Floating point

We primarily work with real numbers, but in practice can only store finite info.

Solution: Floating point representations (essentially scientific notation). Issues:

(1) There are gaps between numbers we can represent.

(2) There exist a largest and smallest (nonzero) number we can store.

In this course, we largely focus on (1) and ignore (2). (2) is so-called overflow/underflow; but for double precision floating point numbers, we can store approx 10^{-308} to 10^{308} so generally not a huge issue. Avoid calculating determinants – could cause issues.

Idea of floating point: gaps between numbers are constant in a relative sense, i.e.:

$$1, 1 + 2^{-52}, 1 + 2 \cdot 2^{-52}, \ldots, 2$$
$$2, 2 + 2^{-51}, 2 + 2 \cdot 2^{-51}, \ldots, 4$$

Each interval $[2^j, 2^{j+1}]$ has the same amount of numbers in it.

2.1 Idealized floating point:

Discrete subset $F \subseteq \mathbb{R}$ defined by integer $\beta \geqslant 2$ (base) and by integer $t \geqslant 1$ (precision). Elements of F are:

$$\{0\} \cup \pm \left(\frac{m}{\beta^t} \right) \beta^e$$

$\beta^{t-1} \leqslant m \leqslant \beta^t$ is called mantissa, e the exponent. In the idealized system, e is unbounded (can't be implemented in practice).

In practice, $\beta = 2$ and for double precision, $t = 53$. Also add bounds on exponent.

Double precision number takes 64 bits to store:

$$\underbrace{\boxed{1 \text{ bit}}}_{\text{sign}} \underbrace{\boxed{11 \text{ bits}}}_{\text{exponent}} \underbrace{\boxed{52 \text{ bits}}}_{\text{mantissa}}$$

Numbers are stored as:

$$\pm \left(1 + \sum_{i=1}^{52} m_i 2^{-i} \right) 2^e$$

where $m_i \in \{0, 1\}$, $-1022 \leqslant e \leqslant 1023$ and the extra 1 guarantees unique representation. Note that $e = -1023, 1024$ are not used – they are reserved to encode special cases. Additionally, we store 52 bits of mantissa, but the value $t = 53$ (includes the additional guaranteed 1).

Key information: resolution of F is summarized by machine precision, defined as:

$$\mu = \frac{1}{2} \beta^{1-t}$$

i.e., half the distance between 1 and the next biggest number. E.g., for double precision, $\mu \approx 10^{-16}$.

Let $fl(x)$ for $x \in \mathbb{R}$ to be the closest floating point to x (round to nearest representation). Then $\forall x \in \mathbb{R}, \exists \varepsilon$ with $|\varepsilon| \leqslant \mu$ such that $fl(x) = x(1+\varepsilon)$.

2.2 Arithmetic with floating point numbers:

We use $+, -, \times, \div$ to represent "real arithmetic." Let the circled versions of those same symbols $\oplus, \ominus, \otimes, \oslash$ represent "floating point arithmetic."

Axioms:

$$x \circledast y = fl(x * y)$$

In particular, for all $x, y \in F$, there exists an ε with $|\varepsilon| \leqslant \mu$ such that:

$$x \circledast y = (x * y)(1+\varepsilon)$$

2.3 Conditioning

Conditioning – property of problems. Accuracy/stability – property of algorithms.

Informally, a problem is a function $f : X \to Y$, where X is called the data/input space and Y is called the output space (assume both normed). We typically study f at a specific $x \in X$, i.e. a problem instance (typically implicit). A well-conditioned problem is one where *any* small change in x induces a small change in $f(x)$. Conversely, an ill-conditioned problem is one where some small change in x induces a large change in $f(x)$.

We define the relative condition number of a problem f as follows:

$$\kappa = \lim_{\delta \to 0} \sup_{\|\delta x\| \leqslant \delta} \left(\frac{\|\delta f\| / \|f(x)\|}{\|\delta x\| / \|x\|} \right)$$

where $\delta f = f(x + \delta x) - f(x)$. With some appropriate assumptions, this can be rewritten in terms of the Jacobian as:

$$\frac{\|J(x)\|}{\|f(x)\| / \|x\|}$$

Example: Subtraction $f(x) = x_1 - x_2$, using $\|\cdot\|_\infty$. Here, $J = \begin{bmatrix} 1 & -1 \end{bmatrix}$.

$$\kappa = \frac{2}{|x_1 - x_2| / \max\{x_1, x_2\}}$$

As $x_1 \to x_2$, becomes increasingly ill-conditioned (as long as not both going to 0). Catastrophic cancellation: don't try to calculate small numbers by subtracting 2 big ones.

Example: Solving linear systems $Ax = b$. As a function, $A, b \mapsto x = A^{-1}b$ (assuming A invertible).

Define the condition number of A as:

$$\kappa_2(A) = \|A\|_2\|A^{-1}\|_2 = \frac{\sigma_1}{\sigma_n} \qquad , A \in \mathbb{R}^{n\times n}$$

Why? (consider only perturbations to A)

$$\begin{aligned} (A+\delta A)(x+\delta x) &= b \\ (\delta A)x + A(\delta x) &= 0 \qquad \text{(if we ignore small term } \delta A(\delta x)) \\ \implies \delta x &= -A^{-1}(\delta A)x \\ \|\delta x\| &\leqslant \|A^{-1}\| \cdot \|\delta A\| \cdot \|x\| \\ \implies \frac{\|\delta x\|/\|x\|}{\|\delta A\|/\|A\|} &\leqslant \|A^{-1}\|\|A\| = \kappa_2(A) \end{aligned}$$

2.4 Accuracy and Stability

Recall: a problem is a function $f : X \to Y$ (assumed to be between normed vector spaces). Today, we consider algorithms the same way. An algorithm for f is a similar map $\tilde{f} : X \to Y$. $\tilde{f}$ takes the "true" x as input; so includes $x \mapsto fl(x)$. General idea: $\tilde{f}$ represents computation we can "actually do" – this includes its output being something we can express on computer.

An algorithm $\tilde{f}$ is <u>accurate</u> if for every $x \in X$:

$$\phi(\mu) = \frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} = O(\mu) \qquad \text{(as } \mu \to 0)$$

Intuitively, algorithm gets more precise as you increase machine precision. This particular condition implies that $\exists c > 0$ s.t. for all sufficiently small μ, $\phi(\mu) \leqslant c\mu$. But actually, ϕ is a function of both x, μ. So $\phi(\mu) = O(\mu)$ must be true uniformly in x, i.e., the constant c doesn't depend on x. But accuracy can be too restrictive – may not always been the best goal.

An algorithm $\tilde{f}$ is <u>stable</u> if for all $x \in X$,

$$\frac{\|\tilde{f}(x) - f(\tilde{x})\|}{\|f(\tilde{x})\|} = O(\mu)$$

for some $\tilde{x}$ with $\frac{\|x-\tilde{x}\|}{\|x\|} = O(\mu)$. Intuitively: gets "nearly" the correct answer to "nearly" the correct question.

There is also a slightly stronger version: an algorithm $\tilde{f}$ is <u>backwards stable</u> if for all $x \in X$,

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

for some $\tilde{x}$ with $\frac{\|x-\tilde{x}\|}{\|x\|} = O(\mu)$.

Example: our axiom for floating point arithmetic operations $\oplus, \ominus, \otimes, \oslash$ implies that these operations are backwards stable.

Theorem: (backwards stability $\Rightarrow$ "accuracy," in the form of relative error guarantees). If $\tilde{f}$ is a backwards stable algorithm for f with condition number κ, then:

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} = O(\kappa(x)\mu)$$

Colloquially, we "lose" $\kappa(x)$ accuracy. This is so-called backwards error analysis. Conversely, we can perform forward error analysis; roughly, "track floating point errors through all operations."

3 Algorithms

First problem: Solving $Ax = b$ for nonsingular A. Our two main approaches:

- Direct methods: Finite sequence of steps yields a solution.
- Iterative methods: $x^{(k)} \to x$ as $k \to \infty$.

We start with direct methods, but will spend more time on iterative methods eventually.

Our first method has 2 steps:

1. "factor" A into the product of matrices $M_1 M_2$ (could be more than 2 matrices). where the M_i are "easy" to solve $M_i y = d$.
2. Split the problem: $Ax = b \implies M_1 M_2 x = b$. Solve $M_1 y = b$ for y and $M_2 x = y$ for x.

What kind of matrices M are easy to solve linear systems with?

(1) M is diagonal

(2) M is upper/lower triangular. Solve via forward substitution (costs $O(n^2)$ arithmetic operations).

(3) Orthogonal matrices Q; since $Qx = b \implies x = Q^T b$.

First algorithm is triangular factorizations. Gaussian elimination:

$$A = LU$$

We accomplish this by constructing a sequence of lower triangular matrices $L_1, \ldots, L_{n-1}$ such that:

$$L_{n-1} \ldots L_2 L_1 A = U$$

Since the product of lower triangular matrices is lower triangular, we have $L^{-1}A = U$; the inverse of lower triangular nonsingular matrix is also lower triangular. This gives $A = LU$. Luckily, the process by which we generate the L_i makes inversion simple. Each L_i will make A upper triangular one column at a time. L_1 makes the first column of A upper triangular:

$$\begin{pmatrix} x & x & \ldots & \ldots & x \\ x & \ddots & & & \vdots \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ x & \ldots & \ldots & \ldots & x \end{pmatrix} \xrightarrow{L_1} \begin{pmatrix} x & x & \ldots & \ldots & x \\ 0 & x & \ldots & & \vdots \\ \vdots & \vdots & \ddots & & \vdots \\ \vdots & & & & \vdots \\ 0 & \ldots & \ldots & \ldots & x \end{pmatrix} \xrightarrow{L_2} \ldots \xrightarrow{L_{n-1}} \begin{pmatrix} x & x & \ldots & \ldots & x \\ 0 & x & \ldots & & \vdots \\ 0 & 0 & x & \ldots & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & \ldots & \ldots & \ldots & x \end{pmatrix}$$

We need L_i to be "easy" to compute and invert, and want L_k to be such that for some vector

$z_k = \begin{pmatrix} z_{1k} \\ \vdots \\ z_{nk} \end{pmatrix}$:

$$L_k z_k = \begin{pmatrix} z_{1k} \\ \vdots \\ z_{kk} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Let $\ell_{j,k} = \frac{z_{jk}}{z_{kk}}$ for $j > k$. We build our matrix by defining $\ell_k = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \ell_{k+1,k} \\ \vdots \\ \ell_{n,k} \end{pmatrix}$. This allows us to define:

$$L_k = \begin{pmatrix} 1 & 0 & \dots & \dots & \dots & 0 \\ -\ell_{2,1} & \ddots & & & & \vdots \\ \vdots & & 1 & & & \vdots \\ \vdots & & -\ell_{k+1,k} & \ddots & & \vdots \\ \vdots & & \vdots & & & 0 \\ -\ell_{n,1} & \dots & -\ell_{n,k} & \dots & \dots & 1 \end{pmatrix}$$

We can also write this more concisely as:

$$L_k = I - \ell_k e_k^T$$

Note that if we can correctly identify z_k, this matrix easy to compute. Even better, this is easy to invert:

$$L_k^{-1} = I + \ell_k e_k^T$$

Additionally, we don't care about calculating individual L_i. We seek $L = L_1^{-1} L_2^{-1} \dots L_{n-1}^{-1}$:

$$L_1^{-1} L_2^{-1} \dots L_{n-1}^{-1} = I + \sum_{k=1}^{n-2} \ell_k e_k^T$$

Where additional terms drop out because the $\ell_n e_k = 0$ for $n > k$. This ends up yielding:

$$L = \begin{pmatrix} 1 & 0 & \dots & 0 \\ \ell_{2,1} & 1 & & \vdots \\ \vdots & & \ddots & \vdots \\ \ell_{n,1} & \dots & \ell_{n,n-1} & 1 \end{pmatrix}$$

With this, we can now provide an algorithm that computes LU factorization, assuming one exists.

Algorithm 1 Gaussian Elimination

Set $L = I, U = A$.
for $k = 1, \ldots, n-1$ **do**
  $L(k+1:n, k) = U(k+1:n, k)/U(k, k)$ ▷ Compute L
  $U(k+1:n, k:n) = U(k+1:n, k:n) - L(k+1:n, k)U(k, k:n)$ ▷ $L_k = I - \ell_k e_k^T$
end for
Return L, U

Gaussian Elimination/LU without pivoting

Analyze the algorithm: How efficient is it (in terms of floating-point operations) and how stable is it?

- Takes $O(n^3)$ floating point operations ($\sim \frac{2}{3}n^3$).
- Stability: This doesn't work for matrices, e.g. $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$. Problem persists even if not dividing by 0.

Example Consider $A = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix}$. In exact arithmetic, we have:

$$L = \begin{pmatrix} 1 & 0 \\ 10^{20} & 1 \end{pmatrix} \qquad U = \begin{pmatrix} 10^{-20} & 1 \\ 0 & 1 - 10^{20} \end{pmatrix}$$

Say that in floating point, $1 - 10^{20}$ rounds to -10^{20} (since precision is on the order of 10^{-16}).

$$\tilde{L} = \begin{pmatrix} 1 & 0 \\ 10^{20} & 1 \end{pmatrix} \qquad \tilde{U} = \begin{pmatrix} 10^{-20} & 1 \\ 0 & -10^{20} \end{pmatrix}$$

But $\tilde{L}\tilde{U} = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 0 \end{pmatrix}$. So stability issues here; this can also affect things like solving linear system with $\tilde{L}, \tilde{U}$.

We'll ask for better properties on L, U (by convention, these are imposed on only L):

1. Force L to have unit diagonal.
2. Force $|\ell_{ij}| \leqslant 1$.

Recall $\ell_{jk} = \frac{z_{jk}}{z_{kk}}$ – we have issues when z_{jk} is large relative to z_{kk}. To fix, we observe that we can permute rows of A without affecting the method, but changing the resulting numbers. To account for that, before we apply each L_i, permute the rows so that the largest entry appears on the "top" row. This so-called "partial pivoting" ensures that every entry of L_i is $\leqslant 1$.

Note that this procedure computes:

$$L_{n-1}P_{n-1} \ldots L_2 P_2 L_1 P_1 A = U$$

It is not immediately obvious that this collapses nicely into $L^{-1}P$. Then $PA = LU$ (permuted/pivoted LU decomposition).

Suppose we have $L_3P_3L_2P_2L_1P_1A = U$. We simplify by introducing additional permutations to collect them to the right.

$$\begin{aligned} L_3P_3L_2P_2L_1P_1A &= U \\ L_3P_3L_2P_2L_1(P_2^{-1}P_3^{-1}P_3P_2)P_1A &= U \\ L_3P_3L_2(P_3^{-1}P_3)P_2L_1(P_2^{-1}P_3^{-1}P_3P_2)P_1A &= U \\ L_3(P_3L_2P_3^{-1})(P_3P_2L_1P_2^{-1}P_3^{-1})(P_3P_2P_1)A &= U \\ L_3'L_2'L_1'PA &= U \end{aligned}$$

For each L_i', note that permutation indices P_n satisfy $n > i$. In particular, they only permute entries below the diagonal, and therefore maintain lower triangular property. Applying the inverses doesn't affect the leading columns, and exactly undoes the permutations – results in still lower triangular matrix with unit diagonal.

For every nonsingular A, there is a $PA = LU$ decomposition that satisfies our desired properties.

Thus, given non-singular $A \in \mathbb{R}^{n\times n}$, compute:

1. Permutation matrix P.
2. Lower triangular L with $\ell_{ij} \leqslant 1$, $\ell_{ii} = 1$.
3. Upper triangular U.

such that $PA = LU$.

Algorithm: LU factorization with partial pivoting (store permutation P as vector, $L + U$ "in place" – don't need to store ℓ_{ii}, as these are all 1).

1. w.r.t. column 1, swap the first row containing the row with the largest entry.

Algorithm 2 LU with Partial Pivoting

for $k = 1, \ldots, n-1$ **do**
  Find j with $k \leqslant j \leqslant n$ s.t. $|A(j,k)| = \max_{k\leqslant\ell\leqslant n} |A(\ell,k)|$
  $Piv(k) = j$ ▷ Encoding P
  $A(k,:) \leftrightarrow A(j,:)$ ▷ Swap rows
  $\eta \leftarrow k+1 : n$ ▷ Abbreviation for conciseness
  $A(\eta,k) = A(\eta,k)/A(k,k)$ ▷ "Compute" L_k
  $A(\eta,\eta) = A(\eta,\eta) - A(\eta,k)A(k,\eta)$ ▷ Apply L_k
end for

Cost is still $\sim \frac{2}{3}n^3$ flops (increased overhead in permutations, but not floating-point opera-

tions). Once finished, A has the form:

$$A = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ \ell_{21} & u_{22} & & \vdots \\ \vdots & & \ddots & \vdots \\ \ell_{n1} & \dots & \ell_{n,n-1} & u_{nn} \end{pmatrix}$$

We call this algorithm partial pivoting since we are looking only at entries of (part of) a single column when pivoting. We can do better by considering instead a submatrix, finding the largest element, then permuting both row *and* column to move the largest entry to its top left. This gives us LU with complete pivoting, and yields $PAQ = LU$ (P for row permutations, Q for column permutations).

Given A, b, how to solve $Ax = b$?

1. Compute $PA = LU$ in $O(n^3)$. Then $PAx = Pb \implies LUx = Pb$.
2. Solve $Ly = Pb$ for y in ($O(n^2)$, since L lower triangular).
3. Solve $Ux = y$ for x in $O(n^2)$.

Observe that the dominant cost is in step 1, solely dependent upon A. Suppose we fix A and change b; don't need to re-solve for A after the first time for lower cost.

Theorem 1 (Stability of backward/forward substitution.)**.** *Suppose we solve $Ux = b$ via backwards substitution for upper triangular U, with solution $\tilde{x}$. Then $\tilde{x}$ satisfies $(U+\delta U)\tilde{x} = b$ for some upper triangular δU with $\frac{\|\delta U\|}{\|U\|} = O(\mu)$, i.e. alg is backwards stable. In fact, we can even guarantee $\frac{|\delta U_{ij}|}{|U_{ij}|} \leqslant n\mu + O(\mu^2)$ (entrywise stability, incurring cost of $n\mu$).*

Theorem 2. *Given A, if A has an LU factorization $A = LU$, then for all sufficiently small μ, the algorithm completes (i.e. no zero division). The computed $\tilde{L}, \tilde{U}$ satisfy: $\tilde{L}\tilde{U} = A + \delta A$ with $\frac{\|\delta A\|}{\|L\|\|U\|} = O(\mu)$.*

For $PA = LU$ (partial pivoting), we know $\|L\| = O(1)$ so $\frac{\|\delta U\|}{\|U\|} = O(\mu)$. This ends up true for complete pivoting as well.

This motivates the <u>growth factor</u> ρ in $PA = LU$ is the following:

$$\rho = \frac{\max_{ij} |u_{ij}|}{\max_{ij} |a_{ij}|}$$

Theorem 3 (Stability of partial pivoted LU.)**.** *If $PA = LU$, with computed $\tilde{P}, \tilde{L}, \tilde{U}$, then for all sufficiently small μ, if no "ties" in true pivoting, $P = \tilde{P}$ and $\tilde{L}\tilde{U} = \tilde{P}A + \delta A$ with $\frac{\|\delta A\|}{\|A\|} = O(\rho\mu)$.*

If we can come up with a reasonable bound for ρ, this is basically backwards stability. So next question is how big can ρ be? For partial pivoting, worst case $\rho = 2^{n-1}$. Note that this technically independent of A, but not good – pathological dependence on dimension.

For complete pivoting, $\rho \leqslant \sqrt{n}\left(\prod_{\ell=2}^{n} \ell^{1/(\ell-1)}\right) \approx n^{\frac{1}{2}+\log\frac{n}{4}}$.

4 Error bounding

We use this technique to solve, for example, $Ax = b$ via $PA = LU$ to obtain some $\tilde{x} \approx A^{-1}b$. Recall that we have relative error $\frac{\|x-\tilde{x}\|_2}{\|x\|_2}$. This is a reasonable error measurement, although we do not always know x. Instead seek an error bound based on quantities that are "always computable." One way to see how good $\tilde{x}$ is is to compute residual vector $r = b - A\tilde{x}$ and consider $\|r\|_2$ – how well does $\tilde{x}$ satisfy the equation we want it to?

Note that this is not scale invariant – multiplying A, b by c increases r as well. To fix this, we instead consider relative residual $\rho = \frac{\|b-A\tilde{x}\|_2}{\|A\|_2\|\tilde{x}\|_2}$.

Lemma 4. *Motivation:* $\rho = \min_{\delta A}\left\{\frac{\|\delta A\|_2}{\|A\|_2} : (A+\delta A)\tilde{x} = b\right\}$

Proof omitted (in textbook).

This lemma implies a bound on the forward error $\frac{\|x-\tilde{x}\|}{\|x\|} \leqslant \kappa(A)\rho$.

This type of bound is useful for iterative algorithms, where we repeatedly compute until we get "close enough" – this provides a metric for how close we are.

Alternatively, we can consider a relative metric $\frac{\|b-A\tilde{x}\|_2}{\|b\|_2} = \epsilon$. This doesn't provide a good bound on backwards error like above, but does provide a relative forward error bound $\frac{\|x-\tilde{x}\|_2}{\|x\|_2} \leqslant \kappa(A)\epsilon$.

5 Triangular factorizations with specific structure

Given "structured" A, what to do? E.g. A is symmetric positive definite. Seems reasonable that we can try something like LU but take advantage of the symmetry to avoid solving for both matrices. Additionally, we will be able to achieve good stability without permuting P.

Suppose we have $A \in \mathbb{R}^{n\times n}$ symmetric positive definite, i.e. $x^TAx \geqslant 0$ for all x and 0 iff $x = 0$; equivalently, all positive eigenvalues. Then there always exists upper triangular R such that $A = R^TR$, the so-called Cholesky decomposition. If all $r_{ii} > 0$, this is unique. Sometimes written LL^T with L lower triangular.

Existence and algorithm: Let A be as follows:

$$A = \begin{pmatrix} \alpha_{11} & w^T \\ w & K \end{pmatrix}$$

$$\left(\begin{array}{c|c} \alpha_{11} & w^T \\ \hline w & \mathrm{K} \end{array}\right)$$

where $\alpha_{11} \in \mathbb{R}, w \in \mathbb{R}^{n-1}, K \in \mathbb{R}^{(n-1)\times(n-1)}$. We claim that we can reduce similarly to before, by one column at a time.

$$A = \begin{pmatrix} \sqrt{\alpha_{11}} & 0 \\ w/\sqrt{\alpha_{11}} & I \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & K - \frac{ww^T}{\alpha_{11}^2} \end{pmatrix} \begin{pmatrix} \sqrt{\alpha_{11}} & w^T/\sqrt{\alpha_{11}} \\ 0 & I \end{pmatrix}$$

Or succinctly, $R_1^T(R_1^{-T}AR_1^{-1})R_1$.

Suppose that $K - \frac{ww^T}{\alpha_{11}}$ is symmetric positive definite (induction argument). 1×1 spd matrix has a Cholesky decomposition (just take the square root). So claim that $K - \frac{ww^T}{\alpha_{11}} = \hat{R}^T\hat{R}$. Then we can write:

$$A = \begin{pmatrix} \sqrt{\alpha_{11}} & 0 \\ w/\sqrt{\alpha_{11}} & I \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \hat{R}^T\hat{R} \end{pmatrix} \begin{pmatrix} \sqrt{\alpha_{11}} & w^T/\sqrt{\alpha_{11}} \\ 0 & I \end{pmatrix}$$

This implies we can rewrite A as:

$$A = \begin{pmatrix} \sqrt{\alpha_{11}} & 0 \\ w/\sqrt{\alpha_{11}} & \hat{R}^T \end{pmatrix} \begin{pmatrix} \sqrt{\alpha_{11}} & w^T/\sqrt{\alpha_{11}} \\ 0 & \hat{R} \end{pmatrix}$$

Couple notes to check: we took $\sqrt{\alpha_{11}}$. Note that this is always possible since spd A must have positive diagonals: $e_i^TAe_i = \alpha_{ii}$ and this quantity must be strictly positive. The other thing to check is that $K - \frac{ww^T}{\alpha_{11}}$ has a Cholesky decomposition, i.e. it is spd. Symmetry is immediate: K and ww^T are both symmetric. Observe that $R_1^{-T}AR_1^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & K - \frac{ww^T}{\alpha_{11}^2} \end{pmatrix}$. But this is also spd:

$$x^T(R_1^{-T}AR_1^{-1})x = (x^TR_1^{-T})A(R_1^{-1}x) \geqslant 0$$

with equality only if $R_1^{-1}x = 0$; since R is invertible, this requires $x = 0$.

For implementation, textbooks useful reference for exact expression w/ indices.

Cost to compute R is $\sim \frac{1}{3}n^3$.

Stability? Do we have issues with entries getting too large? Here, note that $\|R\|_2 = \|R^T\|_2 = \|A\|_2^{1/2}$ (to prove this, use SVD of R). So when A is spd, our previous LU bound $\frac{\|\delta A\|}{\|L\|\|U\|}$ is essentially just $\frac{\|\delta A\|}{\|A\|}$; bounds on our growth factors for spd matrices.

Theorem 5. *Given A spd, compute Cholesky decomposition $\tilde{R}$. Then $\tilde{R}$ satisfies $\tilde{R}^T\tilde{R} = A + \delta A$, with $\frac{\|\delta A\|}{\|A\|} = O(\mu)$.*

Proof is essentially using the error analysis from LU and the above bound on $\|\tilde{R}\|$.

Note: One of the more efficient ways to check that a matrix is spd is to attempt to compute Cholesky. If we take square root of a negative number, we know it is not spd; otherwise, we obtain a Cholesky factorization so our matrix is spd.

Theorem 6. *Solve $Ax = b$ for spd A via Cholesky decomposition, i.e. factor $A = R^TR$ and solve $R^Ty = b, Rx = y$ on triangular matrices. Then $\tilde{x}$ satisfies $(A + \delta A)\tilde{x} = b$ for some δA with $\frac{\|\delta A\|}{\|A\|} = O(\mu)$.*

Note that we never gave bound on distance between $\tilde{R}$ and actual solution R – we don't need it to solve this linear system. If actual Cholesky is required, the forward error bound is not so nice. We will see an analogue of this result for pivoted LU in homework – the backward error bound proof will be cleaner than the direct forward error analysis.

So far, we have examined triangular factorizations (potentially with permutations). These factorizations have some limits. Now, we consider factorizations including orthogonal matrices (as well as triangular). This will allow us to solve a broader class of problems, in particular, least squares problems.

6 Orthogonal Matrices

Small linear algebra warmup: projection matrices/projectors. An $n \times n$ matrix P is a projector iff $P^2 = P$. Additionally, if $P = P^T$ as well (or $P = P^*$ for complex), then P is an orthogonal projector. In this class, we assume all projectors are orthogonal (refer to oblique projectors otherwise). Note that orthogonal projectors are not orthogonal matrices (other than identity).

Key idea: P projects vectors v onto $range(P)$. P takes v to Pv (closest point to v on $range(P)$; vector $Pv - v$ is orthogonal to $range(P)$. This fact implies that Pv is the closest (in $\|\cdot\|_2$) point to v in the range of P. Observe also that if $x \in range(P)$, $Px = x$.

Note that given an orthogonal projector P, then $I - P$ is the orthogonal projector onto $range(P)^\perp$ (orthogonal complement). Observe that $P(I - P) = P - P^2 = 0$. Additionally, we have $y^T P[(I - P)x] = 0$: y captures every element of the range of P, and x captures every element in the range of $I - P$; any such pair of vectors Py and $(I - P)x$ are orthogonal (substituting $P = P^T$ in the above expression). Thus, we can decompose any vector $x \in \mathbb{R}^n$ as $(Px) + (I - P)x$ – projector divides space into everything on P and everything orthogonal to it (and the latter is defined implicitly). Moreover, $\|x\|_2^2 = \|Px\|_2^2 + \|(I - P)x\|_2^2$ (via orthogonality).

Given a subspace we care about, how can we build an orthogonal projector onto it?

1. Compute an orthonormal basis $q_1, \ldots, q_k$ for the subspace. Let $Q = \begin{pmatrix} | & | & & | \\ q_1 & q_2 & \ldots & q_n \\ | & | & & | \end{pmatrix}$ (matrix with orthonormal columns).
2. $P = QQ^T$ is the orthogonal projector onto $range(Q)$. Easy to check that $P^2 = P$ and $P^T = P$. It turns out that this is the unique projector (regardless of choice of Q); we can thus reason about subspaces with unique object.

Given a general basis, we can compute the orthogonal projector by converting first to orthonormal basis.

This leads us to (one-sided) orthogonal factorizations. For now, we first focus on the so-called QR factorizations. Given any $a \in \mathbb{R}^{m \times n}$ with $m \geqslant n$, there exist matrix $Q \in \mathbb{R}^{m \times n}$ with orthonormal columns and upper triangular matrix $R \in \mathbb{R}^{n \times n}$ such that $A = QR$. This is the (reduced) QR factorization of A.

Key points: The factorization is not unique. But there are certain structural assumptions that restrict the degrees of freedom, so the set of possible factorizations is small; the "lack of uniqueness" ends up not really mattering. If A has full column rank, then R is nonsingular (and vice versa).

Side note: A "full" QR of A with $m \geqslant n$ is (some books will just call this QR decomposition):

$A = QR$ where $Q \in \mathbb{R}^{m \times m}$ satisfies $Q^T Q = I$ and $R \in \mathbb{R}^{m \times n}$ and the upper $n \times n$ submatrix

is upper triangular (and all 0's below it).

If $A \in \mathbb{R}^{m \times n}$ with $n \geqslant m$, then the factorization is $A = Q[R_1 \ R_2]$ where $Q \in \mathbb{R}^{m \times m}$ and $Q^T Q = I$, $R_1 \in \mathbb{R}^{m \times m}$ is upper triangular, and $R_2 \in \mathbb{R}^{m \times (n-m)}$ and is dense.

We will see algorithms shortly; but first motivation via applications.

What can we do with QR?

If $m = n$, we can solve linear systems: Solve $Ax = b$ via $A = QR \implies QRx = b$, so $Rx = Q^T b$ and solve for x with upper triangular R.

If $m = n$, we can use QR as a tool in eigenvalue algorithms.

If $m > n$ and we want to solve $\min_x \|Ax - b\|_2^2$ (least squares problem). This can also be written $\sum_{i=1}^{m} (a_i^T x - b_i)^2$ (sum of losses).

In general, how do we solve least squares?

1. "First" choice: solve $\min_x x^T A^T A x - 2x^T A^T b + b^T b$. Last term is constant in x so we can drop it; solve for stationary point by solving where gradient = 0. Gradient w.r.t. x gives $A^t A x = A^T b$, with Hessian $A^T A$. If A is full column rank, then $A^T A$ is spd and therefore nonsingular; stationary point is unique and a minimizer. So we can solve by finding the stationary point (e.g. with Cholesky). This is called solving the normal equations (the equations which arise from $A^T A x = A^T b$). In general, don't do this – squares the condition number of A: $\kappa(A^T A) = \kappa(A)^2$ (can see via SVD).

2. This problem is essentially "find the closest point in $range(A)$ to b." From before, if P is orthogonal projector onto $range(A)$, then closest point is Pb. If we have factorization $A = QR$, then $P = QQ^T$ (assuming full column rank). Thus $\min_x \|Ax - b\|_2^2$ simplifies to finding x such that $Ax = QQ^T b$. Then $QRx = QQ^T b$ so $Q(Rx - Q^T b) = 0$, which is true iff $Rx = Q^T b$. When A full column rank, this is $n \times n$ non-singular linear system and we can solve for x.

3. If $A = U\Sigma V^T$ is reduced SVD, follow same strategy as above option. Left singular vectors gives basis for $range(A)$. $Ax = UU^T b \implies U\Sigma V^T x = UU^T b \implies \Sigma V^T x = U^T b$. Again end up with nonsingular matrix and solve like above.

Generally speaking, normal equations is fastest, SVD is most stable, and QR is nice middle ground (most commonly used in practice). SVD may be better when rank-deficient.

7 Computing QR factorizations

We assume $A \in \mathbb{R}^{m \times n}$ with $m \geqslant n$, and assume A is full column rank. We want $A = QR$ with $Q \in \mathbb{R}^{m \times n}$ with orthonormal columns, $R \in \mathbb{R}^{n \times n}$ upper triangular.

For LU, we started with A and found nice L_i to slowly make it upper triangular. Our procedure here will be similar in spirit.

First method is triangular orthogonalization, i.e. build $R_1, \ldots, R_n$ upper triangular such that $AR_1 \ldots R_n = Q$.

Second method is orthogonal triangularization. $Q_n \ldots Q_1 A = R$ where each Q_i is orthogonal.

Triangular orthogonalization

Observation: If $A = QR$

$$\begin{pmatrix} | & | & & | \\ a_1 & a_2 & \ldots & a_n \\ | & | & & | \end{pmatrix} = \begin{pmatrix} | & | & & | \\ q_1 & q_2 & \ldots & q_n \\ | & | & & | \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & \ldots & r_{1n} \\ & r_{22} & \ldots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \end{pmatrix}$$

We know $a_1 = q_1 r_{11}$; note that for a single vector to satisfy orthonormality, we need $\|q_1\| = 1$. This means that $q_1 = \frac{a_1}{\|a_1\|_2} \implies r_{11} = \pm \|a_1\|_2$.

Similarly, $a_2 = q_1 r_{12} + q_1 r_{22}$. In particular, this implies that q_2 must be in $\text{span}\{a_2, q_1\}$: $a_2 - q_1 r_{12} = q_2 r_{22}$. But q_1 is in the span of a_1 (as it's a scalar multiple). So in particular, q_i is in the span of $\{a_j : j \leqslant i\}$ and $\text{span}\{a_1, \ldots, a_j\} = \text{span}\{q_1, \ldots, q_j\}$.

We use projectors to make this procedure more clear and improve numerical stability. Let $P_i = q_i q_i^T$ (easy to check this is projector). When we go from $\{a_1, \ldots, a_j\}$ to $\{a_1, \ldots, a_{j+1}\}$, need to find q_{j+1} orthogonal to all previous vectors. We know q_{j+1} is in the span of $\{a_1, \ldots, a_{j+1}\}$, perpendicular to $q_1, \ldots, q_j$, and normalized.

For any v, $\left(I - \sum_{i=1}^{j} P_i\right) v$ is orthogonal to $q_1, \ldots, q_j$ (orthogonal projector onto complement). To keep the spans the same, take $v = a_{j+1}$.

General idea (3):

We can make the actual internal calculations more explicit. For our actual factorization, we end up with (4):

Potential issue – Gram-Schmidt is unstable. Suppose A is ill-conditioned. At some point, $a_j \approx \left(\sum_{i=1}^{j-1} P_i\right) a_j$. We subtract two quantities that are close together (but not necessarily small), which allows the condition number to blow up. In practice, re-ordering the columns for projection order makes this better (full explanation in Demmel).

Algorithm 3 Gram-Schmidt (simplified)

Take $q_1 \leftarrow \frac{a_1}{\|a_1\|_2}$
for $j = 2, \ldots, n$ **do**
  $v_j \leftarrow \left(I - \sum_{i=1}^{j-1} P_i\right) a_j$
  $q_j \leftarrow \frac{v_j}{\|v_j\|_2}$
  $P_j \leftarrow q_j q_j^T$
end for

Algorithm 4 Gram-Schmidt (complete)

for $j = 1, \ldots, n$ **do**
  $v_j \leftarrow a_j$
  for $i = 1, \ldots, j-1$ **do**
    $r_{ij} \leftarrow q_i^T a_j$
    $v_j \leftarrow v_j - r_{ij} q_i$
  end for
  $r_{jj} \leftarrow \|v_j\|_2$
  $q_j \leftarrow v_j / r_{jj}$
end for

To fix this, we replace the v_j assignment by the applying the projections successively:

$$v_j = (I - P_{j-1}) \ldots (I - P_1) a_j$$

Since projections are onto orthogonal ranges, any pairwise or greater product is 0. This gives a mathematically equivalent result.

Intuitions: The absolute differences are not as large as with the product of the combined sum. Additionally, each of the vectors remains small at each step, so less likely to be ill-conditioned. The errors at each projection are "roughly orthogonal" – can possibly cancel out.

Algorithm 5 Modified Gram-Schmidt (complete)

for $j = 1, \ldots, n$ **do**
  $v_j \leftarrow a_j$
end for
for $i = 1, \ldots, n$ **do**
  $r_{ii} \leftarrow \|v_i\|_2$
  $q_1 \leftarrow v_i / r_{ii}$
  for $j = i+1, \ldots, n$ **do**
    $r_{ij} \leftarrow q_i^T v_j$
    $v_j \leftarrow v_j - r_{ij} q_i$
  end for
end for

The contents of the inner for loop essentially apply $(I - q_i q_i^T)$ to $v_{i+1}, \ldots, v_n$.

Consider how our procedure builds R:

$$\begin{pmatrix} | & | & & | \\ a_1 & a_2 & \dots & a_n \\ | & | & & | \end{pmatrix} \begin{pmatrix} \tilde{r}_{11} & \tilde{r}_{12} & \dots \\ 0 & \tilde{r}_{22} & \dots \\ 0 & 0 & \ddots \\ 0 & 0 & w \end{pmatrix} = \begin{pmatrix} | & | & & | \\ q_1 & q_2 & \dots & q_n \\ | & | & & | \end{pmatrix}$$

In essence, we are building R^{-1} one column at a time.

Both Gram-Schmidt and modified Gram-Schmidt run in $\sim 2mn^2$ flops.

If $n > m$, this process will break down – at some point, we will find a column in the span of previous vectors. This gives the factorization $Q[R_1\ R_2]$ where R_1 is upper triangular and R_2 is dense. In this case, runtime is $2m^2n$; so we may see overall runtime abbreviated as $2mn \min\{m, n\}$ flops.

Orthogonal triangularization

Now we construct $Q_1, \dots, Q_n$ orthogonal matrices such that:

$$Q_n \dots Q_1 A = R$$

Note that Q_i are all $m \times m$, so computed $\tilde{R}$ is $m \times n$ upper triangular. This means that for $A = \tilde{Q}\tilde{R}$, we "reduce" by only keeping the first n columns of $\tilde{Q} = Q_1^T \dots Q_n^T$.

Broad idea is to follow same idea as with LU.

$$\begin{pmatrix} x & x & \dots & \dots & x \\ x & \ddots & & & \vdots \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ x & \dots & \dots & \dots & x \end{pmatrix} \xrightarrow{Q_1} \begin{pmatrix} x & x & \dots & \dots & x \\ 0 & x & \dots & & \vdots \\ \vdots & \vdots & \ddots & & \vdots \\ \vdots & & & & \vdots \\ 0 & \dots & \dots & \dots & x \end{pmatrix} \xrightarrow{Q_2} \dots \xrightarrow{Q_n} \begin{pmatrix} x & x & \dots & \dots & x \\ 0 & x & \dots & & \vdots \\ 0 & 0 & x & \dots & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & \dots & \dots & \dots & x \end{pmatrix}$$

How do we ensure that our Q_i's don't mess up previous elements? Idea: Make Q_i's block diagonal:

$$Q_i = \begin{pmatrix} I & \\ & F_i \end{pmatrix}$$

Where I is the $(i-1) \times (i-1)$ identity matrix. Note that this preserves our established columns. Key observation: If F_i is orthogonal, then so is Q_i.

How to find F_i? Given a vector x, construct F such that

$$Fx = \begin{pmatrix} \pm\|x\|_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \pm\|x\|_2 e_1$$

Since orthogonal matrices preserve norm. For now, we assume that we pick the positive option (algorithm will eventually dynamically pick).

We want to send x to $\|x\|_2 e_1$. Consider the vector representing this action and the space of everything perpendicular to this vector $(\|x\|_2 e_1 - x)^\perp$. Then $\|x\|_2 e_1$ is the reflection of x over this space. Note that we are not viewing orthogonal matrix as rotations here, since it may be difficult to compute the necessary angles in higher dimensions. But the reflection generalizes nicely, as we use projectors.

Let $v = \|x\|_2 e_1 - x$. Note that v is twice the projection from x onto the orthogonal space H; i.e., $F = I - 2\frac{vv^T}{v^T v}$ (Householder reflector).

Recall that last time, for $A \in \mathbb{R}^{m \times n}$ we want to find Q_i $m \times m$ orthogonal with:

$$Q_n \dots Q_1 A = \begin{bmatrix} R \\ 0 \end{bmatrix}$$

We take each $Q_i = \begin{pmatrix} I & \\ & F_i \end{pmatrix}$ where I is $(i-1) \times (i-1)$ and F_i is $n - (i-1) \times n - (i-1)$. If F_i is orthogonal, then so is Q_i.

Given a vector x, we wish to construct F such that:

$$Fx = \begin{pmatrix} \pm\|x\|_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \pm\|x\|_2 e_1$$

In 2-D, we can think about rotating x until it reaches the x-axis. However, this rotation method along unit sphere does not generalize well to higher dimensions. Instead, we appeal to a different type of orthogonal matrix: reflection matrix.

Take H to be the hyperspace perpendicular to $v = \|x\|_2 e_1 - x$ (vector sending x to $\|x\|_2 e_1$). Note then that reflecting x over H gives $\|x\|_2 e_1$.

A reflection can be performed by first projecting x onto H; then repeating this procedure from the projected point.

When we normalize v, we can write the projector onto the space H by $I - \frac{vv^T}{v^T v}$ (following $I - QQ^T$ notation, with normalized v). So take:

$$\underbrace{(I - \frac{vv^T}{v^T v})y}_{\text{project } y \text{ onto } H} \qquad - \qquad \underbrace{\frac{vv^T}{v^T v}y}_{\text{Keep going same amount}}$$

so $F = (I - 2\frac{vv^T}{v^T v}$ with $v = \|x\|_2 e_1 - x$ is known as Householder reflector.

Note that we can repeat this same procedure with $-\|x\|_2 e_1$ instead. Note that, depending

on our choice, v looks like:

$$v = \begin{pmatrix} \pm\|x\|_2 - x_1 \\ -x_2 \\ \vdots \\ -x_n \end{pmatrix}$$

Note that we may lose precision when $\pm\|x\|_2 \approx x_1$ (subtracting two numbers very close to each other). So we pick the option that reduces this error – if $x_1 \approx \|x\|_2$, take $-\|x\|_2 e_1$ and vice versa. Essentially pick the $\pm$ sign to match x_1 and avoid cancellation – geometrically, we reflect based on the point which is further away.

$$v = sign(x_1)\|x\|_2 e_1 + x$$

Algorithm 6 Householder QR

for $k = 1, \ldots, n$ **do**
  $x \leftarrow A(k:m, k)$
  $V_k \leftarrow sign(x_1)\|x\|_2 e_1 + x$
  $V_k = V_k / \|V_k\|_2$
  $A(k:m, k:n) = A(k:m, k:n) - 2V_k(V_k^T A(k:m, k:n))$
end for

At completion, we have $A = \begin{bmatrix} R \\ 0 \end{bmatrix}$ and "store" Q_i implicitly via $V_i + Q_i = (I - 2V_i V_i^T)$. Then Q in the QR factorization is: $Q = Q_1^T \cdots Q_n^T$. Note that we don't particularly need to solve for Q; just need a way to apply it. Alg to compute $Q^T b$:

for $k = 1, \ldots, n$ **do**
  $b(k:m) = b(k:m) - 2V_k(V_k^T b(k:m))$
end for

This result is $m \times 1$; just keep the first n entries. Similarly, to compute Qx:

for $k = n, \ldots, n-1$ **do**
  $x(k:m) = x(k:m) - 2V_k(V_k^T x(k:m))$
end for

Also note that each Q_i is symmetric, so $Q_i^T = Q_i$ (and the transposes will often be dropped).

Theorem 7. *Stability of Householder QR for complete QR factorization of $A \in \mathbb{R}^{m \times n}$ (assume $m \geqslant n$). Suppose we compute this and store computed vectors $\tilde{V}_k$ and $\begin{bmatrix} \tilde{R} \\ 0 \end{bmatrix}$ (in particular, store $\tilde{V}_k$ and not $\tilde{Q}$. Then we have backwards stability:*

$$\tilde{Q}\tilde{R} = A + \delta A$$

with $\frac{\|\delta A\|}{\|A\|} = O(\mu)$, where $\tilde{Q} = \tilde{Q}_1^T \cdots \tilde{Q}_n^T$ and $\tilde{Q}_k$ is the exact Householder reflector induced by $\tilde{V}_k$:

$$\tilde{Q}_k = \begin{pmatrix} I & 0 \\ 0 & I - 2\frac{\tilde{V}_k \tilde{V}_k^T}{\tilde{V}_k^T \tilde{V}_k} \end{pmatrix}$$

Note that this is not necessarily orthogonal when computed; thus, the need to take $\tilde{Q}_i$ to be exact Householder reflectors. In practice, this isn't too bad because we're not storing Q anyways.

Say we have a matrix:

$$A = \begin{pmatrix} x & \dots & \dots & \dots & x \\ x & x & & & \vdots \\ 0 & x & & & \vdots \\ \vdots & 0 & \ddots & & \vdots \\ \vdots & & & & \vdots \\ 0 & \dots & \dots & \dots & x \end{pmatrix}$$

e.g. i^{th} column only has $i+1$ nonzero entries.

Computing QR with Householder seems to do a lot of work when we only need to introduce a single 0 in each column. Better to strategically replace 0's with row operations (as each such operation is more efficient than Householder operation), even though in worse matrices, can go up to n^2 Q_i's in general.

Suppose our matrix looks like:

$$A = \begin{pmatrix} x & x & \dots & \dots & x \\ 0 & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ x & & & & \vdots \\ 0 & & & & \vdots \\ \vdots & & & & \vdots \\ 0 & \dots & \dots & \dots & x \end{pmatrix}$$

We build an orthogonal matrix:

$$\begin{pmatrix} c & & s & \\ & I & & \\ -s & & c & \\ & & & I \end{pmatrix}$$

where $c = \cos\theta$ and $s = \sin\theta$ to zero out $A(j,i)$. More generally, define the so-called Givens rotation:

$$\begin{pmatrix} I & & & & \\ & c & & s & \\ & & I & & \\ & -s & & c & \\ & & & & I \end{pmatrix}$$

in rows/cols i, k, $c = \cos\theta, s = \sin\theta$. G^T rotates that i_k plane by θ counterclockwise. How

do we pick θ? Need c, s such that:

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}^T \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \pm r \\ 0 \end{pmatrix}$$

where $r = a^2 + b^2$. Since we work with explicit 2×2, can actually compute:

$$c = \frac{a}{\sqrt{a^2 + b^2}} \qquad\qquad s = \frac{-b}{\sqrt{a^2 + b^2}}$$

But like with Householder, want to be slightly careful with our choices (based on whether $|a| > |b|$).

In practice:

if $b = 0$ **then**
  $c \leftarrow 1, s \leftarrow 0$
else if $|b| > |a|$ **then**
  $\tau \leftarrow \frac{-a}{b}, s \leftarrow \frac{1}{\sqrt{1+\tau^2}}, c \leftarrow s\tau.$
else
  $\tau \leftarrow \frac{-b}{a}, c \leftarrow \frac{1}{\sqrt{1+\tau^2}}, s \leftarrow c\tau.$
end if
return c, s

Upcoming topics:

- Sensitivity of LS (least squares)
- Rank-deficient LS (and rank-deficient problems in general)

8 Conditioning of LS

More complex than $m = n$, non-singular systems. When b not in $range(A)$, we solve for x such that Ax is the point in $range(A)$ closest to b. Now consider what happens when we perturb b. Before, square nonsingular matrices produced exact solution for b, so no real issue occurred.

Now, when we perturb b, there are two effects: the point in the range of A changes; as this changes, so too does the x. This ends up being not too bad. Now consider if we perturb A. Before, perturbing A is fine as long as we keep it nonsingular. Again, it affects both the closest point to b as well as the x used to get there.

Denote $y = Ax$ (the point closest to b). We consider (relative) condition number κ of y, x w.r.t. δA, δb (not perturbed simultaneously).

	y	x
b	$\dfrac{1}{\cos(\theta)}$	$\dfrac{\kappa(A)}{\eta\cos(\theta)}$
A	$\dfrac{\kappa(A)}{\cos(\theta)}$	$\kappa(A) + \dfrac{\kappa(A)^2\tan(\theta)}{\eta}$

where $\theta = \cos^{-1}\frac{\|y\|}{\|b\|}$ (the angle between b and y) and $\eta = \frac{\|A\|\|x\|}{\|Ax\|}$ (measures how big Ax is compared to how big it can be; observe that $1 \leqslant \eta \leqslant \kappa(A)$). Informal proofs of these available in Trefethen-Bau; more rigorous in GVL (notational differences too).

Theorem 8. *Solve LS* $\min_x \|b - Ax\|$ *via Householder QR (or with Givens rotations), i.e., compute* $A = QR$ *and solve* $Rx = Q^T b$. *Then the computed* $\hat{x}$ *minimizes* $\|b - (A + \delta A)\hat{x}\| = \min$ *for some* δA *satisfying* $\frac{\|\delta A\|}{\|A\|} = O(\mu)$.

This implies that using QR we get:

$$\frac{\|\hat{x}_{QR} - x\|}{\|x\|} = O\left(\left(\kappa + \frac{\kappa^2}{\eta}\tan(\theta)\right)\mu\right)$$

Compare to what we would get when solving with normal equations:

$$\frac{\|\hat{x}_N - x\|}{\|x\|} = O(\kappa^2\mu)$$

So the question becomes when is this much worse than QR? That is, when is $\frac{\kappa^2}{\eta}\tan(\theta)$ close to κ^2? Typically solving with normal equations is worse; but how bad is it?

If $b \in range(A)$, $\tan(\theta)$ vanishes and we end up with $\kappa\mu$. On the other hand, when b close to orthogonal to y, this term blows up. It turns out that this also results in a large blowup when solving via normal equations (since we compute $A^T b$).

We can write:

$$\begin{aligned}\frac{\kappa \tan(\theta)}{\eta} &= \frac{\|A\|_2\|A^{-1}\|_2\|b-Ax\|_2}{\|Ax\|_2} \times \frac{\|Ax\|_2}{\|A\|_2\|x\|_2} \\ &= \frac{1}{\sigma_n} \times \frac{\|b-Ax\|}{\|x\|}\end{aligned}$$

Observe that when the "mathematical residual" $\|b-Ax\|$ is small, we are generally okay (this matches with the intuition that we can solve well when b is in $range(A)$). We can write in terms of the SVD to shed some insight; let $A = U\Sigma V^T$ be the reduced SVD of A.

$$= \frac{1}{\sigma_n} \times \frac{\|(I-UU^T)b\|}{\|\Sigma^{-1}U^Tb\|}$$

Intuitively, $(I-UU^T)b$ corresponds to "the part of b orthogonal to A", and U^Tb (ignoring the Σ^{-1} term) "the part of b in the range of A."

$$\leqslant \kappa \times \frac{\|(I-UU^T)b\|}{\|UU^Tb\|}$$

What happens if A is rank deficient?

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T, \qquad m \geqslant n, r < n$$

What if A is even rank deficient when converted to floating point (not always mathematically the case), and we don't know beforehand that it is rank deficient? Say we compute SVD of A and get $\hat{\sigma}_i$ for $i = 1, \ldots, n$ (computed singular values). Roughly $|\hat{\sigma}_i - \sigma_i| \leqslant \mu\sigma_1$, i.e. all perturbations roughly on the order of the largest singular value. Indicates a global relationship between singular values. Then $\hat{\sigma}_{r+1}, \ldots, \hat{\sigma}_n \approx \mu\sigma_1$ (so unlikely for them to be 0 and result to be singular). When plotting singular values, graph "bottoms out" at r, with values approx 10^{-16}.

Say A is rank-deficient. To fix a specific solution to $\min_x \|b-Ax\|$, take the one with smallest norm x. We have a closed form expression for x in terms of SVD:

$$x = \sum_{i=1}^{r} \frac{u_i^Tb}{\sigma_i} v_i$$

Solve like normal LS problem, just ignoring zero singular values of A. But the computed $\hat{x} = \sum_{i=1}^{n} \frac{\hat{u}_i^Tb}{\hat{\sigma}_i}\hat{v}_i$ – terms that didn't appear before are now present; we're additionally dividing by $\hat{\sigma}$'s – leads to potentially big errors when b has components in the directions of singular vectors corresponding to value 0.

Instead, we replace the upper bound of sum with $\hat{r}$, the "numerical rank" of A. This helps reduce the error caused by the above summation. Consider:

Pick some $\delta \geqslant 0$ and let $\hat{r}$ be s.t. $\hat{\sigma}_1 \geqslant \cdots \geqslant \hat{\sigma}_{\hat{r}}\ \delta > \hat{\sigma}_{\hat{r}+1} \geqslant \cdots \geqslant \hat{\sigma}_n \geqslant 0$

Use $\hat{x} = \sum_{i=1}^{\hat{r}} \frac{\hat{u}_i^T b}{\hat{\sigma}_i} \hat{v}_i$

i.e. we explicitly discard small singular values. Note that this depends on our choice of δ (one natural way is to make $\delta \approx \mu\sigma_1$ based on the above observation). In practice, people don't want to compute $\hat{\sigma}_1$ so use proxies, e.g. $\|A\|_\infty$.

What can go wrong? What if $\hat{r} < r$, and we end up throwing away singular values that are nonzero. Consider "error" $= \sum_{i=\hat{r}+1}^{r} \frac{u_i^T b}{\sigma_i} v$ (observe that we use exact values to make this point, rather than comparing between exact and computed). We could have big errors if $u_i^T b$ is significant. But we can roughly interpret as solving the problem $\min_x \|b - Ax\|_2$ s.t. $\|x\| \geqslant \frac{1}{\delta}$ (i.e. put additional constraint on how big x can be).

9 Iterative methods

Still solving $Ax = b$ for A square, nonsingular. Before: we were computing e.g. $A = QR$, to find $Q^T b$ and then solve $Rx = Q^T b$. Note that this solution method uses a fixed number of arithmetic operations in each step (which we can explicitly count). We can even give bounds on how bad $\hat{x}$ is compared to x.

Now: We construct a sequence $x^{(k)} \in \mathbb{R}^n$ such that $x^{(k)} \to x = A^{-1}b$ as $k \to \infty$. In other words, we construct a sequence of vectors that converges to the true answer.

We start by considering this problem with exact arithmetic; afterwards, we will reason about the effects of floating-point.

Previous direct methods – no improvement until after fully factoring the matrix. So our best guess is constant until this work is done, at which point the rror becomes $\kappa\mu$. On the other hand, this new method allows us to converge over time – potentially even faster than the $O(n^3)$ from before. Or perhaps we don't need such a precise answer (e.g. measurement uncertainty is higher than machine precision) so we can afford to stop early.

First class of methods: Stationary or Classical iterations.

Idea: Split $A = M - N$ for some M, N with M nonsingular. Iteration defined as $Mx^{(k)} = Nx^{(k-1)} + b$ given some $x^{(0)}$. Observe that plugging in x yields x as a valid solution.

Note that with this method, we want M to be "easy" to solve linear systems with. Additionally, we want this method to converge.

Observe that $Mx = Nx + b$ for $x = A^{-1}b$. We can take $e^{(k)} = x - x^{(k)}$ so that:

$$M(x - x^{(k)}) = N(x - x^{(k-1)}) \implies Me^{(k)} = Ne^{(k-1)} \implies e^{(k)} = M^{-1}Ne^{(k-1)}$$

If we call $G = M^{-1}N$ the "iteration matrix," we can unroll iteration:

$$e^{(k)} = G^k e^{(0)}$$

For any induced norm:

$$\|e^{(k)}\| = \|G^k e^{(0)}\| \leqslant \|G^k\|\|e^{(0)}\| \leqslant \|G\|^k \|e^{(0)}\|$$

So if $\|G\| < 1$ for *some* induced (or sub-multiplicative) norm, this iteration converges. Key quantity to dictate convergence is spectral radius $\rho(G)$.

Theorem 9. *Suppose $A = M - N$ with M nonsingular. Then the iteration $Mx^{(k)} = Nx^{(k-1)} + b$ converges for all $x^{(0)}$ iff $\rho(G) < 1$, where $G = M^{-1}N$.*

Proof. Assume $\rho(G) \geqslant 1$ and pick $x^{(0)}$ such that $e^{(0)}$ is an eigenvector of G with eigenvalue $|\lambda| = \rho(G)$. Then $e^{(k)} = \lambda^k e^{(0)}$; when $|\lambda| \geqslant 1$, does not converge.

For all G, ϵ there exists sub-multiplicative norm $\|\cdot\|_s$ such that $\|G^k\|_s \leqslant c_{\epsilon,G}(\rho(G) + \epsilon)^k$. If $\rho(G) < 1$, we can pick ϵ such that the base of exponent is as well; then $\|G\|_s^k \to 0$ so norm of error for any initial guess goes to 0 as well.

□

Say $G = \begin{pmatrix} \lambda & \alpha \\ & \lambda \end{pmatrix}$. Then $G^k = \begin{pmatrix} \lambda^k & \alpha\lambda^{k-1} \\ & \lambda^k \end{pmatrix}$. So even if the eigenvalues are small, behavior depends on α at low iterations – could potentially have worse error early. Eigenvalues do not tell the entire story for non-normal G. Here, we say A is normal if $A^TA = AA^T$ (orthogonally diagonalizable).

Specific methods: Let $A = D + L + U$ (diagonal part of A, strictly lower triangular, strictly upper triangular).

- Pick $M = D$, $N = -L - U$. This yields the Jacobi iteration. Observe that here $x^{(k)} = D^{-1}((-L - U)x^{(k-1)} + b)$. This method is easy to parallelize.
- Pick $M = D + L$, $N = -U$. This yields the Gauss-Siedel method.

Theorem 10. *If A is strictly diagonally dominant, then Jacobi iteration converges for all $x^{(0)}$.*

A matrix A is strictly diagonally dominant if: $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$.

Proof. First note that $M^{-1}N = D^{-1}(-L - U)$. So we can compute $\|G\|_\infty = \max_{j \neq i} \left|\frac{a_{ij}}{a_{ii}}\right|$. By strict diagonal dominance, this is strictly less than 1, so we converge. □

Theorem 11. *If A is spd, then Gauss-Siedel converges for all $x^{(0)}$.*

Proof. Proof omitted, in GVL. Similar structure to above. □

From Gauss-Siedel, we can obtain a method of successive over-relaxation. Take $M = \frac{1}{w}D + L$, $N = \left(\frac{1}{w} - 1\right)D - U$. Then the key question is how to pick w to achieve small spectral radius.

10 More Iterative Methods

Previously, we had to pick $x^{(0)}$. Today, we explore cases where we can start $x^{(0)} = 0$.

Idea: Iterative refinement. Say we want to solve $Ax = b$ with $x^{(0)} \neq 0$ as initial guess.

1. Compute $r = b - Ax^{(0)}$
2. Solve $Ad = r$ with $d^{(0)} = 0$ (if iterative method).
3. Set $\hat{x} = x^{(0)} + d$ as solution to original problem.

Observe that if we can solve for d exactly, $\hat{x}$ solves the original problem. So our main concern is trying to solve $Ax = b$ with initial guess $x^{(0)} = 0$, since the general problem reduces to this case.

11 Krylov Methods

The next few lectures will focus on various applications of Krylov methods. For $Ax = b$, Krylov methods look for $x^{(k)}$ in a so-called Krylov subspace. More specifically, the k^{th} Krylov subspace.

Define for square A, $\mathcal{K}_k(A, v) = \text{span}\{v, Av, \ldots, A^{k-1}v\}$. In this case, we look for $x^{(k)} \in \mathcal{K}_k(A, b)$.

Note that this is not necessarily the "best" subspace to look in – it would be easy to find our solution in the space spanned by $A^{-1}b$ (but we don't generally know how to find this). Any $y \in \mathcal{K}_k(A, b)$ can be written as $y = P_{k-1}(A)b$ (polynomial in A).

Say we pick $x^{(k)} = P_{k-1}(A)b$ and consider $b - A^{(k)}$. Then we can write:

$$b - Ax^{(k)} = (I - AP_{k-1}(A))b$$

This $\hat{P}_k = (I - AP_{k-1}(A))$ term corresponds to any degree-k polynomial with constant term 1. Now assume A is spd; we can write $A = V\Lambda V^T$. Then we can write $r = V\hat{P}_k(\Lambda)V^Tb$ for some $\hat{P}_k$ degree k with $\hat{P}_k(0) = 1$.

Note that V, V^T are orthogonal so they do not affect the size of vector. So if there exists some $\hat{P}$ such that $|\hat{P}(\lambda_i)|$ is small for all $i = 1, \ldots, n$, then there exists $x^{(k)} \in \mathcal{K}_k(A, b)$ with small residual.

Can reason about Krylov methods as a polynomial approximation problem – if we come up with low-degree polynomial that has low value around k eigenvalues, then $\hat{P}$ is small around all of our eigenvalues. Turns out we can do this implicitly without even solving for eigenvalues.

How to "work with" $\mathcal{K}_k(A, b)$ in a nice way? To work with subspaces, we want a basis – in some sense, the "best" basis is an orthonormal one. Could use:

$$\begin{pmatrix} | & | & & | \\ b & Ab & \ldots & A^{k-1}b \\ | & | & & | \end{pmatrix}$$

or its QR decomposition. But this matrix is ill-conditioned: its columns converge to dominant eigenvector. So instead, we aim to find an orthonormal basis for this matrix without explicitly solving for A^{k-1}. We progressively compute an orthonormal basis (progressively = at step k, we have a basis for $\mathcal{K}_k(A, b)$).

Arnoldi method for $\mathcal{K}_k(A, b)$:

Note that at each step, we apply A to a new vector orthogonal to all of our existing basis vectors (q_ℓ), which helps avoid instability. Additionally, we only use Ay for vectors y – don't directly work with the matrix A (can treat as black box).

This procedure yields orthonormal basis $q_1, \ldots, q_k$ for $\mathcal{K}_k(A, b)$. Additionally, we have a recurrence relation:

$$AQ_k = Q_{k+1}\tilde{H}_k$$

Algorithm 7 Arnoldi Method

$q_1 \leftarrow b/\|b\|_2$
for $\ell = 1, \ldots$ **do**
  $v \leftarrow Aq_\ell$
  for $j = 1, \ldots, \ell$ **do** ▷ Similar to modified Gram-Schmidt
    $h_{j\ell} \leftarrow q_j^T v$
    $v \leftarrow v - h_{j\ell} q_j$
  end for
  $h_{\ell+1,\ell} \leftarrow \|v\|$
  $q_{\ell+1} \leftarrow v/h_{\ell+1,\ell}$
end for

where:

$$Q_k = \begin{pmatrix} | & & | \\ q_1 & \ldots & q_k \\ | & & | \end{pmatrix} \qquad \tilde{H}_k = A = \begin{pmatrix} h_{11} & \ldots & \ldots & \ldots & h_{k1} \\ h_{21} & \ddots & & & \vdots \\ 0 & \ddots & & & \vdots \\ \vdots & 0 & \ddots & & \vdots \\ \vdots & & \ddots & & \vdots \\ 0 & \ldots & \ldots & \ldots & h_{k+1,k} \end{pmatrix}$$

$\tilde{H}_k$ is upper Hessenberg matrix – zero everywhere except upper triangle and one lower sub-diagonal.

We can also write:

$$\tilde{H}_k = \begin{pmatrix} H_k \\ h_{k+1,k} e_k^T \end{pmatrix}$$

for $H_k \in \mathbb{R}^{k \times k}$.

Note that $Q_k^T A Q_k = H_k$, the projection of A onto span$\{Q_k\}$:

$$Q_k^T (Q_k H_k + h_{k+1,k} q_{k+1} e_k^T) = H_k$$

This gives the result of applying A to $Q_k x$ (vector in span of Q_k), written in terms of components of Q_k (via the orthogonal projection Q_k^T), i.e., H.

What if A is symmetric? Observe then that $Q_k^T A Q_k$ is symmetric, so that H_k is as well. But since $\tilde{H}_k$ is 0 below the first lower sub-diagonal, it must be tridiagonal; we call it T_k. In particular, this means that we can avoid computing many entries of T_k, since the majority will be 0. This gives rise to the so-called Lanczo's method.

After this, Q_k is still an orthonormal basis for $\mathcal{K}_k(A, b)$. Now we have $AQ_k = Q_{k+1}\tilde{T}_k$ where:

Algorithm 8 Lanczo's Method

$\beta_0 \leftarrow 0, q_0 \leftarrow 0, q_1 \leftarrow b/\|b\|_2$
for $\ell = 1, \ldots$ **do**
  $v \leftarrow Aq_\ell$
  $\alpha_\ell \leftarrow q_\ell^T v$
  $v \leftarrow v - \beta_{\ell-1} q_{\ell-1} - \alpha_\ell q_\ell$ ▷ Three-term recurrence
  $\beta_\ell \leftarrow \|v\|_2$
  $q_{\ell+1} \leftarrow v/\beta_\ell$
end for

$$\tilde{T}_k = \begin{pmatrix} T_k \\ \beta_k e_k^T \end{pmatrix} \qquad \tilde{T}_k = A = \begin{pmatrix} \alpha_1 & \beta_1 & 0 & \ldots & 0 \\ \beta_1 & \alpha_2 & \ddots & & \vdots \\ 0 & \ddots & \ddots & & \\ \vdots & & & \ddots & \beta_{k-1} \\ 0 & \ldots & \ldots & \beta_{k-1} & \alpha_k \end{pmatrix}$$

In theory, this method is nice; but in practice, we lose orthogonality early on. Disastrous for eigenvectors; mostly benign for solving linear systems

Next time: We solve $Ax = b$ via $x^{(1)}, \ldots, x^{(k)}$, with $x^{(k)} \to x^* = A^{-1}b$. Pick $x^{(k)}$ to be the "best" vector in $\mathcal{K}_k(A, b)$.

For today, we assume A symmetric and recall that Lanczo method yields:

$$AQ_k = Q_{k+1}\tilde{T}_k$$

satisfying Q_k is an orthonormal basis for $\mathcal{K}_k(A, b)$.

Krylov methods for $Ax = b$ are iterative methods with $x^{(k)} \in \mathcal{K}_k(A, b)$. Key ideas: (1) Which $x^{(k)}$ is "best"? (2) Can it be found effeciently?

e.g we could pick $x^{(k)} = \text{argmin}_{x \in \mathcal{K}_k(A,b)} \|A^{-1}b - x\|_2$ – this is the closest point in Krylov subspace to the solution; however, not easily computable. Reasonable for (1), but not (2).

However, we can use this idea with a different norm. Now assume that A is spd. If we denote $x^* = A^{-1}b$, it turns out we can solve:

$$x^{(k)} = \underset{x \in \mathcal{K}_k(A,b)}{\text{argmin}} \|x^* - x\|_A$$

Recall that $\|y\|_B^2 = y^T B y$ for spd B. This is the method of Conjugate Gradients.

How can we efficiently solve for x^*? Note that since we are working with norms, it's equivalent to minimize:

$$\min_{x \in \mathcal{K}_k(A,b)} \|x^* - x\|_A^2$$

We use Lanczo's method (implicitly) to help us. Anything in $\mathcal{K}_k(A, b)$ can be written as $Q_k y$ for some k-dimensional y:

$$y^{(k)} = \min_{y \in \mathbb{R}^k} \|x^* - Q_k y\|_A^2$$

and taking $x^{(k)} = Q_k y^{(k)}$. Expanding out the $\|\cdot\|_A$ gives:

$$\min_{y \in \mathbb{R}^k} (A^{-1}b - Q_k y)^T A (A^{-1}b - Q_k y) \iff \min_y y^T Q_k^T A Q_k y - 2y^T Q_k^T b + b^T A^{-1} b$$

This last term does not depend upon y, so we may discard it and avoid dealing with the A^{-1}:

$$\iff \min_y y^T T_k y - 2\|b\| y^T e_1$$

Where $Q_k^T A Q_{k+1} = \tilde{T}_k$ due to $Q_k^T Q_{k+1} = \begin{pmatrix} & & 0 \\ & I & \vdots \\ & & 0 \end{pmatrix}$ so that $Q_k^T Q_{k+1} \tilde{T}_k = T_k$. and $Q_k^T b$ satisfies: first vector of Q_k is normalized b and all others are orthogonal to it. This yields:

$$\iff y^{(k)} \text{ solves } T_k y^{(k)} = \|b\| e_1$$

Algorithm 9 Conjugate Gradients

Given $A, b, x^{(0)} = 0$;
for $k = 1, 2, \ldots$ **do**
  One step of Lanczos to get $Q_{k-1}, T_{k-1} \to Q_k, T_k$ (also get q_{k+1}, β_k)
  Solve $T_k y^{(k)} = \|b\|_2 e_1$
  Set $x^{(k)} = Q_k y^{(k)}$
  Check convergence
end for

Cost per step:

- $T_{mult}(A) + O(n)$ (i.e. how expensive it is to multiply by A, generally between n and n^2). Note that the 3-term recurrence in Lanczo's means that there is no dependence upon k.
- $O(k^3)$ as written, as we still need to solve $T_k y^{(k)} = \|b\|_2 e_1$.
- $O(nk)$ for finding $x^{(k)} = Q_k y^{(k)}$.

Lanczo's is about the most efficient we can expect to get; however, it turns out that we can remove the dependence upon k in the latter two items.

Note that for $T_k y^{(k)} = \|b\| e_1$, the RHS doesn't change between iterations. Additionally, $T_{k+1} = \begin{bmatrix} & & 0 \\ & T_k & \vdots \\ & & \beta_k \\ 0 & \dots & \beta_k \quad \alpha_{k+1} \end{bmatrix}$ and T_k is tridiagonal.

Our idea is to maintain a factorization of T_k, specifically an LDL^T factorization; where L is unit lower triangular and D is diagonal. $T_k = L_k D_k L_k^T$ allows us to go from L_k, D_k to L_{k+1}, D_{k+1} in $O(1)$ work. This is because T_k is tridiagonal implies that L_k is bidiagonal.

Second key idea is that we don't need to store Q_k; only q_{k-1}, q_k, q_{k+1}. This relies upon the fact that we can rewrite $x^{(k)} = Q_k y^{(k)}$ as: $x^{(k)} = x^{(k-1)} + \xi_k \omega^{(k)}$. Omit the details here; high-level idea is that $\omega^{(k)}$ is related to $Q_k L_k^{-1}$ and can be computed in $O(n)$; ξ_k can be computed in $O(1)$.

This gives a procedure where each step is overall independent of k and only uses $O(n)$ total storage.

All that remains is the check convergence. Most common way is to check $\frac{\|Ax^{(k)} - b\|}{\|A\|\|x^{(k)}\| + \|b\|} < tol$ for some chosen tolerance. Costs of computing $\|b\|, \|x^{(k)}\|$ already baked in. However, we still need norms. Expensive to compute $\|A\|$; but for now we assume we have a good estimate.

We focus on how to compute $\|Ax^{(k)} - b\|_2$. But it turns out that we have already done the required work.

$$\begin{aligned} \|Ax^{(k)} - b\|_2 &= \|b - AQ_k y^{(k)}\|_2 \\ &= \|Q_k(\|b\| e_1) - Q_{k+1} \tilde{T}_k y^{(k)}\|_2 \\ &= \|Q_k \|b\| e_1 - Q_k T_k y^{(k)} - q_{k+1} \beta_k e_k^T y^{(k)}\|_2 \end{aligned}$$

From step 2 above, the first 2 terms cancel out. Then:

$$= \|q_{k+1} \beta_k e_k^T y^{(k)}\|_2 \qquad \text{(Note that this residual is orthogonal to } Q_k\text{)}$$

But q_{k+1} is orthonormal so we have:

$$= \|\beta_k e_k^T y^{(k)}\|_2$$

Now take A to be symmetric, non-singular. We redefine the "best" vector in Krylov space as the one minimizing residual error:

$$\min_{x \in \mathcal{K}_k(A,b)} \|b - Ax\|_2$$

This gives method known as MINRES. Why do we expect to be able to solve this efficiently?

$$\min_{x \in \mathcal{K}_k(A,b)} \|b - Ax\|_2 \iff \min_y \|b - AQ_k y\|_2$$

As written, this is a big least squares problem. We can make the following substitutions:

$$\begin{aligned} &\iff \min_y \left\| Q_{k+1}\|b\|e_1 - Q_{k+1}\tilde{T}_k y \right\|_2 \\ &\iff \min_y \left\| \|b\|e_1 - \tilde{T}_k y \right\|_2 \qquad \text{(now solving } k+1 \times k \text{ LS problem)} \end{aligned}$$

This yields an algorithm quite similar to CG; just with the optimization step replaced by this new minimization. Here we instead store a QR factorization of $\tilde{T}_k$ and set of auxiliary vectors

Algorithm 10 MINRES

Given $A, b, x^{(0)} = 0$;
for $k = 1, 2, \ldots$ **do**
  One step of Lanczos to get $Q_{k-1}, T_{k-1} \to Q_k, T_k$ (also get q_{k+1}, β_k)
  Solve $\min_y \left\| \|b\|e_1 - \tilde{T}_k y \right\|_2$ for $y^{(k)}$
  Set $x^{(k)} = Q_k y^{(k)}$
  Check convergence
end for

representing "$Q_k R_k^{-1}$" to reduce work in each step. Also note that this method guarantees that the residual is nonincreasing (not true for CG as we optimize for $\|A\|$ there).

Summary of Krylov methods covered so far: (for $Ax = b$).

CG: $x^{(k)} = \text{argmin}_{x \in \mathcal{K}_k(A,b)} \|x - x^*\|_A$ for spd A, where $x^* = A^{-1}b$.

MINRES: $x^{(k)} = \text{argmin}_{x \in \mathcal{K}_k(A,b)} \|b - Ax\|_2$ for symmetric A.

Note the similarities in the structures of these algorithms:

- Run one step of Lanczos ($T_{mult}(A) + O(n)$)
- $x^{(k-1)} \to x^{(k)}$ carefully ($O(n)$) (maintain factorizations to make this cheap)
- Check convergence ($O(1)$)

Last key point is that storage required is $O(n)$ – no dependence on k. Every iteration costs the same. One small complication – q_k lose orthogonality due to the 3-term recurrence; this is mostly okay for $Ax = b$.

How long do these algorithms take to converge? We examine both of these algorithms for spd matrices (required for CG, makes analysis simpler for MINRES).

11.1 Convergence Analysis for Krylov Methods

Take P_k to be the polynomials of degree at most k and $\hat{P}_k$ to be the polynomials of degree at most k with $p(0) = 1$.

Note that $x^{(k)} \in \mathcal{K}_k(A,b) \iff x^{(k)} = p(A)b$ for some $p \in P_{k-1}$. Let $e^{(k)} = x^{(k)} - x^*$ denote the error vector and consider $\|e^{(k)}\|_A$.

$$\begin{aligned}\|e^{(k)}\|_A &= \min_{p \in P_{k-1}} \|p(A)b - A^{-1}b\|_A \\ &= \min_{p \in P_{k-1}} \|(p(A)A - I)A^{-1}b\|_A\end{aligned}$$

Note that if $x^{(0)} = 0$, $e^{(0)} = -A^{-1}b$. Additionally, if $p \in P_{k-1}$, $-p(A)A + I \in \hat{P}_k$ (the minus sign is okay because we're working with the norm).

$$= \min_{q \in \hat{P}_k} \|q(A)e^{(0)}\|_A$$

Because A is spd, we have a well-defined $A^{1/2} = V\Lambda^{1/2}V^T$. This allows us to rewrite the A-norm as:

$$= \min_{q \in \hat{P}_k} \|A^{1/2}q(A)e^{(0)}\|_2$$

Polynomials of A commute with powers of A so we have:

$$\begin{aligned}&= \min_{q \in \hat{P}_k} \|q(A)A^{1/2}e^{(0)}\|_2 \\ \|e^{(k)}\|_A &\leqslant \min_{q \in \hat{P}_k} \|q(A)\|_2 \|A^{1/2}e^{(0)}\|_2 \qquad \text{(n.b. this bound is sharp)} \\ &= \|e^{(0)}\|_A \min_{q \in \hat{P}_k} \|q(A)\|_2\end{aligned}$$

We can use the eigendecomposition of A to help simplify:

$$\begin{aligned}\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A} &\leqslant \min_{q \in \hat{P}_k} \|Vq(\Lambda)V^T\|_2 \\ &\leqslant \min_{q \in \hat{P}_k} \|q(\Lambda)\|_2 \\ &\leqslant \min_{q \in \hat{P}_k} \max_{i=1,\ldots,n} |q(\lambda_i)|\end{aligned}$$

It's easier to minimize this quantity when A's eigenvalues are clumped together – intuitively, we can take care of several eigenvalues at once with a single root of polynomial. If A has ℓ distinct eigenvalues, then CG can converge "exactly" in ℓ steps – we can pick a degree ℓ polynomial to annihilate every eigenvalue. In practice, this isn't quite the case as we don't have exact arithmetic (but pretty close for small ℓ). As a slightly more relaxed version, if A has ℓ small clusters of eigenvalues, you get close in ℓ steps (goodness depends upon tightness of clusters).

General result: Assume that the eigenvalues of A lie in interval $[\lambda_{min}, \lambda_{max}]$. We derive a bound on the error $e^{(k)}$ in terms of the condition number of A. Note that because A is spd, $\kappa(A) = \frac{\lambda_max}{\lambda_min}$.

Theorem 12.

$$\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A} \leqslant 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k$$

In other words, to get ε accuracy requires $\approx \sqrt{\kappa}\log\frac{1}{\varepsilon}$ steps. This upper bound is nice, but the real power of the method is through (implicitly) solving the optimization problem highlighted above.

Proof sketch: Note that the above work shows that, for any $q \in \hat{P}_k$:

$$\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A} \leqslant \max_{i=1,\ldots,n} |q(\lambda_i)|$$

Under the eigenvalue bounds assumption, pick a specific q to be uniformly small over this interval. We pick q such that $q(0) = 1$ and q is as small as possible on $[\lambda_{min}, \lambda_{max}]$ – smallest uniform upper/lower bounds on $[\lambda_{min}, \lambda_{max}]$. This turns out to be a scaled and shifted Chebyshev polynomial.

$$q(z) = T_k\left(\frac{2z-\lambda_{max}-\lambda_{min}}{\lambda_{max}-\lambda_{min}}\right)\Big/ T_k\left(\frac{-\lambda_{max}-\lambda_{min}}{\lambda_{max}-\lambda_{min}}\right)$$

where T_k is the k^{th} Chebyshev polynomial of the first kind.

$$T_k(z) = \cos(k\arccos(z)) \text{ on } [-1,1]$$

or alternatively, $T_0 = 1$, $T_1 = z$, and $T_{j+1}(z) = 2zT_j(z) - T_{j-1}(z)$

The key point here is that $|T_k(z)| \leqslant 1$ on $[-1,1]$ (grow quickly outside). This means that:

$$\left|\frac{1}{T_k\left(\frac{-\lambda_{max}-\lambda_{min}}{\lambda_{max}-\lambda_{min}}\right)}\right|$$

is our upper bound.

MINRES:

If $x^{(0)} = 0$, then $r^{(0)} = b$. Similar procedure to before, but this time we only need to factor out b:

$$\begin{aligned}
\|r^{(k)}\|_2 &= \min_{p\in P_{k-1}} \|b - Ap(A)b\|_2 \\
&= \min_{p\in P_{k-1}} \|(I - Ap(A))b\|_2 \\
&\leqslant \|r^{(0)}\|_2 \min_{q\in\hat{P}_k} \|q(A)\|_2 \\
\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2} &\leqslant \min_{q\in\hat{P}_k} \max_{i=1,\ldots,n} |q(\lambda_i)|
\end{aligned}$$

This bound is likewise sharp (there exist an A, b where it holds with equality) but generally stronger.

Why is MINRES analysis hard? If A is indefinite, we need our polynomial to be 1 at 0, but small on two intervals on either side. For more discussion on this, see Greenbaum 3.1.

Now suppose that A is no longer symmetric – $A \in \mathbb{R}^{n \times n}$ nonsingular. Since there's no A-norm, we can't really work with a variant of CG. Instead, we directly try to minimize the residual.

$$x^{(k)} = \underset{x \in \mathcal{K}_k(A,b)}{\operatorname{argmin}} \|b - Ax\|_2$$

GMRES – generalized minimum residuals. Historical fact: MINRES showed up in 1974-1976; GMRES not until 1986 (delay due to "not wanting to store so many Krylov vectors"). This procedure essentially mirrors MINRES, except we can't use Lanczos method. Instead, we replace it with Arnoldi method – instead of getting out a tridiagonal factorization, we maintain an upper Hessenberg matrix.

$$x^{(k)} = Q_k y^{(k)}, y^{(k)} = \underset{y}{\operatorname{argmin}} \|b - AQ_k y\|_2$$

Arnoldi: $AQ_k = Q_{k+1}\tilde{H}_k$ and $b = \|b\|Q_{k+1}e_1$. So $y^{(k)} = \operatorname{argmin}_y \|\|b\|e_1 - \tilde{H}_k y\|_2$, a $(k+1) \times k$ LS problem. But since this is dense, the cost per iteration depends on k – grows as we compute more steps.

11.2 GMRES

Given $A \in \mathbb{R}^{n \times n}$ nonsingular and $b \in \mathbb{R}^n$. Our chosen method is to essentially generalize MINRES to non-symmetric matrices. To a first-order approximation, this replaces Lanczos with Arnoldi as a subroutine.

Algorithm 11 GMRES (Conceptually)

Given A, b;
for $k = 1, 2, \ldots$ **do**
  One step of Arnoldi to get $Q_{k-1}, \tilde{H}_{k-1} \to Q_k, \tilde{H}_k$ (also get q_{k+1} implicitly)
  Solve $\min_y \left\| \|b\| e_1 - \tilde{H}_k y \right\|_2$ for $y^{(k)}$
  Set $x^{(k)} = Q_k y^{(k)}$
  Check convergence
end for

key relation for Arnoldi: $AQ_k = Q_{k+1}\tilde{H}_k$. Also note that $\|r^{(k)}\|_2 = \left\| \|b\| e_1 - \tilde{H}_k y \right\|_2$ because Q_k is orthogonal – residual is preserved.

Comparison with MINRES in time per iteration:

Step #	MINRES	GMRES
Lanczos/Arnoldi	$T_{mult}(A) + O(n)$	$T_{mult}(A) + O(kn)$
Finding $y^{(k)}$	$O(1)$	$O(k)$
Finding $x^{(k)}$	$O(n)$	$O(kn)$
Total Storage	3 vectors	k vectors

Note that GMRES has this dependence upon k – later steps are more expensive. To get the complexity for $y^{(k)}$, need to store and update a QR factorization of $\tilde{H}_k$.

Key idea now is to limit the impact of k. In practice, we use restarting. Fundamental idea: cap the number of iterations (i.e. length of for loop) to m; if we haven't reached convergence yet, restart with new b.

Algorithm 12 Restarted GMRES

Given $A, b, x^{(0)}, m$;
(1) Compute $r = b - Ax^{(0)}$
(2) Compute $d = \texttt{GMRES}(A, r, m)$ ▷ "Solve $Ad = r$" in m iterations
(3) $\hat{x} \leftarrow x^{(0)} + d$
if $\hat{x}$ is good enough **then**
  return $\hat{x}$
else
  $x^{(0)} \leftarrow \hat{x}$ and goto (1)
end if

Note that for this algorithm, convergence can stagnate (intuitively, don't run enough steps of `GMRES` to get it to converge).

Error analysis of GMRES:

We start with $\min_{x \in \mathcal{K}_k(A,b)} \|b - Ax\|_2$. Recall that this is equivalent to finding $\min_{p \in \hat{p}_k} \|p(A)b\|_2$ (same analysis as in MINRES). This implies:

$$\frac{\|r^{(k)}\|_2}{\|b\|_2} \leqslant \min_{p \in \hat{p}_k} \|p(A)\|_2$$

The latter depends upon the eigenvalues of A, but we pay additional cost for GMRES. We assume that $A = X \Lambda X^{-1}$ is diagonalizable.

$$\frac{\|r^{(k)}\|_2}{\|b\|_2} \leqslant \kappa(X) \min_{p \in \hat{p}_k} \max_{i=1,\ldots,n} |p(\lambda_i)|$$

Using the observation that $\kappa(X) = \|X\|\|X^{-1}\|$. Hence, the convergence of GMRES improves when X, i.e. eigenvectors of A, is well-conditioned. Moreover, when A is not symmetric, λ_i may not be real. It is difficult to pick $\hat{p}_k$ to be small on all such eigenvalues when they exist on the complex plane.

Key observation: Convergence depends upon spectrum of A. Generally, a few small clusters of eigenvalues is good – relatively low-degree polynomials with zeroes in those clusters. This works even when conditioning is bad: single rogue eigenvalue is bad for conditioning but only takes one more zero to cancel. Conversely, evenly spaced eigenvalues is bad.

Krylov methods are widely used because of Preconditioners. Goal is to get good solutions in a few iterations. Instead of solving $Ax = b$, solve $M^{-1}Ax = M^{-1}b$ for some nonsingular M. For direct methods, multiplying by M^{-1} doesn't buy us much – still end up with an $n \times n$ matrix. But for Krylov methods, we can pick M in ways that improve the spectrum of A.

Left preconditioning (there is also right preconditioning with $AM^{-1}y = b$, so $x = M^{-1}y$; and $M_1^{-1}AM_2^{-1}y = b$ two-sided conditioning).

Key consideration: We need to pick M such that $M^{-1}A$ is "nice" in spectrum and M^{-1} is easy to apply.

Two extreme points: $M = A$ gives nicest spectrum, $M = I$ easiest to apply. Hope is that something in between gives us something useful.

> Aside: If A is spd, we want M to be spd as well (so that we can run something like CG instead of GMRES):
>
> $$M = LL^T \implies M^{-1}A \sim L^{-1}AL^{-T}$$
>
> More generally, $L^{-1}AL^{-T}(L^Tx) = L^{-1}b$. We can rewrite this in such a way that we can execute CG with A, M^{-1}. Similarly, if A is symmetric, we want to pick a symmetric preconditioner.

As a general rule, no good "all-purpose" way to pick preconditioner. Often use information from the problem which A describes to help inform the choice.

- One generic option: Jacobi preconditioning

$$M = \begin{pmatrix} a_{11} & & 0 \\ & \ddots & \\ 0 & & a_{nn} \end{pmatrix}$$

  Take diagonal of A. If A is strictly diagonally dominant, might tell you info about the eigenvalues.

- Block Jacobi:

$$M = \begin{pmatrix} a_{11} & a_{12} & & 0 \\ a_{21} & a_{22} & & \\ & & \ddots & \\ 0 & & & a_{nn} \end{pmatrix}$$

  Blocks of diagonal, tridiagonal, etc. Performance depends upon the permutation of A.

- If A is sparse: so-called "incomplete factorizations."

$$A \approx LU$$

  and take $LU = M$.

- Coarse versions of the problem (e.g. motivation in solving PDEs, making fine grid coarser). Solve problem at coarse resolution and interpolate to finer.

11.3 Krylov Methods Demo

Recap: We have $Ax = b$ for nonsingular A. Several approaches to choose from:

- CG (A spd)
- MINRES ($A = A^T$)
- GMRES (any A)

For symmetric A, we can use Lanczos – otherwise, we need Arnoldi. Convergence depends upon spectrum of A. We can get upper bounds on convergence time/error depending upon eigenvalues but often loose.

We can precondition to accelerate convergence. Preconditioning consists of picking nonsingular M and solving $M^{-1}Ax = M^{-1}b$ instead. Recall that we want M to satisfy:

1. It is easy to solve linear systems with M.
2. Spectrum of $M^{-1}A$ is nice.

Because of this second condition, the best preconditioners are problem-dependent. But even for known problems, it is difficult to show that preconditioners are good in this regard systematically; mainly measured heuristically.

Many more Krylov methods exist.

For $Ax = b$:

- So-called biCG and biCGSTAB. bi-conjugate gradient methods (latter is stable). Instead of recurrence on A, recurrence on something loosely related to A^T.
- CGN – CG on normal equations, solving $A^TAx = A^Tb$.
- QMR – quasi-minimum residual methods. Instead of solving for min residual, loosely solve for something close to min.

For $\min_x \|b - Ax\|$ with $A \in \mathbb{R}^{m\times n}, m \geqslant n$.

- CGN – form $A^TAx = A^Tb$ and run CG.
- LSQR – least squares QR. Mathematically equivalent to CGN; but it never has to actually form the normal equations, better-behaved numerically.
- LSMR – mathematically equivalent to MINRES applied to $A^TAx = A^Tb$. Same caveats as above – don't need to form normal equations.

We will see Krylov methods again for eigenvalue/eigenvector problems.

Aside: Say we want to solve $\min_x \|b - Ax\|_2 + \lambda\|x\|_2$ for $m \geqslant n$. Recall from hw2 that this can be seen as solving a larger least squares problem:

$$\min_x \left\| \begin{pmatrix} b \\ 0 \end{pmatrix} - \begin{pmatrix} A \\ \sqrt{\lambda} I \end{pmatrix} x \right\|_2 \iff (A^TA + \lambda I)x = A^Tb$$

Krylov subspaces are "shift invariant" – $\mathcal{K}_k(A^TA, b) = \mathcal{K}_k(A^TA + \lambda I, b)$. So even if we want to solve for different λ, only need to run Lanczos once to get correct space. Each value of λ only requires solving a $k \times k$ linear system.