Math 4260/ CS 4220 — Numerical Analysis: Linear and Nonlinear Problems

Derek Lim

Spring 2019

Instructor: Anil Damle

Course Description: Introduction to the fundamentals of numerical linear algebra: direct and iterative methods for linear systems, eigenvalue problems, singular value decomposition. In the second half of the course, the above are used to build iterative methods for nonlinear systems and for multivariate optimization. Strong emphasis is placed on understanding the advantages, disadvantages, and limits of applicability for all the covered techniques. Computer programming is required to test the theoretical concepts throughout the course.

Textbook: *A First Course in Numerical Methods* by Ascher and Greif

Lecture 1: Introduction (2/1/19)

A **problem** f is a map $f : X \to Y$, where X is a vector space of input data with a norm, and Y is the vector space of solutions with a norm.

Example: for a fixed invertible A, given b, solve $Ax = b$. Then our solution at b is $f(b) = A^{-1}b$.

Characterize a problem f

- a problem f at point x is **well-conditioned** if small changes in the input x result in small changes in the solution $f(x)$.
- a problem f at point x is **ill-conditioned** if there exist some small changes in x that result in large changes in $f(x)$.

absolute condition number:

let δx be small changes in x, and $\delta f = f(x + \delta x) - f(x)$.

$$\hat{\kappa} = \lim_{\delta \to 0} \sup_{\|\delta x\| \leq \delta} \frac{\|\delta f\|}{\|\delta x\|}$$

If f is differentiable, and letting $J(x)$ be the Jacobian of f at x, then

$$\hat{\kappa}(f(x)) = \|J(x)\|$$

relative condition number if f differentiable at x:

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

example: $f(x) = \frac{x}{2}$. Then $J(x) = \frac{1}{2}$. Also, we get $\kappa = \frac{1/2}{x/2/x} = 1$

condition number of a matrix:

problem: A is fixed. the problem is to compute the matrix vector product, so $f(x) = Ax$. Then we have that $J(x) = A$. We have that $\kappa(x) = \frac{\|A\|}{\|Ax\|/\|x\|}$. We make this a characterstic of just the matrix by removing dependence on x.

assume that A is invertible. observe that $\frac{\|x\|}{\|Ax\|} \leq \left\|A^{-1}\right\|$ then we have that

$$\kappa \leq \|A\|\left\|A^{-1}\right\| \quad \forall x$$

define $\kappa(A) = \|A\|\left\|A^{-1}\right\|$. this number shows up in upper bounds for many types of problems including:

- fix A, given b, compute the solution to $Ax = b$.
- fix b, given square, nonsingular A, compute $A^{-1}b$. note, $\kappa(A)$ is the actual condition number of this problem.

thus, we define the **condition number of a matrix** A as $\kappa(A) = \|A\|\left\|A^{-1}\right\|$ and $\kappa(A) = \infty$ for non-invertible square A. equivalently, we have $\kappa(A) = \frac{\sigma_1}{\sigma_n}$, interpreted as ratio of largest axis of ellipse to smallest axis, where the ellipse is the image of the $n-$dimensional unit ball under A. note that an orthogonal matrix maps a ball to a ball.

Lecture 2: Floating Point Numbers (2/4)

Can only represent finitely many real numbers. Will have

- Smallest number
- Largest number
- Gaps between the numbers that can be represented

Now, our representation can have any base. We fix base 2. Our number systems will be completely characterized by parameters

- t — precision
- L — smallest exponent
- U — largest exponent

A **floating point number system** is the set of all numbers (plus 0) of the form

$$\begin{aligned} \pm 1.d_1 d_2 \cdots d_{t-1} \times 2^e \\ d_i \in \{0,1\} \\ e \in [L,U] \end{aligned}$$

note that

$$1.d_1 d_2 \cdots d_{t-1} = 1 + \frac{d_1}{2} + \frac{d_2}{4} + \ldots + \frac{d_{t-1}}{2^{t-1}}$$

1 is fixed for uniqueness of representation. For instance in base 10 we have $0.12 \times 10^3 = 1.2 \times 10^2$

Implications:

- distance between floating point numbers is only constant for a fixed e
- there are the same number of floating point numbers in $[1,2)$ and $[2,4)$

Characterized by **machine precision**, denoted $\epsilon_{machine}$ or μ.

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

which is half the distance from 1 to the next closed floating point number greater than 1.

IEEE 754 double. Uses 64 bits per floating point number. Has 1 bit sign, 52 bits for mantissa at $t = 53$, and 11 bits for the exponent. Contrains $e \in [-1022, 1023]$. Note $2^1 1 = 2048$ so two exponents are reserved.

$\mu \approx 10^{-16}$. Our numbers lie in the range $[10^{-308}, 10^{308}]$, but numbers at the higher end are about $10^2 92$ apart from each other.

An exponent of all 1 or all 0 designate special objects. For instance, used to represent 0, NaN, $\pm\infty$, subnormal numbers.

We choose our mathematical model for floating point:

- ignore underflow and overflow, i.e. trying to represent too small or too big a number
- representation: for all $x \in \mathbb{R}$, there exists an ϵ with $|\epsilon| \leq \mu$ such that $fl(x) = x(1+\epsilon)$, where $fl(x)$ is the nearest floating point number to x.
- arithmetic: for all floating point numbers a and b, there exists an ϵ with $|\epsilon| < \mu$ such that $fl(a **b)$ is equal to $(a **b)(1+\epsilon) \in \mathbb{R}$, where $**$ is any of the four $+, -, \cdot, /$ arithmetic operations.

When subtracting two nearby floating pont numbers to get a much smaller answer, we lose precision/ significant digits. For instance, consider weighing a captain of a boat by weighing a boat and then weighing the captain with the boat then subtracting. Big numbers to small numbers make you lose precision. Called **catastrophic cancellation**.

Lecture 3: Stability (2/6)

An algorithm for f is a map $\tilde{f} : X \to X$.

- at the very least, converts x to floating point.
- run algorithm $f(x)$
- can only output floating point numbers

We want an algorithm to be accurate.

$$\textbf{absolute accuracy} \quad \left\| \tilde{f}(x) - f(x) \right\|$$

$$\textbf{relative accuracy} \quad \frac{\left\| \tilde{f}(x) - f(x) \right\|}{\|f(x)\|}$$

we may hope to have an *accurate algorithm,* or one in which

$$\frac{\left\| \tilde{f}(x) - f(x) \right\|}{\|f(x)\|} = O(\mu) \text{ as } \mu \to 0 \quad \text{for all } x \in X$$

where g is $O(\mu)$ if $g \leq C\mu$ for some $C > 0$, and all small enough μ.

This is too much to ask for in ill conditioned problems. Instead, we ask for stability.

$\tilde{f}$ is **stable** for a problem f if for all $x \in X$, we have

$$\frac{\left\| \tilde{f}(x) - f(\tilde{x}) \right\|}{\|f(\tilde{x})\|} = o(\mu) \text{ as } \mu \to 0$$

$$\text{for some } \tilde{x} \text{ with } \frac{\|\tilde{x} - x\|}{x} = o(\mu) \text{ as } \mu \to 0$$

an algorithm $\tilde{f}$ is **backwards stable** for a problem f if for all $x \in X$, we have $\tilde{f}(x) = f(\tilde{x})$ for some $\tilde{x}$ with $\frac{\|\tilde{x}-x\|}{\|x\|} = o(\mu)$ as $\mu \to 0$.

For instance, computing uv^T is not backwards stable in the sense that algorithms to compute if will lead to outputs with not exactly rank 1.

$o(\mu)$ constant can depend on m, n etc. It is often benign such as n, mn, n^2. However, in bad cases, the constant can be exponenetial 2^n in the problem size.

Example: subtraction is backwards stable

We have data $x_1, x_2 \in \mathbb{R}$. $f(x_1, x_2) = x_1 - x_2$.

$\tilde{f}(x_1, x_2) = fl(x_1) \ominus fl(x_2)$ where $\ominus$ is the floating point subtraction.

We know we have $|\epsilon_1|, |\epsilon_2| < \mu$ such that $fl(x_i) = x_i(1 + \epsilon_i)$.

Thus, we have for fixed x_1, x_2 that $\tilde{f}(x_1,x_2) = x_1(1+\epsilon_1) \ominus x_2(1+\epsilon_2)$. Moreover, we have that ϵ_3 with $|\epsilon_3| \leq \mu$ such that

$$\begin{aligned}\tilde{f}(x_1,x_2) &= [x_1(1+\epsilon_1) - x_2(1+\epsilon_2)](1+\epsilon_3) \\ &= x_1(1+\epsilon_3+\epsilon_1+\epsilon_1\epsilon_3) - x_2(1+\epsilon_2+\epsilon_3+\epsilon_2\epsilon_3) \\ &= x_1(1+\epsilon_4) - x_2(1+\epsilon_5) \quad \text{we define}\end{aligned}$$

We have that $\epsilon_4 = o(\mu)$ and $\epsilon_5 = o(\mu)$. Thus, we have that

$$\tilde{f}(x_1,x_2) = f(x_1(1+\epsilon_4), x_2(1+\epsilon_5))$$

Now, we show that backwards stability $\implies$ accuracy.

Suppose we use a backwards stable $\tilde{f}$ to solve f with condition number $\kappa(x)$. Then

$$\frac{\left\|\tilde{f}(x) - f(x)\right\|}{\|f(x)\|} = o(\kappa(x)\mu)$$

proof sketch:

$$\begin{aligned}\frac{\left\|\tilde{f}(x) - f(x)\right\|}{\|f(x)\|} &= \frac{\|f(\tilde{x}) - f(x)\|}{\|f(x)\|} \quad \text{by backwards stability, for some } \tilde{x} \\ \kappa(x) &\approx \frac{\|\delta f\|}{\|f\|} \Big/ \frac{\|\delta x\|}{\|x\|}\end{aligned}$$

note that $\tilde{x}$ has $\frac{\|\tilde{x}-x\|}{\|x\|} = o(\mu)$. Thus, we have the two forms in the κ expression above.

Lecture 4: LU Solve (2/8)

We focus on the problem of finding $x \in \mathbb{R}^n$ satisfying

$$Ax = b$$

where $A \in M_n(\mathbb{R})$ is full rank, and $b \in \mathbb{R}^n$.

Algorithm 1 Backward substitution

for $k = n : -1 : 1$ **do**

$$x_k = \frac{b_k - \sum_{j=k+1}^{n} A_{k,j} x_j}{A_{k,k}}$$

end for

has cost $\sum_{k=1}^{n}(2(n-k)+1) = O(n^2)$.

Backwards substitution is backwards stable. Given R, b and a computed $\tilde{x}$, we have $\tilde{x}$ satisfies $(R+\delta R)\tilde{x} = b$. For some triangular δR with $\frac{\|\delta R\|}{\|R\|} = O(\mu)$

Now, given A, we want to find L and U such that $A = LU$. Add permutation matrix P later. We want to construct

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

Algorithm 2 LU Factorization

for $k = 1 : n-1$ **do**
  for $r = k+1 : n$ **do**
    $A[r,k] = \frac{A[r,k]}{A_{k,k}}$
    $A[r,r] = A[r,r] - A[r,k]A[k,r]$
  end for
end for

$L_k = I - l_k e_k^T$ then

$$L_k x_k = x_k - l_k (x_k)_k$$

nicely, $L_k^{-1} = I + l_k e_k^T$ and $L_k^{-1} L_{k+1}^{-1} = I + l_k e_k^T + l_{k+1} e_{k+1}^T$.

Loops over columns, choose rows that matter, compute l_k, and then apply L_k. At the end of this algorithm, A has elements of L in lower triangle, and U on diagonal and upper triangle.

Lecture 5: LU (2/11)

Equivalent algorithm,

Algorithm 3 Equivalent LU algorithm

$U = A, L = I$
for $k = 1 : n-1$ **do**
  for $j = k+1 : n$ **do**
    $l_{jk} = \frac{u_{j,k}}{u_{k,k}}$
    $u[j, k:n] = u[j, k:n] - l_{j,k} \cdot u[k, k:n]$
  end for
end for

only use upper triangular part of U after this algorithm is done, and essentially set the rest to zero (only use upper triangle to solve $Ax = b$).

Cost of LU factorization:

$$\sum_{k=1}^{n} \sum_{j=k+1}^{n} [1 + 2(n-k+1)] \approx \sum_{k=1}^{n} (n-k)^2$$

$$= O(n^3)$$

there is no LU decomposition of $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. likewise, do not want to divide by small ϵ in $\begin{pmatrix} \epsilon & 1 \\ 1 & 0 \end{pmatrix}$

Thus, we use pivoting.

LU with partial pivoting, ensure that all entries of L have $|L_{i,j}| \leq 1$.

Say $|x_{i,k}| \geq |x_{j,k}|$ for $k \leq j \leq n$. Then P_k swaps rows k and i. Then proceed as before, and compute L_k to zero out rows. Note that all entries of L_k are at most 1, since the largest entry in the column is on the diagonal and hence the divisor.

We compute permutation matrices $P_1, \dots P_{n-1}$ and lower triangular matrices $L_1, \dots, L_{n-1}$ such that

$$L_{n-1}P_{n-1}\cdots L_2P_2L_1P_1A = U$$

let L_i' be L_i with entries in column i below the diagonal permuted:

$$L_i' = P_{n-1}\cdots P_{i+1}L_iPP_{i+1}^T\cdots P_{n-1}^T$$

then we have

$$L_{n-1}'\cdots L_1'P_{n-1}\cdots P_1A = U$$

let $\tilde{P}_i = P_{n-1}\cdots P_{i+1}$. Then we have that $L_i' = \tilde{P}_iL_i\tilde{P}_i^T$. Note that $\tilde{P}_i$ only permutes the $i+1$ and above rows and columns, and then multiplying by its inverse on the right inverts these changes on the $i+1$ and above rows and columns in some sense.

Thus, we get that

$$PA = LU$$

Algorithm:

Algorithm 4 LU with Partial Pivoting

```
U = A, L = I, p = 1 : n, p(i) = i
for k = 1 : n − 1 do
    i = argmax_i |u_{i,k}|
    swap u[k, k : n] and u[i, k : n]
    swap L[k, 1 : k − 1] and L[i, 1 : k − 1]
    swap p[i] and p[j]
    for j = k + 1 : n do
        l_{j,k} = u_{j,k} / u_{k,k}
        u[j, k : n] = u[j, k : n] − l_{j,k} u[j, k : n]
    end for
end for
```

Lecture 6: LU Stability, Cholesky (2/13)

Given A nonsingular, with an LU facotrization without pivoting, if the process does not break down and our floating point axioms hold, then the computed $\tilde{L}, \tilde{U}$ satisfy

$$\tilde{L}\tilde{U} = A + \delta A$$

for some δA with

$$\frac{\|\delta A\|}{\|L\|\|U\|} = O(\mu)$$

note this is not quite a backwards stability result, since the denominator needs to be the norm of the actual input A, not the multiplied L and U norms. This bound is tight in some sense, so that LU factorization is not backwards stable.

For LU with partial pivoting, given A compute $\tilde{P}$, $\tilde{L}$, $\tilde{U}$. Then

$$\tilde{L}\tilde{U} = \tilde{P}A + \delta A$$

$$\frac{\|\delta A\|}{\|A\|} = O(\rho\mu)$$

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

furthermore, if $|l_{i,j}| < 1, i, j$, then for small enough μ, $\tilde{P} = P$. In other words, fif there are no "ties" (each column has unique largest element in bottom half), then for enough precision, we have the computed permuation matrix is the exact permutation matrix.

Note, ρ can be 2^{n-1}. So LU with partial pivoting is mathematically backwards stable, but ρ can be very big.

Now, suppose A is symmetric positive definite. Could compute $A = LU$.

Theorem 1 (Existence of Cholesky factorization)**.** *Every symmetric positive definite A has a unique Cholesky factorization. We can find a R upper triangular with $R_{i,i} > 0$ such that*

$$A = R^T R$$

Proof. If A is 1×1, then the Cholesky factorization is the positive square root $R = +\sqrt{A_{1,1}}$

Assume we can construct a Cholesky factorization of size $n-1 \times n-1$. Let A be $n \times n$ spd. Write A as

$$A = \begin{pmatrix} a_{1,1} & w^T \\ w & K \end{pmatrix}$$

$w \in \mathbb{R}^{n-1}$, and $K \in \mathbb{R}^{n-1 \times n-1}$. We can write A as

$$A = \begin{pmatrix} \sqrt{a_{1,1}} & \\ w/\sqrt{a_{1,1}} & I \end{pmatrix} \begin{pmatrix} 1 & \\ & K - \frac{ww^T}{a_{1,1}} \end{pmatrix} \begin{pmatrix} \sqrt{a_{1,1}} & w^T/\sqrt{a_{1,1}} \\ & I \end{pmatrix}$$

Now, we know that K is symmetric positive definite since A is. We show that $K - \frac{ww^T}{a_{1,1}}$ is positive definite (obviously symmetric). Call the left matrix above R_1^T, and thus the right R_1. Then the middle matrix is $R_1^{-T} A R_1^{-1}$. Note that for $x \neq 0$

$$x^T R_1^{-T} A R_1^{-1} x > 0$$

since A is positive definite, and $R_1^{-1} x \neq 0$ since R_1 is nonsingular. This implies that the middle matrix is positive definite, so that the lower right submatrix is indeed positive definite. We factor $K - \frac{ww^T}{\sqrt{a_{1,1}}} = \tilde{R}^T \tilde{R}$. Then we have

$$A = \begin{pmatrix} \sqrt{a_{1,1}} & \\ w/\sqrt{a_{1,1}} & I \end{pmatrix} \begin{pmatrix} 1 & \\ & \tilde{R}^T \end{pmatrix} \begin{pmatrix} 1 & \\ & \tilde{R} \end{pmatrix} \begin{pmatrix} \sqrt{a_{1,1}} & w^T/\sqrt{a_{1,1}} \\ & I \end{pmatrix}$$

So that by defining $R^T = \begin{pmatrix} \sqrt{a_{1,1}} & \\ w/\sqrt{a_{1,1}} & \tilde{R}^T \end{pmatrix}$, we know that R is upper triangular with positive diagonal entries and $A = R^T R$. □

In algorithm form, given A spd,

The cost of this is $O(n^3)$, but about $\frac{1}{2}$ the work of LU as seen by working out the constants.

Algorithm 5 Cholesky Factorization

$R = A$
for $k = 1 : n$ **do**
  for $j = k+1 : n$ **do**
    $R[j, j:n] = R[j, j:n] - R[k, j:n]\frac{r_{k,j}}{r_{k,k}}$
  end for
  $R[k, k:n] = R[k, k:n]/\sqrt{r_{k,k}}$
end for
only use the upper triangular part of R.

$$\|R\|_2 = \|A\|_2^{1/2} \quad \text{prove by SVD}$$

stability: given A spd, compute cholesky with floating point axioms. Then the computed $\tilde{R}$ satisfies

$$\tilde{R}^T \tilde{R} = A + \delta A$$
$$\frac{\|\delta A\|}{\|A\|} = O(\mu)$$

so that cholesky factorization is **backwards stable**.

Note that the best we can say in forward error analysis, about $\tilde{R} - R$, is

$$\frac{\left\|\tilde{R} - R\right\|}{\|R\|} = O(\kappa(A)\,\mu)$$

so can be far from true factorized R if A is ill-conditioned.

Lecture 7: Least Squares (2/15)

Solving $Ax = b$ for A $n \times m$ and $n > m$ and A full column rank:

$$\min_{x \in \mathbb{R}^n} \frac{1}{2}\|b - Ax\|_2^2$$

example: say given n data points $(x_i, y_i), x_i, y_i \in \mathbb{R}$ with x_i distinct. Want to find coefficients α_i such that $f(x) = \alpha_1 + \alpha_2 x + \ldots + \alpha_n x^{n-1}$ exactly interpolates the data points $(f(x_i) = y_i)$. Can be solved since is a square non-singular system.

Perhaps better: for some fixed $m < n$, find coefficients $c_1, \ldots, c_m$ such that the polynomial $p(x) = c_1 + c_2 x + \ldots + c_m x^{m-1}$ has $p(x_i) \approx y_i$. In particular, want

$$\min_{c_1,\ldots,c_m} \frac{1}{2}\sum_{i=1}^{n} |p(x_i) - y_i|^2$$

let A be the Vandermonde matrix

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{m-1} \\ \vdots & \vdots & \vdots & \ddots & \\ 1 & x_n & x_n^2 & \ldots & x_n^{m-1} \end{pmatrix}$$

Then the minimization problem is equivalent to solving

$$\min_{c\in\mathbb{R}^m} \frac{1}{2}\|y-Ac\|_2^2$$

Aside: projectors. A *projector* is a square matrix P that satisfies $P^2 = P$. Say $v \in imP$. Then there exists x such that $v = Px$. Hence, we have $Pv = PPx = Px = v$ (projectors do not change vectors in their own range).

There are *oblique projectors* and *orthogonal projector*. We will only concern ourselves with *orthogonal projectors*. A projector is *orthogonal* if $P = P^T$. Note that these are not orthogonal matrices.

Orthogonal projectors project vectors V onto a subspace $S_1 = imP$ along a direction in a subspace $S_2 = \ker P$ that is orthogonal to S_1. We have that Pv is the closest point in imP to v in $\|\cdot\|_2$. Moreover, we have that $v - Pv \in \ker P$, so that $v - Pv$ is orthogonal to Pv.

For any projector P, define its *complementary projector* $I - P$, which is the orthogonal projector onto $\ker P$. Then we can decompose any vector v as

$$v = v_1 + v_2 = Pv + (I-p)v$$

where $v_1 \in imP$ and $v_2 \in \ker P$ are orthogonal.

Now, lets say we have a subspace S_1 of dimension k and want to build a projector onto it. Given vectors $q_1, \ldots, q_k$ orthonormal basis for S_1, let $Q = \begin{pmatrix} q_1 & \ldots & q_k \end{pmatrix}$. Then the **unique** orthogonal projector P_{S_1} onto S_1 can be written as $P_{S_1} = QQ^T$. Note that this is the SVD $P_{S_1} = QIQ^T$, so that P_{S_1} has rank k. The product with any vector $Q(Q^Tv)$ has information.

Can restate $\min_x \frac{1}{2}\|b-Ax\|_2^2$ as find $z \in imA$ closest to b. Then find x such that $Ax = z$ (which has a solution since $z \in \mathrm{im}A$).

Thus, we want to find the projection Pb of b onto $\mathrm{im}A$. The residual $r = b - Ax$ is, if P an orthogonal projector, is orthogonal to $\mathrm{im}A$.

Theorem 2. *Given A, an $n \times m$ full column rank matrix, with $n \geq m$, and $b \in \mathbb{R}^n$, then a vector x minimizes $\|r\|_2^2 = \|b-Ax\|_2^2$ if and only if*

$$r \perp \mathrm{im}A \iff A^Tr = 0$$

or, equivalently,

$$A^TAx = A^Tb$$

or, equivalently,

$$Pb = Ax \quad P \text{ an orthogonal projector onto } \mathrm{im}A$$

Can solve normal equations with cholesky. Costs about $m^2n + \frac{1}{3}m^3$. Drawback is in conditioning:

$$\kappa(A^TA) = \kappa(A)^2$$

Lecture 8: QR factorizations (2/18)

A *QR factorization* is to find an orthongal Q and an upper-triangular R such that $A = QR$.

Note: for all square $n \times n$ A, there exists an orthogonal matrix Q and an upper triangular matrix R such that $A = QR$.

- if A is non-singular, then so is R
- almost unique, is unique if $r_{i,i} \geq 0$
- easy to solve $Ax = b$ with $A = QR$

  $y = Q^T b$

  $Rx = y$

Now, for $n > m$, and for all $A \in \mathbb{R}^{n \times m}$, there exists an orthogonal $Q \in \mathbb{R}^{n \times n}$ and upper triangular $R_1 \in \mathbb{R}^{m \times m}$ such that

$$A = QR \qquad R \in \mathbb{R}^{n \times m} \qquad R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

Now, letting $Q = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix}$, where $Q_1 \in \mathbb{R}^{n \times m}$, so that

$$A = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R_1 \\ 0 \end{pmatrix} = Q_1 R_1$$

Thus, given any $A \in \mathbb{R}^{m \times m}$ where $n > m$, then there exists $Q \in \mathbb{R}^{n \times m}$ with orthonormal columns and an upper triangular $R \in \mathbb{R}^{m \times m}$ such that $A = QR$.

Note we have $\text{im}A = \text{im}Q$. Thus, columns of Q are an orthonormal basis for the image of A.

Now, suppose that $A = QR$. Then we have by multiplication that

$$\begin{aligned} a_1 &= r_{1,1} q_1 \\ a_2 &= r_{1,2} q_1 + r_{2,2} q_2 \\ &\vdots \quad \vdots \end{aligned}$$

in words, for all $j \in [m]$, we have $\text{span}(a_1, \ldots, a_j) = \text{span}(q_1, \ldots, q_j)$

we can set

$$\begin{aligned} q_1 &= \frac{a_1}{\|a_1\|} \\ v_2 &= a_2 - (q_1^T a_2) q_1 \quad \text{removes part of } a_2 \text{ in direction of } q_1 \\ q_2 &= \frac{v_2}{\|v_2\|} \\ &\vdots \quad \vdots \\ v_j &= a_j - (q_1^T a_j) q_1 - \ldots - (q_{j-1}^T a_j) q_{j-1} \\ q_j &= \frac{v_j}{\|v_j\|} \\ &\vdots \quad \vdots \end{aligned}$$

Gram-Schmidt algorithm, given $A \in \mathbb{R}^{n \times m}$ with full column rank,

is good mathematically but can be numerically problematic.

Algorithm 6 Gram-Schmidt

for $j = 1, \ldots, m$ **do**
  $v_j = a_j$
  for $i = 1 : j-1$ **do**
    $r_{i,j} = q_i^T a_j$
    $v_j = v_j - r_{i,j} q_i$
  end for
  $r_{j,j} = \|v_j\|_2$
  $q_j = \frac{v_j}{r_{j,j}}$
end for

Let $P_i = q_i q_i^T$ then

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

$$v_j = \left(I - \begin{pmatrix} q_1 & \ldots & q_{j-1} \end{pmatrix} \begin{pmatrix} q_1^T \\ \vdots \\ q_{j-1}^T \end{pmatrix} \right) a_j$$

In **modified Gram-Schmidt**, we do not leave all of the projectors to the end.

$$v_j = (I - P_{j-1}) \Big[\cdots [(I - P_1) a_j] \cdots \Big]$$

Algorithm 7 Modified Gram-Schmidt

for $i = 1 : m$ **do**
  $v_i = a_i$
end for
for $i = 1 : m$ **do**
  $r_{i,i} = \|v_{i,i}\|_2$
  $q_i = \frac{v_i}{r_{i,i}}$
  for $j = i+1 : n$ **do**
    $r_{i,j} = q_i^T v_j$
    $v_j = v_j - r_{i,j} q_i$
  end for
end for

Lecture 9: 2-20-19

Can view Gram-Schmidt or Modified-Gram-Schmidt as building $R_1, \ldots, R_m$ such that

$$AR_1 \cdots R_m = Q$$

so the process is like building R to make A orthogonal.

Today, we will instead cook-up a sequence of orthogonal matrices $Q_1, \ldots, Q_m$ such that

$$Q_1 \cdots Q_m A = R$$

like in LU, but now the transformations are restricted to orthogonal ones and not lower-triangular ones. Also, A can be rectangular now.

We use Q to reduce A to R one column at a time.

$$A = \begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix} \to Q_1 A = \begin{pmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{pmatrix} \to Q_2 Q_1 A = \begin{pmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{pmatrix} \to Q_3 Q_2 Q_1 A = \begin{pmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \end{pmatrix}$$

$$Q_2 = \begin{pmatrix} 1 & \\ & F_2 \end{pmatrix} \qquad Q_3 = \begin{pmatrix} I & \\ & F_3 \end{pmatrix}$$

where Q_1, Q_2, and Q_3 are orthogonal.

Thus, we have

$$Q_k = \begin{pmatrix} I & \\ & F_k \end{pmatrix}$$

where I is $(k-1) \times (k-1)$, and F_k is $(n-k+1) \times (n-k+1)$. We want F_k such that for some vector $y \in \mathbb{R}^{n-k+1}$, we have $F_k y = \begin{pmatrix} \times \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

Thus, we have a subtask. For $y \in \mathbb{R}^n$, construct an orthogonal $n \times n$ matrix F such that $Fy = \pm\|y\|_2 e_1$. Geometrically, want to reflect/ rotate y onto $\|y\| e_1$ on the first axis. Rotation is hard, so we reflect.

We want to reflect over the plane orthogonal to $v = \|y\| e_1 - y$. We have that $F = I - 2\frac{vv^T}{v^T v}$ achieve this goal. $Fy = y - 2(\frac{vv^T}{v^T v} y)$.

Can have catastrophic cancellation in making v if y is close to $\pm\|y\| e_1$. Thus, we can take the sign to be the one who makes $\pm\|y\| e_1$ further from y.

These F are called Householder reflectors.

Thus, given y, compute $v = \text{sign}(y_1)\|y\| e_1 + y$. Then $F = I - 2\frac{vv^T}{v^T v}$ satisfies $Fy = \begin{pmatrix} \pm\|y\|_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

We can compute the product of F with y in $O(n)$ time, so we do not really need to form F. We just keep the v.

Algorithm 8 QR Factorization

Given $A \in \mathbb{R}^{n \times m}$
for $k = 1, \ldots, m$ **do**
  $x = A[k:n, k]$
  $v_k = \text{sign}(x_1)\|x\|_2 e_1 + x$
  $v_k = \frac{v_k}{\|v_k\|_2}$
  $A[k:n, k:m] = A[k:n, k:m] - 2v_k(v_k^T A[k:n, k:m])$
end for
A contains R, and $Q_k = \begin{pmatrix} I_{k-1,k-1} & \\ & I - 2v_k v_k^T \end{pmatrix}$

At the end, get $Q_m \cdots Q_1 A = R$, so that $A = Q_1 \cdots Q_m R$, since Q_k is symmetric, and $Q = Q_1 \cdots Q_m$.

Lecture 10: 2-22-19

Above *Householder QR* costs $O(nm^2)$. For square matrix, asymptotically same cost as LU, but in practice somewhat slower. Same complexity as Gram-Schmidt and Modified Gram-Schmidt.

Often do not need to form the actual matrix Q. Thus, we have algorithms to apply Q and/or Q^T.

Algorithm 9 Compute $Q^T b$

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

If we want to compute $Q[1:m]^T b$, can just look at first m entries after using above algorithm.

If only want to compute $Q[:, 1:m] \cdot y$, put y at top and apply above by having $x = \begin{pmatrix} y \\ 0 \end{pmatrix}$

Algorithm 10 Compute Qx

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

Suppose given $Q \in \mathbb{R}^{n\times m}$ and $R \in \mathbb{R}^{m\times m}$ such that $A = QR$. Assume A full column rank so R is nonsingular.

Recall x solves $\min_x \frac{1}{2}\|b - Ax\|_2^2$ if and only if $Pb = Ax$, where P orthogonal projector onto im$A$.

Q's columns are an orthonormal basis for the columns of A. Thus, $P = QQ^T 4$ is an orthogonal projector onto im$A$. Hence, solution x satisfies $QQ^T b = Ax$. We write

$$\begin{aligned} QQ^T b &= QRx \\ Q(Q^T b - Rx) &= 0 \\ \iff Q^T b - Rx &= 0 \quad Q \text{ has trivial kernel} \\ \iff Q^T b &= Rx \end{aligned}$$

Solving for x is easy, since we have an $m \times m$ upper triangular system that we can solve by back sub.

Thus, to solve the least squares problem, we have a routine: decompose $A = QR$, then compute $Q^T b$, then solve $Q^T b = Rx$ for x.

Has cost $O(nm^2) + O(nm) + O(m^2)$, Same asymptotically but somewhat slower than normal equations. However, much more stable than normal equations.

Can also solve least squares given a reduced SVD of A, $A = U\Sigma V^T$. Can be a better choice that QR for ill-conditioned A. Note SVD algorithms are iterative, and not direct methods.

Let's talk about sensitivity of the least squares problems to perturbations in A and b. Suppose we have A with $n \geq m$, and b. Call the projection of b onto im$A$ as $y = Ax$.

Define

$$\kappa(A) = \frac{\sigma_1}{\sigma_m}$$

$$\theta = \cos^{-1}\frac{\|y\|}{\|b\|}$$

$$\eta = \frac{\|A\|\|x\|}{\|Ax\|}$$

$$\text{Have some bounds:} \quad 1 \leq \kappa(A) \qquad 0 \leq \theta \leq \frac{\pi}{2} \qquad 1 \leq \eta \leq \kappa(A)$$

We have input/data A, b and solutions x, y. Fix A full column rank, and b.

Here we have 2-norm relative condition numbers for sensitivity of x and y with respect to perturbations in A and b.

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

means how does y and x change when you change b and A.

Lecture 11: 2-27-19: Givens rotations

Suppose we have

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

We only need to zero out one entry, but householder could cause lots of more nonzeros to be made.

Consider a 2×2 matrix G and vector $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$

Want to construct G given x such that

$$Gx = \begin{pmatrix} \|x\| \\ 0 \end{pmatrix}$$

Can accomplish this with

$$G = \begin{pmatrix} c & -s \\ s & c \end{pmatrix}$$

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

$$s = \frac{-x_2}{\sqrt{x_1^2 + x_2^2}} \quad c = \frac{x_1}{\sqrt{x_1^2 + x_2^2}}$$

Then $Gx = \frac{\|x\|}{0}$.

Now, to use it for constructing QR, consider

$$G_{i,j}(x) = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & \dots & 0 & \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \\ 0 & 0 & 1 & \dots & 0 & \dots & 0 & \\ 0 & 0 & 0 & c & \dots & -s & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \\ 0 & 0 & 0 & s & \dots & c & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \\ 0 & 0 & 0 & 0 & \dots & 0 & \dots & 1 \end{pmatrix}$$

where the nonzeros are in the i and j rows and columns.

applying $G_{i,j}(x)$ to A:

$$G_{i,j}(x)A = \begin{cases} A[k,:] & \text{for } k \neq i, k \neq j \\ \begin{pmatrix} A[i,:] \\ A[j,:] \end{pmatrix} & = G(x) \begin{pmatrix} A[i,:] \\ A[j,:] \end{pmatrix} \end{cases}$$

Only may introduce nonzeros in upper triangle.

Too many matrices I'm not tryna type all of them.

In matrix of above example, let $Q_1 = G_{1,n}\begin{pmatrix} A_{1,1} \\ A_{n,1} \end{pmatrix}$. When solving $\min_x \frac{1}{2}\|b - Ax\|_2^2$, if $\kappa(A)$ is large then small perturbations in b can cause large perturbations in x.

Consider b as data, and A as factors/model, and x are coefficients. If A is ill-conditioned, there are many approximately equally good models. For these ill-conditioned problems, don't only minimize $\frac{1}{2}\|b - Ax\|_2^2$, but also look for a simple model.

As an example, consider Tikhonov regularization/ ridge regression/ l_2 regularization:

$$\min_x \frac{1}{2}\|b - Ax\|_2^2 + \lambda\|x\|_2^2$$

for some $\lambda > 0$. We can solve this as a bigger least squares problem.

Can change the norm applied to x, for Lasso

$$\min_x \frac{1}{2}\|b - Ax\|_2^2 + \lambda\|x\|_1^2$$

gives sparser solutions as λ increases. However, do not have the tools to solve this without more optimization techniques.

Lecture 12: 3-1-19

Now we are solving $Av = \lambda v$. Given a matrix $A \in \mathbb{R}^{n\times n}$ that is symmetric, $A = A^T$, we want to find real $\lambda_i \in \mathbb{R}$ and associated nonzero vectors $v_i \in \mathbb{R}^n$ such that $Av_i = \lambda_i v_i$.

We assume $A = A^T$ so that the spectral theorem applies. We will often assume some or all of the eigenvalues are distinct. For instance, we may assume $|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$.

To see why we may make that assumption, suppose we have $\lambda_1 = \lambda_2$. Then let A have an eigendecomposition

$$A = \begin{pmatrix} V_1 & V_2 \end{pmatrix} \begin{pmatrix} \lambda_1 I_{2\times 2} & \\ & \Lambda_2 \end{pmatrix} \begin{pmatrix} V_1^T \\ V_1^T \end{pmatrix}$$

where V_1 is $n \times 2$. Then we can replace V_1 with $V_1 Q$ for any orthogonal Q and then we have

$$A = \begin{pmatrix} V_1 Q & V_2 \end{pmatrix} \begin{pmatrix} \lambda_1 I_{2\times 2} & \\ & \Lambda_2 \end{pmatrix} \begin{pmatrix} Q^T V_1^T \\ V_1^T \end{pmatrix}$$

so that there are no unique unit eigenvectors.

We do not solve for eigenvalues by finding roots of the characteristic polynomial $\det(A - xI)$. In fact, we sometimes reverse this problem. Given a monic polynomial $p(x) = x^n + a_{n-1}x^{n-1} + \ldots + a_0$, the roots are the eigenvalues of

$$A = \begin{pmatrix} 0 & 0 & \ldots & 0 & -a_0 \\ 1 & 0 & \ldots & 0 & -a_1 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 & -a_{n-1} \end{pmatrix}$$

we also know that there is no closed-form solution for the roots of a polynomial of degree ≥ 5, so that there is also no closed-form solution for the eigenvalues of an arbitrary matrix of size $n \geq 5$. Hence, we have that *all eigenvalue/vector algorithms must be iterative.* In other words, we have to consider a sequence of iterates $q^{(k)} \in \mathbb{R}^n$ such that

$$q^{(k)} \to v_i \quad \text{as} \quad k \to \infty$$

The **Rayleigh Quotient** for a matrix A is a function $r_A : \mathbb{R}^n \to \mathbb{R}$ given by

$$r_A(x) = \frac{x^T A x}{x^T x}$$

note that if v_i is an eigenvector with eigenvalue λ_i, we have $r_A(v_i) = \lambda_i$. This function has a lot of nice properties.

Consider $A = V \Lambda V^T$. Then we have $A^k = V \Lambda^k V^T = \sum_{i=1}^n \lambda_i^k v_i v_i^T$

From this idea, we have *power iteration.* Given $A = A^T$ and $q^{(0)}$ with unit $2-$norm:

Algorithm 11 Power Iteration

for $k = 1, 2, \ldots$ **do**
  $w = Aq^{(k-1)}$
  $q^{(k)} = \frac{w}{\|w\|_2}$
  $\lambda^{(k)} = q^{(k)T} A q^{(k)}$
  check for convergence
end for

Theorem 3. *If $|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$, and if $v_i^T q^{(0)} \neq 0$, then the iterates $q^{(k)}$ from power iteration satisfy*

$$\left\| q^{(k)} \pm v_1 \right\| = O\Big(\Big| \frac{\lambda_2}{\lambda_1} \Big|^k \Big)$$

$$\left\| \lambda^{(k)} - \lambda_1 \right\| = O\Big(\Big| \frac{\lambda_2}{\lambda_1} \Big|^{2k} \Big)$$

Write $q^{(0)} = \sum_{i=1}^n \alpha_i v_i$, where v_i are the eigenvectors, which is possible since A symmetric implies that there is an eigenbasis. Since $v_1^T q^{(0)} \neq 0$, we have $\alpha_1 \neq 0$. Notice that

$$\begin{aligned} q^{(k)} &= c_k A^k q^{(0)} \\ &= c_k \sum_{i=1}^n \alpha_i A^k v_i \\ &= c_k \sum_{i=1}^n \lambda_i^k \alpha_i v_i \end{aligned}$$

then we can write $q^{(k)} = c_k \lambda_1^k \alpha_1 \Big[v_1 + \sum_{i=2}^k \frac{\alpha_i}{\alpha_1} \Big(\frac{\lambda_i}{\lambda_1} \Big)^k \Big]$. Of the eigenvalue fractions, $\frac{\lambda_2}{\lambda_1}$ converges to zero the quickest. Note that c_k normalizes $A^k q^{(0)}$, so that

$$c_k^{-1} = \sqrt{\sum_{i=1}^n \alpha_i^2 \lambda_i^{2k}} = \alpha_1 \lambda_1^k \sqrt{1 + \sum_{i=2}^n \Big(\frac{\alpha_i}{\alpha_1} \Big)^2 \Big(\frac{\lambda_i}{\lambda_1} \Big)^{2k}}$$

so that

$$q^{(k)} = \Big(v_1 + \sum_{i=2}^{k} \frac{\alpha_i}{\alpha_1}\Big(\frac{\lambda_i}{\lambda_1}\Big)^k\Big) \Big/ \sqrt{1 + \sum_{i=2}^{n} \Big(\frac{\alpha_i}{\alpha_1}\Big)^2 \Big(\frac{\lambda_i}{\lambda_1}\Big)^{2k}}$$

Lecture 13: 3-4-19

Now, want to find other eigenvalues λ_i of A, besides just the one of largest magnitude. Given A, want to find a transform of A, $f(A)$, such that $f(A)$'s largest eigenvalue is related to the eigenvalue of A closest to μ.

Consider

$$A^{-1} = (V\Lambda V^T)^{-1} = V\Lambda^{-1}V^T$$

then, if λ_n is eigenvalue of A of small magnitude, then $\frac{1}{\lambda_n}$ is eigenvalue of A^{-1} of largest magnitude. Moreover, eigenvectors stay the same.

Consider $A - \gamma I$, for $\gamma \in \mathbb{R}$. Then we have

$$A - \gamma I = V\Lambda V^T - \gamma V V^T$$

$$A - \gamma I = V(\Lambda - \gamma I)V^T$$

in particular, each eigenvalue of A is shifted by γ, and the eigenvectors are unchanged.

Now, if we look at $(A - \mu I)^{-1}$, then we have that the eigenvalues are $\frac{1}{\lambda_i - \mu}$

Shifted inverse iteration: Given $A, A = A^T, \mu$

Algorithm 12 Shifted inverse iteration

pick $q^{(0)}$ with $\left\|q^{(0)}\right\| = 1$
for $k = 1, 2, \ldots$ **do**
  solve $(A - \mu I)w = q^{(k-1)}$ for w (equiv to applying inverse to $q^{(k-1)}$)
  $q^{(k)} = \frac{w}{\|w\|}$
  $\lambda^{(k)} = q^{(k)T} A q^{(k)}$
  check for convergence
end for

in exact analogy with the above theorem for the power method,

Theorem 4 (Convergence of shifted inverse iteration)**.** *let λ_J be the eigenvalue of A closest to μ, and assume λ_l is the next closest to μ with $|\mu - \lambda_J| < |\mu - \lambda_l| \leq |\mu - \lambda_i|$ for all $i \neq J$. Then, if $q^{(0)T} v_J \neq 0$, then*

$$\left\|v_J \pm q^{(k)}\right\| = O\left(\left|\frac{\lambda_j - \mu}{\lambda_l - \mu}\right|^k\right)$$

$$\left\|\lambda^{(k)} - \lambda_J\right\| = O\left(\left|\frac{\lambda_j - \mu}{\lambda_l - \mu}\right|^{2k}\right)$$

note that we our $\lambda^{(k)}$ become increasing close to λ_J, so that we can leverage this for faster convergence

Algorithm 13 Rayleigh Iteration

$\lambda^{(0)} = q^{(0)T} A q^{(0)}$
for $k = 1, 2, \ldots$ **do**
  solve $(A - \lambda^{(k-1)} I) w = q^{(k-1)}$ for w (equiv to applying inverse to $q^{(k-1)}$)
  $q^{(k)} = \frac{w}{\|w\|}$
  $\lambda^{(k)} = q^{(k)T} A q^{(k)}$
  check for convergence
end for

Rayleigh iteration: Given $A, A = A^T$ and $q^{(0)}$

note that the system $A - \lambda^{(k-1)} I$ becomes increasingly ill-conditioned, so that the solutions w become worse. However, in practice the magnitude of w may be bad, but the direction is fine, so that the normalized w is close to the true one.

Theorem 5 (Convergence of Rayleigh Iteration)**.** *For symmetric A, Rayleigh iteration converges for a.e. $q^{(0)}$. When it converges, there exists some $J \in \{1, \ldots, n\}$ such that*

$$\left\| q^{(k+1)} \pm v_J \right\| = O\left(\left\| q^{(k)} \pm v_J \right\|_2^3 \right) \quad \text{as } k \to \infty$$

$$\left\| \lambda^{(k+1)} - \lambda_J \right\| = O\left(|\lambda^{(k)} - \lambda_J|^3 \right) \quad \text{as } k \to \infty$$

there is a catch—we do not know which J, that is, which eigenpair, we converge to.

Lecture 14: 3-6-19

Cost of the eigenvalue algorithms. No longer a fixed number of arithmetic operations done.

Cost per iteration of power iteration is $O(n^2)$ due to the matrix-vector product.

For shifted inverse, we are solving a linear system at each iteration. Note that the matrix is the constant, $A - \mu I$ at each iteration, so that we can intially factor $A - \mu I$ with an upfront cost of $O(n^3)$, and then per iteration have a $O(n^2)$ time to solve the linear system with the factorization.

For Rayleigh iteration, the matrix changes at each step, so cannot just factor at start. Thus, it has a per iteration cost of $O(n^3)$.

Now, we consider convergence

$(q^{(k)}, \lambda^{(k)})$ is the eigenvector/value estimate at step k. We know that

$$(q^{(k)}, \lambda^{(k)}) \to (v_J, \lambda_J) \quad \text{as } k \to \infty$$

Given $q^{(k)}, \lambda^{(k)}$, see if it is an approximate eigenvector/value pair by checking

$$\left\| A q^{(k)} - \lambda^{(k)} q^{(k)} \right\|_2 < \epsilon \quad \text{where } \epsilon \text{ a given accuracy}$$

Theorem 6 (Eigenvalue Nearness)**.** *For symmetric A, if $\left\| A q^{(k)} - \lambda^{(k)} q^{(k)} \right\|_2 \leq \epsilon$, then there exists $\lambda_J \in \Lambda(A)$ such that*

$$|\lambda^{(k)} - \lambda_J| \leq \sqrt{2}\epsilon$$

Can also say something about $q^{(k)}$, because $q^{(k)}, \lambda^{(k)}$ are an exact eigenpair of

$$A - eq^{(k)T} \quad \text{where } e = Aq^{(k)} - \lambda^{(k)} q^{(k)}$$

this is because

$$\begin{aligned}(A - eq^{(k)T})q^{(k)} &= e + \lambda^{(k)} q^{(k)} - e \\ &= \lambda^{(k)} q^{(k)}\end{aligned}$$

thus, we have a backwards-stability-resembling result, where $q^{(k)}$ is an eigenvector of a matrix that is close to A. Note that $\left\| e - q^{(k)T} \right\|_2 \leq \epsilon$.

Now, we can trade up front cost for per-iteration cost. We consider the pipeline of putting the matrix A through some fixed work which makes it into a nicer to work with $\hat{A}$. Then we feed $\hat{A}$ into an iterative algorithm, and finally convert the eigenvalues and eigenvectors of $\hat{A}$ into those of A.

Given A symmetric, then for any orthogonal Q, we have an orthogonal similarity tranform, where $Q^T AQ$ has the same eigenvalues of A.

$$Q^T AQ = Q^T V D V^T Q = U D U^T$$

so the eigenvalues of A are the columns of $V = QU$.

Cannot make $\hat{A}$ diagonal, so we pick Q such that $\hat{A}$ is tridiagonal. Say T is an $n \times n$ symmetric tridiagonal matrix, then we can solve $Tx = b$ in $O(n)$ time.

Thus, we can reduce symmetric A to tridiagonal form in $O(n^3)$ time with Householder.

Lecture 15: 3-8-19

We will work to reduce our matrices A to tridiagonal form by orthogonal similarity transforms. Know how to reduce to upper triangular by Householder, but this is too greedy.

$$\begin{bmatrix} \times & \times & \dots & \times \\ \times & \times & \dots & \times \\ \vdots & \ddots & \ddots & \vdots \\ \times & \times & \dots & \times \end{bmatrix} \xrightarrow{Q_1} \begin{bmatrix} \times & \times & \dots & \times \\ 0 & \times & \dots & \times \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \times & \dots & \times \end{bmatrix} \xrightarrow{Q_1^T} \begin{bmatrix} \times & \times & \dots & \times \\ \times & \times & \dots & \times \\ \vdots & \ddots & \ddots & \vdots \\ \times & \times & \dots & \times \end{bmatrix}$$

$$\begin{bmatrix} \alpha & a_1^T \\ a_1 & A_{2,2} \end{bmatrix} \to \begin{bmatrix} 1 & \\ & Q_1 \end{bmatrix} \begin{bmatrix} \alpha & a_1^T \\ \pm \|a_1\| e_1 & Q_1 A_{2,2} \end{bmatrix} \to \begin{bmatrix} 1 & \\ & Q_1^T \end{bmatrix} \begin{bmatrix} \alpha & \pm \|a_1\| e_1^T \\ \pm \|a_1\| e_1 & Q_1 A_{2,2} Q_1^T \end{bmatrix}$$

note that they key is that we do not touch the upper left element, so that we know that the row is zeroed out as well after applying the transpose from the right.

Algorithm 14 Reducing symmetric A to tridiagonal form

for $k = 1, 2, \ldots$ **do**
$x = A[k+1:n, k]$
$v_k = \text{sign}(x_1)\|x\|_2 e_1 + x$
$v_k = \frac{v_k}{\|x_k\|}$
$A[k+1:n, k:n] = A[k+1:n, k:n] - 2v_k(v_k^T A[k+1:n, k:n])$
$A[k:n, k+1:n] = A[k:n, k+1:n] - 2(A[k:n, k+1:n]v_k)v_k^T$
end for

Note householder reflector symmetric. Costs $O(n^3)$ operations. Once we are done, the tridiagonal part of A contains $\hat{A} = QAQ^T$. Given eigenvectors of $\hat{A}$, need to use Q to get eigenvectors of A.

$$\hat{A} = Q_{n-2} \cdots Q_1 A Q_1^T \cdots Q_{n-2}^T$$
$$Q_k = \begin{bmatrix} I_{k \times k} & \\ & I - 2v_k v_k^T \end{bmatrix}$$

We will now make progress towards the QR algorithm to find all eigenvalues/vectors of A.

Consider a "block" power iteration

for $k = 1, 2, \ldots$ **do**
$W = AQ_{k-1}$
normalize columns of W, call that Q_k
end for

note that this does not give us anything more, since the columns of the Q_k will tend towards the dominant eigenvector. However, we can enforce conditions of orthogonality, since we know that the eigenvectors of a symmetric matrix are orthogonal.

Simultaneous iteration, given A symmetric and $Q^{(0)} \in \mathbb{R}^{n \times k}$ orthonormal columns, where $k < n$

for $k = 1, 2, \ldots$ **do**
$W = AQ^{(k-1)}$
$W = Q^{(k)}R$ reduced QR factorization
end for

if $|\lambda_k| > |\lambda_{k+1}|$, then $\text{span}(Q^{(k)}) \to \text{span}(v_1, \ldots, v_k)$ as $O\left(\left|\frac{\lambda_{k+1}}{\lambda_k}\right|^k\right)$

Lecture 16: 3-11-19

Assume $|\lambda_1| > |\lambda_2| > \ldots > |\lambda_k| > |\lambda_{k+1}| \geq \ldots \geq 0$. And a mild condition on $V_1^T Q^(0)$, where $V_1 = \begin{bmatrix} v_1 & \ldots & v_k \end{bmatrix}$ [non-singular leading minors] (analogous to initial conditions on power iteration). Then

$$\text{span}(Q^{(l)}) \to \text{span}(V_1) \quad \text{at rate } O\left(\left|\frac{\lambda_{k+1}}{\lambda_k}\right|^l\right)$$

$$\implies \left\| Q^{(l)} Q^{(l)T} - V_1 V_1^T \right\|_2 \to 0$$

if the norm here is 0, then $Q^{(l)} = V_1 U$ for some orthogonal U (because orthogonal projectors are unique ...) Then $Q^{(l)T} A Q^{(l)} = U^T \Lambda_1 U$, where $A = \begin{bmatrix} V_1 & V_2 \end{bmatrix} \begin{bmatrix} \Lambda_1 & \\ & \Lambda_2 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix}$. Thus, we can compute the eigenvectors/ values of $\hat{A} = Q^{(l)T} A Q^{(l)}$, so that we have eigenvalues of A and a U such that $Q^{(l)}U$ are eigenvectors. Important, $\hat{A}$ is $k \times k$, so it is quicker to compute this stuff.

Now, for convergence, consider $A^l Q^{(0)}$. Then we know $A^l Q^{(0)} e_1 \approx c_l v_1$, where $c_l \in \mathbb{R}$. This is a fact from the convergence of power iteration. Now, consider $(I - v_1 v_1^T) A^l Q^{(0)} e_2$, which orthogonalizes against v_1. We have $A^l Q^{(0)} e_2 = \hat{c}_l \sum_{i=1}^n \alpha_i \lambda_i^l v_i$. Then we get that the product is equal to

$$\hat{c}_l \sum_{i=2}^n \alpha_i \lambda_i^l v_i = \hat{c}_l \alpha_2 \lambda_2^l \Big(v_2 + \sum_{i=3}^n \frac{\alpha_i}{\alpha_2} \Big(\frac{\lambda_i}{\lambda_2} \Big)^l v_i \Big)$$

Thus, taking QR of $A^l Q^{(0)}$ gives information about each eigenvector and eigenvalue.

This implies that a QR factorization of A^l is a reasonable way to get eigenvectors as $l \to \infty$ if $|\lambda_1| > |\lambda_2| > \ldots > |\lambda_n| \geq 0$. (like picking $Q^{(0)}$ to be identity). If A is symmetric positive definite, then its eigendecomposition is its SVD, so that $A^l = V \Sigma^l V^T$ implies $\kappa(A^l) = \kappa(A)^l$. This is bad. Recall for instance that the forward error depends on the condition number.

Algorithm 15 QR Algorithm

given $A = A^T$	
$A^{(0)} = Q^{(0)T} A Q^{(0)}$ reduce to tridiagonal	$O(n^3)$
for $l = 1, 2, \ldots$ **do**	
$Q^{(l)} R^{(l)} = A^{(l-1)}$ QR factorization	$O(n)$
$A^{(l)} = R^{(l)} Q^{(l)}$	$O(n)$
end for	

Claim: with mild conditions on V (same as before, non-singular $V[1:k, 1:k]$), and $|\lambda_1| > |\lambda_2| > \ldots > |\lambda_n| \geq 0$. Then

$$A^{(l)} \to \Lambda \quad \text{as } l \to \infty$$
$$\bar{Q} = Q^{(0)} Q^{(1)} \cdots Q^{(l)} \to V$$

note that $R^{(l)} = Q^{(l)T} A^{(l-1)}$ so that $A^{(l)} = Q^{(l)^T} A^{(l-1)} Q^{(l)}$ so that by induction $A^{(l)}$ has the same eigenvalues of A. Thus, if the iterates $A^{(l)}$ converge to a diagonal matrix, the diagonal elements must be the eigenvalues of A.

Indeed, $A^{(l)} = Q^{(l)T} \cdots Q^{(0)T} A Q^{(0)} \cdots Q^{(l)}$. Call $\bar{Q} = Q^{(1)} \cdots Q^{(l)}$

Claim: $\bar{Q}$ is the Q factor in a QR factorization of $(A^{(0)})^l = \bar{Q} \bar{R}$, which is what we justified was a good idea before.

To see this, we have

$$\begin{aligned}(A^{(0)})^l &= A^{(0)}\cdots A^{(0)}\\ &= Q^{(1)}R^{(1)}\cdots Q^{(1)}R^{(1)}\\ &= Q^{(1)}A^{(1)}\cdots A^{(1)}R^{(1)} \quad l-1 \text{ copies of } A^{(1)}\\ &= Q^{(1)}Q^{(2)}R^{(2)}\cdots Q^{(2)}R^{(2)}R^{(1)}\\ &= Q^{(1)}Q^{(2)}A^{(2)}\cdots A^{(2)}R^{(2)}R^{(1)} \quad l-2 \text{ copies of } A^{(2)}\\ &\vdots\\ &= Q^{(1)}\cdots Q^{(l)}R^{(l)}\cdots R^{(1)}\end{aligned}$$

so that we are done, $\bar{Q} = Q^{(1)}\cdots Q^{(l)}$, and $\bar{R} = R^{(l)}\cdots R^{(1)}$

Lecture 17: 3-13-19

Above, in QR iteration, both convergences are at rate

$$O\Big(\max_{1\le j\le n-1}\left|\frac{\lambda_{j+1}}{\lambda_j}\right|^k\Big)$$

this is the rate for which the slowest columns/ eigenvalues to converge do converge. Different columns/ eigenvalues converge at different rates.

Practical QR Algorithm, given A symmetric

Algorithm 16 Practical QR Algorithm

given A symmetric
$A^{(0)} = Q^{(0)T}AQ^{(0)}$ tridiagonalize A
for $k = 1, 2, \ldots$ **do**
  Pick a shift $\mu^{(k)}$
  $A^{(k-1)} - \mu^{(k)}I = Q^{(k)}R^{(k)}$
  $A^{(k)} = R^{(k)}Q^{(k)} + \mu^{(k)}I$
  if any off diagonal element of $A^{(k)}$ is small enough, e.g. $|A^{(k)}_{j,j+1}| \le \epsilon(|A^{(k)}_{j,j} + A^{(k)}_{j+1,j+1}|)$, **then**
    set $A^{(k)}_{j+1,j} = A^{(k)}_{j,j+1} = 0$
    $\implies \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} = A^{(k)}$
    call the QR algorithm on A_1 and A_2
    (Base case 1×1 or 2×2)
  end if
end for

a common choice of $\mu^{(k)}$ is $(A^{(k-1)})_{n,n}$. Can check that the transform is still a similarity transform. Say we have a symmetric matrix $A = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}$ then we can get the eigenvalues and eigenvectors of A by computing those of A_1 and A_2.

Note it is easy to get the eigenvalues from the recursion (just union of returned eigenvalues). However, it is somewhat more convoluted to get eigenvectors of A back by undoing all of the similarity transforms.

Using the shift $\mu^{(k)} = (A^{(k-1)})_{n,n}$ lets us relate a column of the matrix $\bar{Q}$ to running Rayleigh iteration with it. Converges really fricken quickly, looks like constant iterations (like direct methods), say C iterations to converge per eigenvalue. Then like Cn iterations overall, so overall cost $O(Cn^2)$ to run the iterations, so the most expensive part is usually the $O(n^3)$ tridiagonalization. Note that the time complexity of QR for tridiagonal matrices if $O(n)$ using Givens rotations.

Stability/ accuracy of standard QR iteration Let $A \in \mathbb{R}^{n\times n}$ symmetric, and diagonalized by the pure QR algorithm with out floating point axioms. Let $\tilde{\Lambda}$ be the computed eigenvalues as a diagonal matrix. Let $\tilde{Q}$ be the *mathematically exact* product of the *numerically computed* householder reflectors in each QR factorization. Note that mathematical orthogonality is delicate, as numerically computed matrices are essentially never actually orthogonal. Then

$$\tilde{Q}\tilde{A}\tilde{Q}^T = A + \delta A \quad \text{with } \frac{\|\delta A\|}{\|A\|} = O(\mu_{\text{machine}})$$

$$\implies \frac{|\tilde{\lambda}_j - \lambda_j|}{\|A\|} = O(\mu)$$

note that this means that for λ_j close in magnitude to $\|A\|$, then the statement is approximately a relative stability statement, so such λ_j are computed accurately.

Lecture 18: 3-15-19

Our problem is now to find x such that $f(x) = 0$. We want methods that only require mild assumptions on f, and only require evaluation of f, and maybe its derivatives f', f''. We will build sequences of iterates $x^{(k)} \to x^*$, where $f(x^*) = 0$.

For the moment, assume $x \in \mathbb{R}$, $f : \mathbb{R} \to \mathbb{R}$.

First, we consider different termination criteria:

1. stop when $|f(x^{(k)})| < f_{tol} \in \mathbb{R}^+$

   can fail if the function just gets really close to 0 without achieving it

   can also fail if f goes to zero very slowly (flat graph), so that the algorithm terminates when $x^{(k)}$ is still very far from x^*

2. stop when $|x^{(k+1)} - x^{(k)}| < a_{tol}$ (absolute tolerance)

3. stop when $|x^{(k+1)} - x^{(k)}| < r_{tol}|x_k|$ (relative tolerance)

4. stop when $|x^{(k+1)} - x^{(k)}| < \epsilon|1 + x_k|$ (combines above two)

1. kinda sucks, needs to be used in conjunction with other stuff, such as the other ones listed or by taking into account derivative information when we have that.

Bisection

Given $f : [a,b] \subseteq \mathbb{R} \to \mathbb{R}$ find a root in $[a,b]$. Assume $f \in C[a,b]$.

First, check if $f(a)f(b) \leq 0$. If equals 0 then done. If is negative, then f changes sign in the interval, so by the intermediate value theorem, there is a zero of f in this interval. Note that we do not know anything if $f(a)f(b) > 0$.

Let $p = \frac{a+b}{2}$, the midpoint. Evaluate $f(p)$. Then either $f(a)f(p) \leq 0$, or $f(b)f(p) \leq 0$, so we can continue recursively on the halved interval.

given a, b, f, tol,

Algorithm 17 Bisection Method

for $k = 1, 2, \ldots, \left\lceil \log_2 \left(\frac{b-a}{2\text{tol}} \right) \right\rceil$ **do**
  $p = \frac{a+b}{2}$
  if $f(p)f(a) < 0$ **then**
    $b = p$
  else if $f(p)f(a) > 0$ **then**
    $a = p$
  else
    return p
  end if
end for
$p = \frac{a+b}{2}$
return p

Convergence of Bisection

If we want $|x^* - p| < \text{tol}$, then we need $\frac{b-a}{2} 2^{-k} < \text{tol}$. Then (taking logs etc), $k = \left\lceil \log_2 \left(\frac{b-a}{2\text{tol}} \right) \right\rceil$

Fixed points

$f(x) = 0$ can be solved by finding x such that $g(x) = x$ for specificallly constructed g. For instance, a fixed point of $g(x) = f(x) + x$ is also a root of f.

Algorithm to solve $g(x) = x$ (fixed point iteration)

Algorithm 18 Fixed point iteration

pick $x^{(0)}$
for $k = 0, 1, \ldots$ **do**
  $x^{(k+1)} = g(x^{(k)})$
  check convergence
end for

Theorem 7 (Fixed point)**.** *Let $g \in C[a,b]$ with $a \leq g(x) \leq b$ for all $x \in [a,b]$ (i.e. g maps interval back into itself). Then there exists a fixed point $x^* \in [a,b]$ such that $g(x^*) = x^*$. Furthemore, the*

fixed point is unique if g is differentiable and there exists some $\rho < 1$ such that $|g'(x)| < \rho$ for all $x \in [a,b]$.

Proof of existence: let $\phi(x) = g(x) - x$, and prove there is a root of $\phi(x) \in [a,b]$.

Convergence of fixed point iteration if x^* is unique

$$\begin{aligned} |x^{(k+1)} - x^*| &= |g(x^{(k)}) - g(x^*)| \\ &\leq \rho |x^{(k)} - x^*| \\ &\vdots \\ &\leq \rho^{k+1} |x^{(0)} - x^*| \end{aligned}$$

Lecture 19: 3-18-19

We go from

$$\text{fixed point iteration} \to \text{standard iterations for } Ax = b$$

Want to solve $Ax = b$, let's assume A is $n \times n$ non-singular. Let $A = M - N$ with M non-singular. Then we have

$$\begin{aligned} Ax = b &\implies Mx - Nx = b \\ Mx &= Nx + b \\ x &= M^{-1}(Nx + b) =: g(x) \end{aligned}$$

We wish to find an x that is a fixed point of the g defined here. This gives us our first iterative method for solving linear systems

Given $A, M, N, x^{(0)}, b$,

Algorithm 19 Classical iterations/ Stationary iterative methods

for $k = 1, 2, \ldots$ **do**
  Solve $Mx^{(k)} = Nx^{(k-1)} + b$
  Check convergence
end for

We consider convergence criteria. We have $Mx^{(k)} = Nx^{(k-1)} + b$. Moreover, we know that $Mx^* = Nx^* + b$. Subtracting these, we get $M(x^{(k)} - x^*) = N(x^{(k-1)} - x^*)$ so that

$$\begin{aligned} x^{(k)} - x^* &= M^{-1}N(x^{(k-1)} - x^*) \\ x^{(k)} - x^* &= (M^{-1}N)^k (x^{(0)} - x^*) \\ \left\| x^{(k)} - x^* \right\| &= \left\| (M^{-1}N)^k (x^{(0)} - x^*) \right\| \\ &\leq \left\| M^{-1}N \right\|^k \left\| x^{(0)} - x^* \right\| \end{aligned}$$

Theorem 8 (Convergence of Basic Iterative Method)**.** *Given b, $A = M - N$ with M, A nonsingular, if $\rho(M^{-1}N) = \max_i |\lambda_i| < 1$, i.e. the spectral radius of $M^{-1}N$ is less than 1, then the sequence $(x^{(k)})_k$ given by $Mx^{(k)} = Nx^{(k-1)} + b$ converges to $x^* = A^{-1}b$ for any initial $x^{(0)}$.*

$$x^{(k)} \to x^* = A^{-1}b$$

Let $A = L + D + U$, its strict lower triangle, diagonal, and strict upper triangle. The upcoming methods will pick different parts of these to be M.

Lecture 20: 3-20-19

To see importance of spectral radius $\rho(M^{-1}N)$, note that there exists a submultiplicative matrix norm $\|\cdot\|_*$ such that $\left\|M^{-1}N\right\|_* < 1$

As above, let $A = L + D + U$.

Jacobi picks $M = D$, so that $N = -L - U$. We need D to have nonzeros on the diagonal. This iteration converges for all matrices that are *strictly diagonal dominant*—those that have $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$ for $i = 1, \ldots, n$.

Proof. We use a special case of Gershgorin's Disk theorem: given $A \in \mathbb{R}^{n \times n}$, let $r_i = \sum_{j \neq i} |a_{ij}|$

Let $D_i = \{z \in \mathbb{C} : |z - a_{ii}| < r_i\}$. Then all of the eigenvalues of A lie in $\cup_i D_i$.

Now, consider the matrix $M^{-1}N = D^{-1}(-L - U)$, and note that its diagonal is 0. Note that

$$\begin{aligned} r_i &= \sum_{j \neq i} \left| \frac{1}{a_{ii}} a_{ij} \right| \\ &= \frac{1}{|a_{ii}|} \sum_{j \neq i} |a_{ij}| \\ &< 1 \end{aligned}$$

Thus, the eigenvalues of $M^{-1}N$ are all of magnitude less than 1, so that the spectral radius $\rho(M^{-1}N) < 1$ and thus Jacobi converges. □

Gauss-Seidel picks $M = D + L$ so that $N = -U$. If A is symmetric positive definite, then Gauss-Seidel converges for any initial $x^{(0)}$.

New topic: say we have sequences ot iterates

$$\tilde{x}^{(0)}, \tilde{x}^{(1)}, \tilde{x}^{(2)}, \ldots, \tilde{x}^{(k)} \quad \text{want } \tilde{x}^{(k)} \to x^*$$

given $\tilde{x}^{(0)}, \ldots, \tilde{x}^{(k)}$, consider

$$x^{(k)} = \sum_{i=0}^{k} \alpha_i \tilde{x}^{(i)}$$

we use the past information to better our next iterate.

More generally, to solve $Ax = b$, we could consider a sequence of subspaces

$$V_1, V_2, \ldots, V_k \qquad \dim(V_i) = i$$

then define $x^{(k)}$ as the "best" vector in V_k.

We will pick V_i as *Krylov subspaces*. The kth Krylov subspace associated with a matrix A and vector b is $\mathcal{K}_k(A,b) = \text{span}\{b, Ab, \ldots, A^{k-1}b\}$

Lecture 21: 3-22-19

Construct an iterative method for solving $Ax = b$ by picking $x^{(k)} \in \mathcal{K}_k(A,b)$ to be the best choice in some sense. Under mild assumptions, $\mathcal{K}_n(A,b) = \mathbb{R}^n$.

Algorithm 20 General Krylov method form

for $k = 1, 2, \ldots$ **do**
  $x^{(k)} = \text{argmin}_{x \in \mathcal{K}_k(A,b)} f(x, A, b)$
  Check convergence
end for

We can choose

$$f(x,A,b) = \frac{1}{2}\|b - Ax\|_2^2$$

$$f(x,A,b) = \|x - x^*\|_2^2$$

this function actually turns out to be impossible to minimize. However, this works:

$$f(x,A,b) = \|x - x^*\|_A^2$$

For symmetric positive definite A, we define $\|x\|_A = \sqrt{x^T A x}$.

We can actually minimize the first and third functions.

Choosing (for symmetric positive definite A) $f(x,A,b) = \|x - x^*\|_A^2$ yields the conjugate gradient method.

Algorithm 21 CG (conceptually)

for $k = 1, 2, \ldots$ **do**
  $x^{(k)} = \text{argmin}_{x \in \mathcal{K}_k(A,b)} \frac{1}{2}\|x - x^*\|_A^2$
  Check convergence
end for

Need to solve

$$\min_{x \in \mathcal{K}_k(A,b)} \frac{1}{2}\|x - x^*\|_A^2$$

say we are given a matrix V_k that is an orthonormal basis for $\mathcal{K}_k(A,b)$. Then

$$\min_{x \in \mathcal{K}_k(A,b)} \frac{1}{2}\|x - x^*\|_A^2 \iff \min_{y \in \mathbb{R}^k} \frac{1}{2}\|V_k y - x^*\|_A^2$$

in general, to minimize over vectors in a subspace, can take a basis of the subspace, and characterize the vectors in the subspace as the image of the matrix of the basis. Then we only minimize over a smaller dimensional vector space.

Observe that

$$\begin{bmatrix} b & Ab & A^2b & \dots & A^{k-1}b \end{bmatrix}$$

has columns that converge to the dominant eigenvector. Thus, it is ill-conditioned. Instead, we use other methods.

Lanczos. $\beta_0 = 0, \alpha_0 = 0, v_1 = \frac{b}{\|b\|}$.

Algorithm 22 Lanczos

for $k = 1, 2, \dots$ **do**
  $q = Av_k$
  $\alpha_k = v_k^T q$
  $q = q - \beta_{k-1} v_{k-1} - \alpha_k v_k$
  $\beta_k = \|q\|_2$
  $v_{k+1} = \frac{q}{\beta_k}$
end for

note the similarities to Gram-Schmidt. We have that

$$V_k = \begin{bmatrix} v_1 & \dots & v_k \end{bmatrix}$$

is an orthonormal basis for $\mathcal{K}_k(A, b)$. Furthemore, the matrices V_k and V_{k+1} satisfy an interesting recurrence relation:

$$AV_k = V_{k+1}\tilde{T}_k \qquad \tilde{T}_k = \begin{bmatrix} T_k \\ \beta_k e_k^T \end{bmatrix} \in \mathbb{R}^{k+1,k}, \quad T_k = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{k-1} \\ & & \beta_{k-1} & \alpha_k \end{bmatrix}$$

given this specific V_k, want to solve the above minimization problem. Recall we can solve

$$\min_{y \in \mathbb{R}^k} \frac{1}{2} \|V_k y - x^*\|_A^2$$

, and then set $x^{(k)} = V_k y^{(k)}$ where $y^{(k)}$ is the above solution. This is equivalent to minimizing

$$\begin{aligned}
\frac{1}{2}(V_k y - A^{-1}b)^T A (V_k y - A^{-1}b) &= \frac{1}{2}\left[y^T V_k^T A V_k y - y^T V_k^T b - b^T V_k y + b^T A^{-1} b \right] \\
&\iff \frac{1}{2}\left[y^T V_k^T A V_k y - y^T V_k^T b - b^T V_k y \right] \\
&\iff \frac{1}{2}\left[y^T T_k y - 2 b^T V_k y \right] \quad \text{using below} \\
&\iff \frac{1}{2}\left[y^T T_k y - 2\|b\| e_1^T y \right] \quad \text{non-first cols of } V_k \text{ orthog to } v_1 = b \\
&\implies T_k y - \|b\| e_1 = 0 \quad \text{taking derivative wrt } y \\
&\iff T_k y = \|b\| e_1
\end{aligned}$$

note that we eliminate $b^T A^{-1} b$, which we cannot compute (since $x^* = A^{-1}b$). We use above that

$$V_k^T A V_k = V_k^T \begin{bmatrix} V_k & v_{k+1} \end{bmatrix} \begin{bmatrix} T_k \\ \beta_k e_k^T \end{bmatrix}$$

So our minimization step is to minimize $T_k y = \|b\| e_1$, and then set $x^{(k)} = V_k y^{(k)}$. The T_k is tridiagonal so the solve is fast.

Lecture 22: 3-25-19: Krylov Subspaces and Conjugate Gradients

Algorithm 23 CG given spd A, $x^{(0)} = 0, r^{(0)} = b, p^{(0)} = r^{(0)}$

for $k = 1, 2, \ldots$ **do**
$\gamma^{(k)} = \frac{r^{(k-1)T} r^{(k-1)}}{p^{(k-1)T} A p^{(k-1)}}$
$x^{(k)} = x^{(k-1)} + \gamma^{(k)} p^{(k-1)}$
$r^{(k)} = r^{(k-1)} - \gamma^{(k)} A p^{(k-1)}$
$\mu^{(k)} = \frac{r^{(k)T} r^{(k)}}{r^{(k-1)T} r^{(k-1)}}$
$p^{(k)} = r^{(k)} + \mu^{(k)} p^{(k-1)}$
Check convergence.
end for

The iterates $x^{(k)}$ are in fact equivalent to the above minimizations over the Krylov subspaces.

walking through, first we have a step size of the squared norm of the residual divided by the inner product inducing the A norm. Then we take a step in the direction of the previous residual of step size $\gamma^{(k)}$.

Convergence of CG

Since $r^{(k)} = b - Ax^{(k)}$, we can test whether

1. Check if $\left\|r^{(k)}\right\|_2 < \epsilon$
2. Check if $\left\|r^{(k)}\right\|_2 < \epsilon\left(\left\|x^{(k)}\right\|_2\right)$. This is like a relative error, except since we do not have the true solution we check against the current iterate.
3. Check if $\left\|r^{(k)}\right\|_2 < \epsilon\left(\left\|x^{(k)}\right\|_2 + \|b\|_2\right)$

Cost of CG

We only do one thing with A, namely, a multiplication $Ap^{(k-1)}$. Thus, our cost per iteration is

$$T_{mult}(A) + O(n)$$

everything besides the multiplication by A are inner products and basic arithmetic. Notably, there is no dependence on the iteration number. Another key point is that we only need to know how to apply A to a vector, without forming A.

Convergence theory

Say that $A^l b \in \mathcal{K}_l(A,b)$, and l is the first such time that this happens. Then

$$\begin{aligned} A^l b &= c_1 b + c_2 Ab + \ldots + c_l A^{l-1} b \\ &\text{Can prove } c_1 \neq 0 \\ A^{l-1} b &= A^{-1} c_1 b + c_2 b + \ldots + A^{l-2} b \\ A^{-1} b &= \frac{1}{c_1}\Big(A^{l-1} b - c_2 b - \ldots - c_l A^{l-2} b\Big) \end{aligned}$$

so that the solution x is in $\mathcal{K}_l(A,b)$. Thus, the process only stops when the solution is found.

This happens if A has only l distinct eigenvalues.

Krylov subspaces have a close relationship with polynomials. Any $y \in \mathcal{K}_k(A,b)$ can be written

$$\begin{aligned} y &= c_1 b + c_2 Ab + \ldots + c_k A^{k-1} b \\ &= (c_1 I + c_2 A + \ldots + c_k A^{k-1}) b \end{aligned}$$

let $e^{(k)} = (x^{(k)} - x^*)$, where $x^{(k)}$ are the iterates from CG.

Theorem 9.

$$\frac{\left\|e^{(k)}\right\|_A}{\left\|e^{(0)}\right\|_A} \leq \inf_{p \in P_k^*} \max_{\lambda \in \Lambda(A)} |p(\lambda)|$$

where P_k^* is the set of polynomials of degree k with $p(0) = 1$.

this proves the above results on the l distinct eigenvalues. Moreover, we have that if eigenvalues are close together, then this converges well because polynomials can be fit at 0 near that group.

General theory says convergence takes roughly $\sqrt{\kappa_2(A)}$ iterations. (This can be misleading).

Note that this explains why we do not need the entries of A. We need only the multiplications of A and convergence depends only on the spectrum of A.

In practice, rather than solving $Ax = b$, pick $M \approx A$ is some way and solve $M^{-1}Ax = M^{-1}b$. This is a huge field known as preconditioning.

Lecture 23: 3-27-19

Note that for our iterative methods to solve $Ax = b$, some assumed $x^{(0)} = 0$. To deal with this, we can use *iterative refinement*. We wish to solve $Ax = b$ with guess $\overline{x}$ for x^*.

1. set $r = b - A\overline{x}$
2. solve $Ad = r$ for d with initial guess zero
3. set $\hat{x} = \overline{x} + d$

this makes sense since if we solve for d exactly, we have $A(\overline{x}+d)=b$.

For an iterative method, define rates and order of convergence. Assume we have a sequence $x_k \to x^*$ as $k \to \infty$.

Order of convergence:

1. *linear*: there exists $\rho < 1$ such that for k large enough,
$$|x_{k+1} - x^*| \le \rho |x_k - x^*|$$
2. *superlinear*: there exists a sequence $\rho_k \to 0$ such that for large enough k,
$$|x_{k+1} - x^*| \le \rho_k |x_k - x^*|$$
3. *quadratic*: there exists an $M > 0$ such that for large enough k,
$$|x_{k+1} - x^*| \le M |x_k - x^*|^2$$
4. *cubic*: there exists an $M > 0$ such that for large enough k,
$$|x_{k+1} - x^*| \le M |x_k - x^*|^3$$

$$\text{cubic} \implies \text{quadratic} \implies \text{superlinear} \implies \text{linear}$$

for linear convergence, we define the *rate of convergence* as $-\log_{10}\rho$. We justify this: note that

$$|x_k - x^*| = \rho^k |x_0 - x^*|$$

if we want to reduce the initial factor by $\epsilon > 0$, so we want $|x_k - x^*| \le \epsilon |x_0 - x^*|$, or equivalently that

$$\begin{aligned} \rho^k &\le \epsilon \\ k \log_{10}\rho &\le \log_{10}\epsilon \\ -k\log_{10}\rho &\ge \log_{10}\frac{1}{\epsilon} \\ k &\ge \frac{1}{-\log_{10}\rho}\log_{10}\frac{1}{\epsilon} \end{aligned}$$

so the number of iterations needed to converge, is inversely proportional to the rate of convergence.

Now, we are back to finding root-finding, want $f(x^*)=0$. Note that bisection converges linearly with $\rho = \frac{1}{2}$. The only assumption needed was that f was continuous.

Say $f \in C^2[a,b]$, and say we can evaluate $f(x), f'(x)$. Given a point $x_0 \in [a,b]$, then

$$f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{1}{2}f''(\xi(x))(x-x_0)^2 \qquad \xi(x) \text{ between } x \text{ and } x_0$$

Say that x^*, a root exists. Then plugging in,

$$0 = f(x_0) + f'(x_0)(x^*-x_0) + \frac{1}{2}f''(\xi)(x^*-x_0)^2$$

Algorithm 24 Newton's Method for root finding

for $k = 0, 1, 2, \ldots$ **do**
  $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$
  check convergence
end for

if f were linear, so that $f'' = 0$. Moreover, if we were just close to a root, then $(x^* - x_0)^2$ is very small, and thus possibly negligible compared to $(x^* - x_0)$.

If f were linear, $x^* = x_0 - \frac{f(x_0)}{f'(x_0)}$. Using this idea, we have Newton's method for root finding. Given $f \in C^2[a,b]$ and x_0,

Theorem 10 (Convergence of Newton's Method)**.** *if $f \in C^2[a,b]$, and there is a root $x^* \in [a,b]$ such that $f(x^*) = 0$, and $f'(x^*) \neq 0$. Then there exists $\delta > 0$ such that if $|x_0 - x^*| < \delta$, then Newton's Method converges quadratically.*

If $f'(x^*) = 0$, then Newton's method may still converge, but only linearly if it does.

Lecture 24: 3-29-19

Say we do not have f'. We want to approximate $f'(x_k)$ is some way if we want to use a method similar to Newton's method. We use the first-order finite difference approximation

$$f(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

Algorithm 25 Secant method for root finding (Given $f \in C^2[a,b], x_0, x_1$)

for $k = 1, 2, \ldots$ **do**
  $x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}$
  check convergence
end for

Theorem 11 (Convergence of secant method)**.** *If $f \in C^2[a,b]$, and there exists a root $x^* \in [a,b]$, with $f'(x^*) \neq 0$, then there exists some $\delta > 0$ such that for all $x_0, x_1 \in (x^* - \delta, x^* + \delta)$, then secant method converges superlinearly to x^*.*

$f(x) = x^2$ provides an example where secant method need not converge when $f'(x^*) = 0$, since with $x^* = 0$, we see that for any $\delta > 0$, we can choose two points x_0, x_1 with the same y values on either side of the root, so that the approximated derivative is zero and thus secand method does not move.

Now, we consider the problem $\min_x \phi(x)$. Assume $\phi \in C^2$. At any x_0, we can write

$$\phi(x) = \phi(x_0) + \phi'(x_0)(x - x_0) + \frac{1}{2}\phi''(\xi)(x - x_0)^2$$

any point x^* with $\phi'(x^*) = 0$ is called a **critical point**.

If $\phi''(x^*) > 0$, we call this point a local minimizer of ϕ. This is because there exists $\delta > 0$ such that $|x^* - x| < \delta \implies \phi(x^*) < \phi(x)$.

Is a local max if $\phi''(x^*) < 0$.

Need more information if $\phi''(x^*) = 0$.

We will be satisfied if we can find a local minimum.

Since we know that $\phi'(x^*) = 0$ is necessary for local min/max, we try Newton applied to $\phi'(x)$.

---

for $x = 1, 2, \ldots$ **do**
  $x_{k+1} = x_k - \frac{\phi'(x_k)}{\phi''(x_k)}$
  check convergence
end for

In this algorithm, at each k, we pick x_{k+1} to minimize or maximize the quadratic

$$g(x) = \phi(x_k) + \phi'(x_k)(x - x_k) + \frac{1}{2}\phi''(x_k)(x - x_k)^2$$

note that a quadratic is either convex/ concave, so it has a single global minimum or maximum.

To prevent possible convergence to a maximum, we could enforce $\phi(x_{k+1}) < \phi(x_k)$. To enforce something like this, we have to go back and change the algorithm.

Lecture 25: 4-8-19

We are considering the minimization problem

$$\min_{x \in \mathbb{R}^n} \phi(x) \quad \text{where } \phi : \mathbb{R}^n \to \mathbb{R}$$

first consider how to solve a set of non-linear equations.

$$\begin{aligned} f_1(x) &= 0 \\ f_2(x) &= 0 \\ &\vdots \quad \vdots \\ f_n(x) &= 0 \end{aligned}$$

where $f_i : \mathbb{R}^n \to \mathbb{R}$.

For example, say $f_1(x) = x_1^2 - 2x_1 - x_2 + 1, \quad f_2(x) = x_1^2 + x_2^2 - 1$. The first equation $f_1(x) = 0$ defines a parabola ($x_2 = x_1^2 - 2x_1 + 1$), and $f_2(x) = 0$ defines a circle of radius 1. Solutions x give points of intersection.

Since direct methods will clearly not work, we wish to construct iterative methods with convergence $x_k \to x^*$ where $f(x^*) = 0$ if x_0 is close enough to x^*. Our approach is Newton's method. To do this, we need the Taylor expansion: given $p \in \mathbb{R}^n$,

$$f(x + p) = f(x) + df(x)p + O(\|p\|^2)$$

is true if $f \in C^2$ and has bounded second derivatives (so we can write the error term as such).

Say x is near x^* (where $f(x^*) = 0$)). Then define p^* by $x^* = x + p^*$. Then we have

$$f(x + p^*) = 0 = f(x) + J(x)p^* + O(\|p^*\|^2)$$

Algorithm 26 Newton's Method for nonlinear equations

for $k = 1, 2, \ldots$ **do**
  Solve $J(x_{k-1})p_k = -f(x_{k-1})$
  Set $x_k = x_{k-1} + p_k$
end for

thus, we have a nice estimate for p^* by $-J(x)^{-1}f(x)$.

If there exists a neighborhood of x^* where $J(x)$ is non-singular, has continuous derivatives, and has a bounded inverse, then there exists a neighborhood of x^* such that any initial iterate x_0 in the neighborhood gives Newton's method converging quadratically to x^*.

Now, we go back to our minimization problem. A point x^* is a *global minimizer* of $\phi(x)$ if $\phi(x^*) \leq \phi(x)$ for all x. We also say a point x^* is a *strict global minimizer* of $\phi(x)$ if $\phi(x^*) < \phi(x)$ for all $x \neq x^*$.

We often seek local minimizers instead of global minimizers. x^* is a *local minimzer* of $\phi(x)$ if there exists a neighborhood N of x^* such that $\phi(x^*) \leq \phi(x)$ for all $x \in N$. A *strict local minimizer* is defined analogously.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be C^2. Then for $p \in \mathbb{R}^n$,

$$f(x+p) = f(x) + \nabla f(x+tp)^T p \quad \text{for some } t \in (0,1)$$

$$f(x+p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x+tp) p \quad \text{for some } t \in (0,1)$$

If we assume further that third order derivatives of f are bounded, then we can write

$$f(x+p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x) p + O(\|p\|^3)$$

Lecture 26: (4/10)

Given $x^* \in \mathbb{R}^n$, we will discuss necessary and sufficient conditions for x^* to be a (strict) local minimizer of $f : \mathbb{R}^n \to \mathbb{R}$.

First order necessary condition: If x^* is a local minimizer of f and f is C^1 in some neighborhood of x^*, then $\nabla f(x^*) = 0$.

Proof. Suppose $\bar{x}$ has $\nabla(\bar{x}) \neq 0$. Let $p = -\nabla f(\bar{x})$. Then $p^t \nabla f(\bar{x}) = -\|\nabla f(\bar{x})\|_2^2$. Since ∇f is continuous, there exists ρ such that $p^T \nabla f(\bar{x} + tp) < 0$ for all $t \in [0, \rho)$.

$$f(\bar{x} + \delta p) = f(\bar{x}) + p^T \nabla f(x + \bar{t}\delta p)$$

for $0 < \delta < \rho$, and $\bar{t} \in (0,1)$. Thus, $f(\bar{x} + \delta p) < f(\bar{x})$, and holds for arbitrarily small δ. □

Second order necessary condition: If x^* a local minimizer of f, and f is C^2 in some neighborhood of x^*, then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive semi-definite.

Suppose we have $\bar{x}$ with $\nabla f(\bar{x}) = 0$, so that

$$f(\bar{x} + p) = f(\bar{x}) + \frac{1}{2} p^T \nabla^2 f(\bar{x} + tp) p \quad t \in (0,1)$$

then if $p^T \nabla^2 f(\bar{x}) p$ were negative, we could use continuity etc and see that $f(\bar{x}+p) < f(\bar{x})$.

Second order sufficient condition: Suppose $\nabla^2 f$ is continuously differentiable in a neighborhood of a point x^*. If $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite, then x^* is a strict local minimizer of f.

Our first attempt at optimization: Find points where $\nabla f(x) = 0$.

Algorithm 27 Newton's method for unconstrained minimization

for $k = 0, 1, 2, \ldots$ **do**
  solve $\nabla^2 f(x_k) p_k = -\nabla f(x_k)$ for p_k
  $x_{k+1} = x_k + p_k$
  check convergence
end for

Geometrically, p_k is picked such that $x_k + p_k$ is a stationary point of

$$m_k(x_k + p) = f(x_k) + p^T \nabla f(x_k) + \frac{1}{2} p^T \nabla^2 f(x_k) p$$

this is like a model function (second order approximation) of f at x_k. (Take derivative with respect to p). However, this may be a bad model for f, only good locally.

Lecture 27: (4/12)

If in the Newton's method for unconstrained minimization, assuming $\nabla^2 f(x_k)$ is positive definite, then picking $x_{k+1} = x_k + p_k$ corresponds to setting x_{k+1} as the minimizer of

$$m_k(x) = f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{1}{2} (x - x_k)^T \nabla^2 f(x_k)(x - x_k)$$

so in words, once we are close enough to a minimizer where we have positive definite Hessian, then every iterate is just the minimizer of the local quadratic approximation to the function at the previous iterate.

Theorem 12 (Convergence of Newton's Method)**.** *If f has a local minimizer x^* and is C^2 in some neighborhood around x^*, and $\nabla^2 f(x)$ satisfies $\left\| \nabla^2 f(x) - \nabla^2 f(y) \right\|_2 \leq L \|x - y\|_2$ for some $L > 0$ in some neighborhood of x^* (Hessian is locally Lipschitz continuous), with $\nabla^2 f(x^*)$ positive definite and $\nabla f(x^*) = 0$, then the iterates of Newton's method satisfy*

1. *if x_0 is close enough to x^*, the sequence $(x_k)_k$ converges to x^**
2. *the rate of convergence is quadratic*
3. *the sequence $\left(\|\nabla f(x_k)\|_2 \right)_k$ converges quadratically to zero*

Far from x^*, minimizing m_k may not be a good choice, like when $f(x_{k+1}) > f(x_k)$, or maybe we are far enough so that $\nabla^2 f(x_k)$ is not positive definite.

Even if our quadratic model is bad globally, can we at least find a direction p_k to move downhill?

$$f(x + p) = f(x) + \nabla f(x + tp)^T p \quad t \in (0, 1)$$

observe that for a vector p such that $\nabla f(x)^T p < 0$, there exists some T such that $f(x+\gamma p) < f(x)$ for all $\gamma \in (0,T)$.

At a point x, any vector p that satisfies $\nabla f(x)^T p < 0$ is a **descent direction**. This motivates search direction methods for minimizing a function.

Algorithm 28 Search direction methods

for $k = 0,1,2\ldots$ **do**
find p_k, a descent direction at x_k
pick $\alpha_k > 0$, how far to move in that direction
set $x_{k+1} = x_k + \alpha_k p_k$
check convergence
end for

We see that Newton's method fits into this framework: First we choose $p_k = -\nabla^2(x_k)^{-1}\nabla f(x_k)$. If $\nabla^2 f(x_k)$ is positive definite, then p_k is a descent direction. To see this, note that $\nabla f(x_k)^T p_k = -\nabla f(x_k)^T \nabla^2 f(x_k)^{-1} \nabla f(x_k)$. Since $\nabla^2 f(x_k)^{-1}$ is positve definite, we have that this is negative so that p_k is indeed a descent direction.

Lastly, Newton's method picks the step size $\alpha_k = 1$.

Now since we want $\nabla f(x_k)^T p_k < 0$, if we suppose $\nabla f(x_k) \neq 0$ (since otherwise we would be checking if x_k is a local minimizer), then $p_k = -\nabla f(x_k)$ works.

Algorithm 29 Gradient descent

for $k = 0,1,2,\ldots$ **do**
$p_k = -\nabla f(x_k)$
pick α_k
$x_{k+1} = x_k + \alpha_k p_k$
check convergence
end for

Now, we consider how to pick α_k. This process is known as **line search**. Once we have picked p_k, we consider $\phi(\alpha) = f(x_k + \alpha p_k)$ for $\alpha > 0$. We can then consider problems such as minimizing $\phi(\alpha)$, but not necessarily this problem explicitly.

We ask that $\phi(0) > \phi(\alpha)$ at the moment, as this is very reasonable.

Lecture 28: (4/15)

In regards to choosing the step size α_k, we know that in Newton's method, $\alpha_k = 1$, since this corresponds to minimizing the quadratic approximation that we were minimizing. However, in gradient descent, we have a linear approximation, so there is no natural choice of α_k; there is no minimum of the linear approximation.

Recall that our problem is, given a search direction p_k, consider $\phi(\alpha) = f(x_k + \alpha p_k)$, where $\alpha > 0$. Ideally, we wish to solve $\min_{\alpha>0} \phi(\alpha)$, but this can be very hard to solve.

Let us get some criteria for picking α—we want:

- $\phi(\alpha_k) < \phi(0)$ (locally improving)

- $\phi(\alpha_k) \leq \phi(0) + c_1\alpha_k\nabla f(x_k)^T p_k$ (not too small of a step)
  i.e. $f(x_{k+1}) \leq f(x_k) + c_1\alpha_k\nabla f(x_k)^T p_k$

in the second requirement, note that the amount required to decrease by is relative to the gradient. If the gradient is smaller, then the amount required to decrease is smaller. Moreover, note that the requirement for decrease is a linear function of α with negative slope, so that as α increases, and we are choosing a high step size, we require a larger decrease in f.

If desired, we normalize our search direction $\|p_k\|_2 = 1$, but this will be taken into account later anyway.

We choose $0 < c_1 < 1$. Note that c_1 must be less than 1. No natural way to choose c_1, and it is often picked `1e-4` or `1e-3`. Moreover, it is often chosen constant, but does not have to be. It is not absolutely necessary to change c_1 over iterations since the rest of the terms are scaled by iteration anyway.

This still does not rule out short steps. Thus, we introduce another condition that does this. We add a curvature condition, that α_k must satisfy:

$$\nabla f(x_k + \alpha_k p_k)^T p_k \geq c_2 \nabla f(x_k)^T p_k \quad c_2 \in (c_1, 1)$$

$$\text{equivalently, } \phi'(\alpha_k) \geq c_2\phi'(0)$$

It is also common to symmetrically ask for no large positive slopes, but we do not really consider that here. i.e. could ask $|c_2\phi'(0)| \geq \phi'(\alpha_k)$.

c_2 is often picked near .9. Note that taking c_2 closer to 1 makes this an easier condition to satisfy.

The two conditions above, for some $0 < c_1 < c_2 < 1$, are known as the Wolfe conditions.

Say f is continuously differentiable, p_k is a descent direction, and f is bounded from below along the line $x_k + \alpha p_k$, $\alpha > 0$. Then there exists an $\alpha > 0$ that satisifes the Wolfe conditions.

Algorithm 30 Backtracking Line Search

```
Given α_max ≤ 1, ρ ∈ (0,1), max_steps
Set α = α_max
for k = 1,2,..., max_steps do
    if α satisfies Wolfe then
        return α
    else
        α = ρα
    end if
end for
```

Lecture 29: (4/17)

Note the second Wolfe condition is kinda like approaching a local minimum, as in this case the gradient would be zero. Also, the first Wolfe condition is in general easier to satisfy than the second. Thus, the second Wolfe condition is occasionally omitted.

Note that our above line search is just one method to search for α satisfying the conditions. For instance, another method is given by binary search. One reason why backtracking is popular: we can pair it with Newton and set $\alpha_{max} = 1$.

Convergence criteria:

- $\|\nabla f(x_k)\|_2 \leq \texttt{ftol}$
- $\|x_{k+1} - x_k\|_2 \leq \texttt{atol}$
- $\|x_{k+1} - x_k\|_2 \leq \texttt{rtol}\|x_k\|_2$
- $\|x_{k+1} - x_k\|_2 \leq \texttt{tol}(1 + \|x_k\|_2)$

this set of criteria tell if we have converged to a stationary point. To tell if we converged to a local minima, need to look at the hessian $\nabla^2 f(x_k)$. The Wolfe conditions help with this because they guarantee that the function value decreases at each iteration. However, counterexamples that converge to local maxima can still be constructed.

Now, for the problem of $\min_x f(x)$, we have Since the B_k are spd, the p_k are search directions. For

---

Given $x_0, f, B_0 \in \mathbb{R}^{n \times n}$ symmetric positive definite,
for $k = 0, 1, 2, \ldots$ **do**
  Solve $B_k p_k = -\nabla f(x_k)$
  Pick α_k
  $x_{k+1} = x_k + p_k$
  Check convergence
  Construct B_{k+1} symmetric positive definite.
end for

$B_k = I$, this is gradient descent, so that solving for p_k is cheap. If $B_k = \nabla^2 f(x_k)$, we have Newton's method, which converges very nicely locally, but is more expensive since need to compute $\nabla^2 f(x_k)$ and have to solve a nontrivial linear system.

In between these two choices are **Quasi-Newton methods**. The idea is to update B_k at each step to incorporate some information about $\approx \nabla^2 f(x_k)$. Kinda somewhat analogous to replacing the derivative by the secant approximation in the secant method.

Lecture 30: (4/19)

Interpretation for the above: For a B_k spd, p_k is chosen to take x_k to the global minimizer of the model function

$$m_k(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T B_k p$$

so that $p_k = -B_k^{-1} \nabla f(x_k)$.

Given B_k, how to move to B_{k+1}? We will use information about the gradients we know to update it. Our model function at the new step is

$$m_{k+1}(p) = f(x_{k+1}) + \nabla f(x_{k+1})^T p + \frac{1}{2} p^T B_{k+1} p$$

we pick B_{k+1} so that ∇m_{k+1} matches ∇f and x_k and x_{k+1}. We want that $\nabla m_k(-\alpha_k p_k) = \nabla f(x_k)$. Note that $\nabla m_{k+1}(0) = \nabla f(x_{k+1})$, so by construction we get that ∇m_{k+1} matches ∇f at x_{k+1} for free.

Define $s_k = x_{k+1} - x_k = \alpha p_k$, and define $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$. We have

$$\nabla m_{k+1}(-\alpha_k p_k) = \nabla f(x_{k+1}) - B_{k+1}\alpha_k p_k$$

We want this to be equal to $\nabla f(x_k)$. It is necessary that

$$B_{k+1}\alpha_k p_k = \nabla f(x_{k+1}) - \nabla f(x_k)$$

$$B_{k+1}s_k = y_k$$

this is know as a **secant condition**. Note that for B_{k+1} to be spd, we must have $y_k^T s_k > 0$, because $s_k^T B_{k+1} s_k = s_k^T y_k$.

Now, we consider if this is feasible to satisfy in an algorithm. We could for instance request for this condition to be satisfied in the line search. In fact, the second Wolfe condition, the curvature condition, enforces this condition. This is because we can write the curvature condition as

$$\nabla f(x_{k+1})^T s_k \geq (c_2 - 1)\alpha_k \nabla f(x_k)^T p_k$$

Thus, given B_k, we pick $B_{k+1}s_k = y_k$. Note that the matrix has $\approx n^2/2$ degrees of freedom, while there are only n linear constraints. There are infinitely many spd B_{k+1} that satisfy this. To determine which one to pick, we will consider two techniques.

Of all possible B_{k+1}, we can pick the one closest to B_k in some sense:

$$B_{k+1} \quad \text{solves} \quad \min \|B - B_k\|_W = \left\|W^{-1/2}(B - B_k)W^{-1/2}\right\|_F$$
$$\text{s.t.} \quad Bs_k = y_k, \quad B \ \text{ spd}$$

note the use of the weighted Frobenius norm. Any choice of norm gives a **Quasi-Newton method**. There are good choices of W. With special choices of W, we get closed form solutions to this problem. Note that is we just naively used say the Eucldean norm, then we would not have a closed form solution (singular value).

For a nice choice of W, yields **DFP** (Davidson-Fletcher-Powell):

$$B_{k+1} = (I - \rho_k y_k s_k^T)B_k(I - \rho_k s_k y_k^T) + \rho_k y_k y_k^T$$
$$\rho_k = \frac{1}{y_k^T s_k}$$

can verify that $B_{k+1}s_k = y_k$.

Note that we only solve a linear system with B_k. It is not important to know it exactly. Thus, we can choose to maintain and update "inverses" of B_k. Let $H_k = B_k^{-1}$. Now, we can pick B_{k+1} and therefore H_{k+1} as

$$\min \|H - H_k\|_W$$
$$\text{s.t.} \quad H^{-1}s_k = y_k \quad H \text{ spd}$$

Doing this yields the method **BFGS** (Broyden, Fletcher, Goldfarb, Shanno). This has a closed form solution

$$H_{k+1} = (I - \rho_k s_k y_k^T)H_k(I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T$$
$$\rho_k = \frac{1}{y_k^T s_k}$$

Algorithm 31 BFGS

given x_0, H_0
for $k = 0, 1, 2, \ldots$ **do**
  $p_k = -H_k \nabla f(x_k)$
  pick α_k via line search
  $x_{k+1} = x_k + \alpha_k p_k$
  check convergence
  Compute H_{k+1} via the above
end for

Lecture 31: (4/22)

Note that in general, we will get convergence of the iterates long before convergence of the B_k to the Hessian.

Theorem 13 (Convergence of BFGS)**.** *Assume $\nabla^2 f$ is Lipschitz continuous in a neighborhood of a strong local minimizer x^*. Then if BFGS converges to x^*, then the convergence is superlinear. i.e. $x_k \to x^*$ superlinearly.*

If $f \in C^2$, and α_k is picked to satisfy the Wolfe conditions with $c_1 \leq \frac{1}{2}$, if $x_k \to x^*$, with $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is spd, and if

$$\lim_{k \to \infty} \frac{\left\| (B_k - \nabla^2 f(x_k)) p_k \right\|_2}{\|p_k\|_2}$$

then $\alpha_k = 1$ is admissable for all $k > k_0$ and if $\alpha_k = 1$ for $k > k_0$, then $x_k \to x^*$ superlinearly.

Note the analogy to the superlinearity of the secant method. In the secant method we approximate the derivative. In BFGS we approximate the Hessian. Also, note that the limit condition is that the B_k become good approximations of the Hessian in the direction of the steps that are taken, and relative to the length of the steps.

Now we will consider **modified Newton methods**, which deal with what happens when the Hessian is not positive definite away from the solution x^*.

Algorithm 32 Modified Newton Method

given x_0, f
for $k = 0, 1, 2, \ldots$ **do**
  $B_k = \nabla^2 f(x_k) + E_k$ is spd
  solve $B_k p_k = -\nabla f(x_k)$
  pick α_k
  $x_{k+1} = x_k + \alpha_k p_k$
  check convergence
end for

We can pick $E_k = \gamma I$, with $\gamma \geq 0$, such that $\lambda_{\min}(\nabla^2 f(x_k) + \gamma I) \geq \delta > 0$.

Another method is to find $\nabla^2 f(x_k) = V \Lambda V^T$, and only modify the ones that are negative. $B_k = V \tilde{\Lambda} V^T$, where $\tilde{\Lambda} = \max(\Lambda, \delta I)$, $\delta > 0$.

Computationally, Quasi-Newton methods are nicer than Newton's method, since we need only computer $\nabla f(x_k)$ and not the Hessian, and also we need only compute a matrix multiplication at every step instead of a linear system solve.

Lecture 32: (4/24)

First we will look at a specific example of optimization, with

$$f(x) = f_1(x)f_2(x)f_3(x)$$

$$f_i(x) \geq 0 \qquad f_i(x) = (x_1 - r_{i,1})^2 + (x_2 - r_{i,2})^2$$

so that f is nonnegative, and f is 0 if and only if at least one of the $f_i(x)$ are 0.

We place the roots on the third roots of unity, and see that Newton's method has crazy regions (of initial choices) of convergence, large regions with no convergence, and disconnected regions of convergence. Modified Newton's method gives nice connected regions of convergence. BFGS with the identity as the initial is absolutely wild, with points of no convergence all over the place. BFGS with the true Hessian (with constant added to the diagonal as needed) as the initial looks more like the regions for Newton's method, with still a large region of non-convergence.

Before we were considering line search methods. Now, we consider trust region methods. These have a distance Δ_k which is how far one is willing to move in a given iteration. $\|x_{k+1} - x_k\| \leq \Delta_k$.

How to choose x_{k+1}? Pick a model function $m_k(x)$ for $f(x)$ at x_k. Then x_{k+1} is picked to solve

$$\min_{\|x - x_k\| \leq \Delta_k} m_k(x)$$

for example, if $m_k(x) = f(x_k) + \nabla f(x_k)^T(x - x_k) + \frac{1}{2}(x - x_k)^T \nabla^2 f(x_k)(x - x_k)$, which is just like our model function for Newton's method. Recall that in Newton's, there are issues if the Hessian is not spd. If the Hessian is in fact negative definite, then Newton's steps towards a local maximizer. However, in trust region methods, since we do not have a fixed step direction, f will still (probably) be getting smaller in the iteration.

Algorithm 33 Trust Region Methods

Given x_0, δ_0, f
for $k = 0, 1, 2, \ldots$ **do**
  $x_{k+1} = \text{argmin}_{\|x - x_k\| \leq \Delta_k} m_k(x)$
  Update from Δ_k to Δ_{k+1}
  Check for convergence
end for

How to update Δ_k? Define $\rho_k = \frac{f(x_k) - f(x_{k+1})}{m_k(x_k) - m_k(x_{k+1})}$. This measures how much the function decreases in the step versus the decrease predicted by the model. Note that the denominator is nonnegative by choice of x_{k+1}. It is good if $\rho_k \approx 1$, since then the model is a good predictor.

Thus, our strategy is: If $\rho_k \geq 1 - \delta$, then increase Δ_{k+1}.

If $\rho_k \leq c < 1$ decrease Δ_{k+1}.

If $c < \rho_k < 1 - \delta$, keep Δ_{k+1} the same.

Note that often the model functions are chosen so that there are closed form solutions.

Lecture 33: Constrained Optimization (4/26)

Now we will move onto considering $\min_{x\in\Omega} f(x)$, minimizing over a region Ω. For instance, we can consider

$$\min_x f(x)$$
$$\text{s.t.} \quad x_i \geq 0 \quad i = 1, 2, \ldots, n$$

in our case, we will only consider Ω of the form

$$\Omega = \{x \in \mathbb{R}^n : \underbrace{c_i(x) = 0, i \in E,}_{\text{equality constraints}} \quad \underbrace{c_i(x) \geq 0, i \in I}_{\text{inequality constraints}} \}$$

we will not consider strict inequality constraints. Any $x \in \Omega$ is called a **feasible point**.

If we have a problem in which is (strong) local minimizer is strictly interior to Ω, we may be able to use existing methods (perhaps with minor modification). However, we will be more interested in the other case, in which there is no (strong) local minimizer in the interior of Ω.

A point $x^* \in \Omega$ is a **local minimizer** of $\min_{x\in\Omega} f(x)$ is there exists a neighborhood N of x^* such that $f(x^*) \leq f(x)$ for all $x \in N \cap \Omega$. It is strict if $f(x^*) < f(x)$ for all $x \in N \cap \Omega \setminus \{x^*\}$.

For any point $x \in \Omega$, we define $\mathcal{A}(x)$, the set of **active constraints** at x.

$$\mathcal{A}(x) = E \cup \{i \in I : c_i(x) = 0\}$$

We will make some assumptions on the problem: $f \in C^1$ and $c_i(x) \in C^1$. Note that this assumption is that each $c_i(x)$ is independently C^1. We can still get Ω with rough edges by combining several constraints.

Let us consider $\min_{x\in\Omega} x_1 + x_2$ with $\Omega = \{x : x_1^2 + x_2^2 = 1\}$. The solution is given by $(\frac{-\sqrt{2}}{2}, \frac{-\sqrt{2}}{2})$. Consider $c(x) = x_1^2 + x_2^2 - 1$, and note $\nabla c(x) = \begin{bmatrix} 2x_1 & 2x_2 \end{bmatrix}$. Thus ∇c is normal to the surface of the circle. For an extreme point on the boundary, we need $\nabla c = \lambda \nabla f$, i.e. for the two be parallel.

Let us make the assumption that for any $x \in \Omega$, let $A(x)^T$ be the matrix whose columns are $\nabla c_i(x)$ for $i \in \mathcal{A}(x)$. We say the **constraint qualification condition** holds at x if $A(x)^T$ has full column rank. This guarantees that our set of constraints does not have redundancies at the point x. Define the **Lagrangian** $\mathcal{L}(x, \lambda) = f(x) - \sum_{i\in E\cup I} \lambda_i c_i(x)$. Then we have the **first order optimality (KKT = Karush, Kuhn, Tucker) conditions**. Suppose $x^* \in \Omega$ is a local minimizer of f, and constraint qualifications hold at x^*. Also, suppose that f, c_i are continuously differentiable. Then there exists a vector λ^* (of Lagrange multipliers) such that

- $\nabla_x \mathcal{L}(x^*, \lambda^*) = 0$
- $c_i(x^*) = 0$ for $i \in E$
- $c_i(x^*) \geq 0$ for $i \in I$
- $\lambda_i^* \geq 0$ for $i \in I$
- $\lambda_i^* c_i(x^*) = 0$ for $i \in E \cup I$

Lecture 34: (4/29)

Say that at a candidate minimizer x^*, we have $\mathcal{A}(x^*) = \emptyset$, so no active constraints. Then we can treat this as an unconstrained minimization problem. Now, we consider what happens when $\mathcal{A}(x^*)$ is nonempty.

Let us again consider $\min_{x\in\Omega} x_1 + x_2$, where $\Omega = \{x_1^2 + x_2^2 - 1 = 0\}$. Consider a point x with $c_1(x) = x_1^2 + x_2^2 - 1 = 0$. For small s we have $c_1(x+s) \approx c_1(x) + \nabla c_1(x)^T s = \nabla c_1(x)^T s$. Thus, if we want to move in some way s while staying on the constraint, then we need s to be orthogonal to the gradient $\nabla c_1(x)$, since we need $c_1(x+s) = 0$. For a direction s to be a descent direction, need $\nabla f(x)^T s < 0$. This is why to first order, at an optimum, we necessarily need existence of λ such that $\nabla f = \lambda \nabla c_1$, since in this case we cannot move while satisfying our constraint and lowering f.

Say we have a problem with one inequality constraint

$$\min f(x)$$

$$\text{s.t. } c_1(x) \geq 0$$

given an x such that $c_1(x) = 0$, can we find an s such that $c_1(x+s) \geq 0$ and $f(x+s) < f(x)$? Now, for $f(x+s) < f(x)$ need $\nabla f(x)^T s < 0$. For $c_1(x)$ we have $c_1(x+s) \approx c_1(x) + \nabla c_1(x)^T s = \nabla c_1(x)^T s$. We can either not change c_1, or increase it, to keep satisfying the constraint. Thus, we need $\nabla c_1(x)^T s \geq 0$. Hence, any valid s, (to first order) satsifies $\nabla c_1(x)^T s \geq 0$ and $\nabla f(x)^T s < 0$. In this case, we necessarily need existence of a $\lambda \geq 0$ such that $\nabla f(x) = \lambda \nabla c_1(x)$.

Note that in the KKT conditions, if there are no active constraints at x, then since $\nabla_x \mathcal{L}(x, \lambda) = \nabla f(x) = 0$, so this is our first order condition in unconstrained minimization.

Lecture 35: Quadratic Programming (5/1)

Now we consider a (not fully general) quadratic programming problem

$$\min_x \frac{1}{2} x^T H x - d^T x$$

$$\text{s.t. } Ax - b$$

where we assume H is symmetric, $A \in \mathbb{R}^{m\times n}$, $m < n$, and A has full row rank. Note that the full row rank assumption is not really restrictive since we can remove redundant rows when there is linear dependence.

$\mathcal{L}(x, \lambda) - \frac{1}{2} x^T H x - d^T x - \lambda^T (b - Ax)$. From KKT, we know that at a solution, the gradient of the Lagrangian is zero, so $Hx - d + A^T \lambda = 0$. We also know that at a solution, $b - Ax = 0$. Thus, we have

$$\begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} d \\ b \end{bmatrix}$$

Note that this is a square linear system of size $n + m \times n + m$. Now, we ask when this linear system has a unique solution. If H is symmetric positive definite, the matrix can still be indefinite. Call the matrix K. K is nonsingular if $y^T H y \neq 0$ for all nonzero $y \in \ker A$.

Algorithm overviews:

Active set methods look for solutions by moving along the boundary of Ω, adding and removing constraints from the active set.

Penalty methods. Suppose we only have equality constraints. Define
$\psi(x,\mu) = f(x) + \frac{1}{2\mu}\sum_{i\in E} c_i(x)^2$. Thus, we pay a penalty to move away from the constraints. thus,

Algorithm 34 Penalty methods

a sequence $(\mu_k)^N, \quad \mu_k \to 0$
for $k = 1,2,\ldots$ **do**
  $x_{k+1} \leftarrow \min_x \psi(x,\mu_k)$, using x_k as initial guess
end for

this solves a sequence of unconstrained problems.

Barrier methods. Now, say we only have inequality constraints, and define
$\phi(x,\mu) = f(x) - \mu\sum_{i\in I}\log c_i(x)$

Algorithm 35 Barrier methods

start with x in interior of Ω
solve a sequence of unconstrained problems
use solution at μ_k to initialize the problem with M_{k+1}.

Perhaps the most general methods for these optimization problems are know as sequential quadratic programs (SQP). Given (x_0, λ_0)

Algorithm 36 Barrier methods

given
for $k = 0,1,2,3,\ldots$ **do**

$$solve \min_p f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla_{xx}\mathcal{L}(x_0) p$$

$$\nabla c_i(x_k)^T p + c_i(x_k) = 0,\ i \in E$$

$$\nabla c_i(x_k)^T p + c_i(x_k) \geq 0,\ i \in I$$

  Set $x_{k+1}, \lambda_{k+1} = (x_k + p,$ Lagrange multipliers from solution of above)
end for

Lecture 36: Types of Optimization Problems (5/3)

One specific instance of constrained optimization is linear programming, given as

$$\min_x\ c^T x$$

$$\text{s.t. } Ax = b$$

$$x \geq 0$$

the objective is linear and the feasible set is a polytope. Since the objective f is linear, if f is bounded from below, all local minimizers occur on the boundary of the set. The simplex method

moves along the boundary to find the minimizer. Also, we have interior point (primal-dual) methods that move through the interior of the feasible set towards a solution on the boundary.

In **convex optimization**, we consider optimization of convex functions f over convex sets Ω. A set S is **convex** if for all $x, y \in S$, we have $\alpha x + (1-\alpha)y \in S$ for all $\alpha \in [0,1]$. A function f is **convex** if its domain S is convex and for all $x, y \in S$, $f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y)$ for all $\alpha \in [0,1]$. f is strictly convex if the inequality is strict.

If f and Ω are convex, then any local solution is also a global solution.

Now, we consider nonlinear least squares. Before, we were considering $\min \frac{1}{2}\|Ax-b\|_2^2$. What if we knew $v(t) = x_1 e^{x_2 t}\cos(x_3 t)$, and we measure samples $b_i = v(t_i)$+noise for some $t_1, \ldots, t_m$ and some specific x_1^*, x_2^*, x_3^*. we want to fit the parameters x_1, x_2, x_3. We can express this problem as

$$\min_x \frac{1}{2}\|g(x) - b\|_2^2$$

where $g_i(x) = v_x(t_i)$.

Algorithm 37 Gauss-Newton algorithm for nonlinear least squares

given x_0
for $k = 0, 1, 2 \ldots$ **do**
  solve $p_k = \text{argmin}_p \frac{1}{2}\|J_g(x_k)p - (b - g(x_k))\|_2^2$
  set $x_{k+1} \leftarrow x_k + p_k$
end for

note that the idea is in linearizing the function g utilizing the derivative J_g around x_k.

Lecture 37: Randomized Numerical Linear Algebra (5/6)

We consider the problem of computing low-rank approximations. Given $A \in \mathbb{R}^{n\times n}$, we say A is well approximated by a matrix of rank-k if there are matrices $W, Z \in \mathbb{R}^{n\times k}$ such that $A \approx WZ^T$. Recall the optimal rank-k approximation is the k-truncated SVD.

Our idea for solving this problem is the capture the range of A. Say we could do this and get an orthonormal basis $Q \in \mathbb{R}^{n\times k}$ for the range of A or $\approx \text{span}(U_k)$. Given Q, it is good if $A \approx QQ^TA$, so that $\left\|(I - QQ^T)A\right\|_2$ is small. Note that if $Q = U_k$, the optimal choice, then the norm is σ_{k+1}.

Now, say that $A \approx QQ^TA = Q(\underbrace{Q^TA}_{B})$. Now, take the reduced SVD $B = \tilde{U}\Sigma V^T$, so $\tilde{U}$ and Σ are $k \times k$, and V^T is $k \times n$. Now, plugging this back in to $QQ^TA = QB$, we have $QQ^TA = Q\tilde{U}\Sigma V^T$, so that this is in fact an SVD for QQ^TA. Thus, we have $A \approx Q\tilde{U}\Sigma V^T$.

Thus, our general recipe is to find a Q that approximates the range of A, and then use Q to build some sort of standard matrix factorization.

To find Q, we use randomness. Take a random vector $w \in \mathbb{R}^n$, with entries drawn iid $N(0,1)$ entries. Construct $y = Aw$. This is like a sample of the range of A in some sense. Now, say we are given multiple random vectors $w^{(i)}$, $i = 1, \ldots, k$. Then we get different vectors $y^{(i)} = Aw^{(i)}$.

However, A is not rank-k. Say $A = H + E$, where H is rank-k and E is small. Thus, we may use $w^{(i)}$ for $i = 1, 2, \ldots, k+p$ for p a small parameter.

Algorithm 38 Approximate range

Given A, k, p
Construct $\Omega \in \mathbb{R}^{n \times k+p}$ with iid $N(0,1)$ entries
$Y = A\Omega$
$Y = QR$

Note that Q is an orthonormal basis for the range of Y. This is one way to choose a Q, and is a provably good way in some sense. For instance, $\mathbb{E}\left[\left\|(I - QQ^T)A\right\|\right] = \left(1 + \frac{4\sqrt{k+p}}{p-1}\sqrt{n}\right)\sigma_{k+1}$, so in expectation we are somewhat close to σ_{k+1}, the optimal approximation error.