CME 307 / MS&E 311: Optimization

Least squares

Professor Udell

Management Science and Engineering
Stanford

April 2, 2023

Linear system

find $x \in \mathbf{R}^n$ such that

$$Ax = b$$

given **design matrix** $A \in \mathbf{R}^{m \times n}$, **righthand side** (rhs) $b \in \mathbf{R}^m$

how to solve?

- factor and solve
  - QR
  - singular value decomposition (SVD)
  - Cholesky (for symmetric A)
- iterative methods
  - conjugate gradient (CG) (for symmetric A)
  - iterative refinement

we will talk about QR, CG, and iterative refinement

Regularized linear system

find $x \in \mathbf{R}^n$ such that

$$(A + \mu I)x = b$$

where $A \in \mathbf{S}^n_+$, $b \in \mathbf{R}^m$, and $\mu \geq 0$.

- eigenvalues of A $\lambda_1 \geq \cdots \geq \lambda_n$
- condition number $\kappa(A) = \lambda_1(A)/\lambda_n(A)$
- regularized matrix $A_\mu = A + \mu I$ has $\kappa(A_\mu) \leq \kappa(A)$

Why solve a regularized linear system?

find $x \in \mathbf{R}^n$ such that

$$(A + \mu I)x = b$$

- iterative reweighted least squares
- (kernel) ridge regression
- Gaussian processes
- approximate cross validation [**stephenson2020LowRank**]
- influence functions [**koh2017understanding**]
- hyperparameter optimization [**lorraine2019optimizing**]
- ...

How to solve a regularized linear system?

direct methods, *e.g.*, Cholesky:

- factor $A = LL^T$ into easy-to-solve (*e.g.*, triangular) matrices, then solve
- $O(n^3)$ flops, require entrywise access to A

How to solve a regularized linear system?

direct methods, *e.g.*, Cholesky:

- factor $A = LL^T$ into easy-to-solve (*e.g.*, triangular) matrices, then solve
- $O(n^3)$ flops, require entrywise access to A

indirect methods, *e.g.*, conjugate gradient (CG):

- main work per iteration is Ar where $r = Ax - b$ is residual
- at kth iteration, finds x in kth Krylov subspace $\mathcal{K}_k = \mathrm{span}\{b, Ab, \dots, A^{k-1}b\}$
- $O(\sqrt{\kappa(A)}\log(\frac{1}{\epsilon}))$ matvecs to reach ϵ accuracy

How to solve a regularized linear system?

direct methods, *e.g.*, Cholesky:

- factor $A = LL^T$ into easy-to-solve (*e.g.*, triangular) matrices, then solve
- $O(n^3)$ flops, require entrywise access to A

indirect methods, *e.g.*, conjugate gradient (CG):

- main work per iteration is Ar where $r = Ax - b$ is residual
- at kth iteration, finds x in kth Krylov subspace $\mathcal{K}_k = \mathrm{span}\{b, Ab, \ldots, A^{k-1}b\}$
- $O(\sqrt{\kappa(A)}\log(\frac{1}{\epsilon}))$ matvecs to reach ϵ accuracy

series expansions for the inverse: (unstable!)

- Neumann series [**lorraine2019optimizing**]
- (stochastic) Taylor series [**koh2017understanding**; **agarwal2017secondorder**]

Considerations in choosing a method

- sparse or dense A?
- symmetric A or rectangular problem?
- conditioning of A?
- one problem, or many righthand sides b with the same design matrix A?

Optimality condition for least squares is a linear system

given $A \in \mathbf{R}^{m \times n}$, $y \in \mathbf{R}^m$. find x to solve

$$\text{minimize} \qquad \|Ax - b\|^2.$$

to solve, take gradient, set to 0. solution x satisfies **normal equations**

$$A^\top Ax = A^\top b.$$

a linear system! (with psd $A^\top A$.)

Outline

QR

Conjugate gradient

Preconditioned CG

Iterative refinement

How to solve a linear system?

never form the inverse explicitly: numerically unstable!

Corollary: never type `inv(A'*A)` or `pinv(A'*A)` to solve the normal equations.

How to solve a linear system?

never form the inverse explicitly: numerically unstable!

Corollary: never type `inv(A'*A)` or `pinv(A'*A)` to solve the normal equations.

Instead: compute the inverse using easier matrices to invert, like

- Orthogonal matrices Q:

$$a = Qb \iff Q^\top a = b$$

- Triangular matrices R:
  if $a = Rb$, can find b given R and a by solving sequence of simple, stable equations.

The QR factorization

every matrix A can be written using **QR decomposition** as
$A = QR$

- $Q \in \mathbf{R}^{n\times d}$ has orthogonal columns: $Q^\top Q = I_d$
- $R \in \mathbf{R}^{d\times d}$ is upper triangular: $R_{ij} = 0$ for $i > j$
- diagonal of $R \in \mathbf{R}^{d\times d}$ is positive: $R_{ii} > 0$ for $i = 1, \ldots, d$
- this factorization always exists and is unique
  (proof by Gram-Schmidt construction)

can compute QR factorization of X in $2nd^2$ flops

The QR factorization

every matrix A can be written using **QR decomposition** as $A = QR$

- $Q \in \mathbf{R}^{n \times d}$ has orthogonal columns: $Q^\top Q = I_d$
- $R \in \mathbf{R}^{d \times d}$ is upper triangular: $R_{ij} = 0$ for $i > j$
- diagonal of $R \in \mathbf{R}^{d \times d}$ is positive: $R_{ii} > 0$ for $i = 1, \ldots, d$
- this factorization always exists and is unique (proof by Gram-Schmidt construction)

can compute QR factorization of X in $2nd^2$ flops

use `LinearAlgebra.qr`:

```
Q,R = qr(X)
```

advantage of QR: it's easy to invert R!

QR for least squares

use QR to solve least squares: if $A = QR$,

$$\begin{aligned}
A^\top A x &= A^\top b \\
(QR)^\top QRx &= (QR)^\top b \\
R^\top Q^\top QRx &= R^\top Q^\top b \\
R^\top Rx &= R^\top Q^\top b \\
Rx &= Q^\top b \\
x &= R^{-1} Q^\top b
\end{aligned}$$

Computational considerations

never form the inverse explicitly: numerically unstable!

instead, use QR factorization:

- compute QR factorization of A ($2nd^2$ flops)
- to compute $x = R^{-1}Q^\top b$
  - form $z = Q^\top b$ ($2nd$ flops)
  - compute $x = R^{-1}z$ by back-substitution (d^2 flops)

Computational considerations

never form the inverse explicitly: numerically unstable!

instead, use QR factorization:

- compute QR factorization of A ($2nd^2$ flops)
- to compute $x = R^{-1}Q^\top b$
  - form $z = Q^\top b$ ($2nd$ flops)
  - compute $x = R^{-1}z$ by back-substitution (d^2 flops)

in julia (or matlab), the **backslash operator** solves least-squares efficiently (usually, using QR)

```
x = A \ b
```

in python, use `numpy.lstsq`

Demo: QR

https://colab.research.google.com/github/stanford-cme-307/demos/blob/main/lsq.ipynb

Sparse QR

complexity of QR depends on the sparsity of Q and R:

- compute QR factorization of A (?? flops)
- to compute $x = R^{-1}Q^\top b$
  - form $z = Q^\top b$ ($\mathbf{nnz}(Q)$ flops)
  - compute $x = R^{-1}z$ by back-substitution ($\mathbf{nnz}(R)$ flops)

Q-less QR

during QR, can compute $Q^\top b$ essentially for free!

- compute QR of $\begin{bmatrix} A & b \end{bmatrix}$.

Q-less QR

during QR, can compute $Q^\top b$ essentially for free!

- compute QR of $\begin{bmatrix} A & b \end{bmatrix}$.

or compute it afterwards without forming Q:

$$\begin{aligned} A^\top b &= (QR)^\top b = R^\top Q^\top b \\ R^{-1} A^\top b &= Q^\top b \end{aligned}$$

Cholesky and QR

consider **Gram matrix** $G = A^\top A \succeq 0$. if $A = QR$,

$$G = R^\top Q^\top QR = R^\top R$$

this construction gives **Cholesky factorization** of a spd matrix G

- factors spd matrix into triangular matrices
- Cholesky factors of $X^\top X$ have same structure as R

Sparse QR: exercise

- can you guess the sparsity of R given sparsity of A?
- can you change sparity of R by permuting columns of A?

Sparse QR: exercise

- can you guess the sparsity of R given sparsity of A?
- can you change sparity of R by permuting columns of A?

use ‘colamd‘ in Matlab, equivalents in Python and julia

Chordal fill-in

to analyze fill-in

- consider spd matrix, for simplicity
- interpret matrix as directed graph
- form clique tree
- identify fill-in

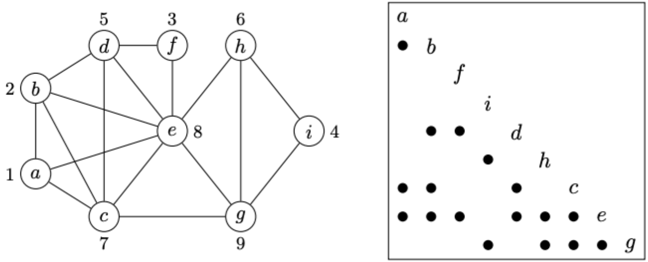


Figure 4.1: *Left.* Filled graph with 9 vertices. The number next to each vertex is the index $\sigma^{-1}(v)$. *Right.* Array representation of the same graph.

source: VA15,
https://www.seas.ucla.edu/ vandenbe/publications/chordalsdp.pdf

Outline

QR

Conjugate gradient

Preconditioned CG

Iterative refinement

Conjugate gradients

symmetric positive definite system of equations

$$Ax - b, \qquad A \in \mathbf{R}^{n \times n}, \qquad A = A^\top \succeq 0$$

Conjugate gradients

symmetric positive definite system of equations

$$Ax - b, \qquad A \in \mathbf{R}^{n \times n}, \qquad A = A^\top \succeq 0$$

why use conjugate gradients?

- uses only matrix-vector multiplies with A
  - useful for structured (from PDE or graph) or sparse matrices, easy to parallelize, ...
- most useful for problems with $n > 10^5$ or more
- converges exactly in n iterations
- converges approximately much faster
- quick-and-dirty solve is appropriate **inside** inner loop of optimization algo

Conjugate gradients

symmetric positive definite system of equations

$$Ax - b, \qquad A \in \mathbf{R}^{n \times n}, \qquad A = A^\top \succeq 0$$

why use conjugate gradients?

- uses only matrix-vector multiplies with A
  - useful for structured (from PDE or graph) or sparse matrices, easy to parallelize, ...
- most useful for problems with $n > 10^5$ or more
- converges exactly in n iterations
- converges approximately much faster
- quick-and-dirty solve is appropriate **inside** inner loop of optimization algo

other variants for indefinite (MINRES) or nonsymmetric matrices (GMRES)

Iterative methods for least squares

define

- (convex) objective $f(x) = (1/2)x^\top Ax - x^\top b$
- gradient $\nabla f(x) = Ax - b$
- condition number $\kappa(A) = \sigma_n(A)/\sigma_1(A)$
- bound $R \geq \|x_\star\|$ on norm of solution $x_\star$
- goal: find apx solution within accuracy $f(x) - f(x_\star) \leq \epsilon$

Iterative methods for least squares

define

- (convex) objective $f(x) = (1/2)x^\top Ax - x^\top b$
- gradient $\nabla f(x) = Ax - b$
- condition number $\kappa(A) = \sigma_n(A)/\sigma_1(A)$
- bound $R \geq \|x_\star\|$ on norm of solution $x_\star$
- goal: find apx solution within accuracy $f(x) - f(x_\star) \leq \epsilon$

how many iterations (matvecs) required?

Iterative methods for least squares

define

- (convex) objective $f(x) = (1/2)x^\top Ax - x^\top b$
- gradient $\nabla f(x) = Ax - b$
- condition number $\kappa(A) = \sigma_n(A)/\sigma_1(A)$
- bound $R \geq \|x_\star\|$ on norm of solution $x_\star$
- goal: find apx solution within accuracy $f(x) - f(x_\star) \leq \epsilon$

how many iterations (matvecs) required?

- conjugate gradient
  - $O\left(\sqrt{\kappa}\log(\frac{1}{\epsilon})\right)$

Iterative methods for least squares

define

- (convex) objective $f(x) = (1/2)x^\top Ax - x^\top b$
- gradient $\nabla f(x) = Ax - b$
- condition number $\kappa(A) = \sigma_n(A)/\sigma_1(A)$
- bound $R \geq \|x_\star\|$ on norm of solution $x_\star$
- goal: find apx solution within accuracy $f(x) - f(x_\star) \leq \epsilon$

how many iterations (matvecs) required?

- conjugate gradient
  - $O\left(\sqrt{\kappa}\log(\frac{1}{\epsilon})\right)$
- gradient descent (GD)
  - $O\left(\kappa\log(1/\epsilon)\right)$

Iterative methods for least squares

define

- (convex) objective $f(x) = (1/2)x^\top Ax - x^\top b$
- gradient $\nabla f(x) = Ax - b$
- condition number $\kappa(A) = \sigma_n(A)/\sigma_1(A)$
- bound $R \geq \|x_\star\|$ on norm of solution $x_\star$
- goal: find apx solution within accuracy $f(x) - f(x_\star) \leq \epsilon$

how many iterations (matvecs) required?

- conjugate gradient
  - $O\left(\sqrt{\kappa}\log(\frac{1}{\epsilon})\right)$
- gradient descent (GD)
  - $O\left(\kappa \log(1/\epsilon)\right)$
- accelerated gradient descent
  - $O\left(\sqrt{\kappa}\log(\frac{R^2}{\epsilon})\right)$ more generalizable, but more parameters to tune

source: **karimi2016unified**; **bubeck2014convex**

Residual

define **residual** $r = b - Ax$ at putative solution x

- $r = -\nabla f(x) = A(x_\star - x)$

Residual

define **residual** $r = b - Ax$ at putative solution x

- $r = -\nabla f(x) = A(x_\star - x)$

measures of error:

- objective function $f(x) - f(x_\star)$
- norm of residual $\|r\|$
- norm of gradient $\|\nabla f(x)\|$
- in terms of r, can compute error in objective

$$
\begin{aligned}
f(x) - f(x_\star) &= \|x - x_\star\|_A \\
&= \frac{1}{2}(x - x_\star)^\top A(x - x_\star) \\
&= \frac{1}{2}(r)^\top A^{-1}(r) \\
&= \|r\|_{A^{-1}}
\end{aligned}
$$

Krylov subspace

the Krylov subspace of dimension k is

$$\mathcal{K}_k = \mathrm{span}\{b, Ab, \ldots, A^{k-1}b\} = \mathrm{span}\{p_k(A)b \mid \mathit{degree}(p) < k\}$$

Krylov subspace

the Krylov subspace of dimension k is

$$\mathcal{K}_k = \mathrm{span}\{b, Ab, \ldots, A^{k-1}b\} = \mathrm{span}\{p_k(A)b \mid degree(p) < k\}$$

the iterates of the **Krylov sequence** $x^{(1)}, x^{(2)}, \ldots,$ minimize objective over successive Krylov subspaces

$$x^{(k)} = \underset{x \in \mathcal{K}_k}{\mathrm{argmin}}\, f(x) = \underset{x \in \mathcal{K}_k}{\mathrm{argmin}}\, \|Ax - b\| = \underset{x \in \mathcal{K}_k}{\mathrm{argmin}}\, \|x - x_\star\|_A$$

the CG algorithm generates the Krylov sequence

Properties of Krylov sequence

- $f(x^{(k+1)}) \leq f(x^{(k)})$ (but $\|r\|$ can increase)
- $x^{(n)} = x_\star$
- $x^{(k)} = p_k(A)b$, where p_k is a polynomial with degree $< k$
- less obvious: there is a two-term recurrence

$$x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)} \qquad \text{where} \qquad p^{(k)} = -r^{(k)} + \beta_k p^{(k-1)}$$

Properties of Krylov sequence

- $f(x^{(k+1)}) \leq f(x^{(k)})$ (but $\|r\|$ can increase)
- $x^{(n)} = x_\star$
- $x^{(k)} = p_k(A)b$, where p_k is a polynomial with degree $< k$
- less obvious: there is a two-term recurrence

$$x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)} \qquad \text{where} \qquad p^{(k)} = -r^{(k)} + \beta_k p^{(k-1)}$$

  - α_k and β_k are determined by the CG algorithm

Properties of Krylov sequence

- $f(x^{(k+1)}) \leq f(x^{(k)})$ (but $\|r\|$ can increase)
- $x^{(n)} = x_\star$
- $x^{(k)} = p_k(A)b$, where p_k is a polynomial with degree $< k$
- less obvious: there is a two-term recurrence

$$x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)} \qquad \text{where} \qquad p^{(k)} = -r^{(k)} + \beta_k p^{(k-1)}$$

  - α_k and β_k are determined by the CG algorithm
  - can derive recurrence from optimality conditions: each new iterate $x^{(k+1)}$ must have gradient (residual) orthogonal to $\mathcal{K}_k$

Coordinate descent does not solve in n iterations

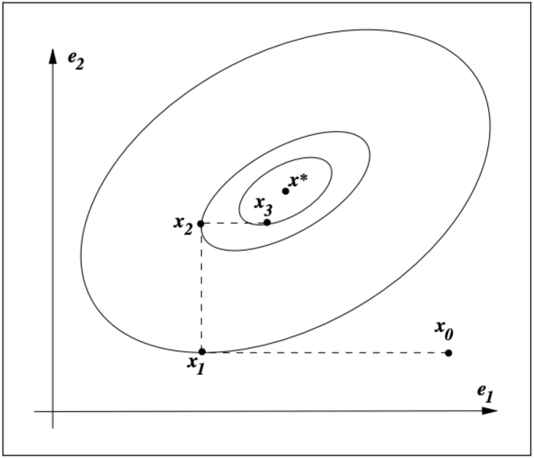


Figure 5.2 Successive minimization along coordinate axes does not find the solution in n iterations, for a general convex quadratic.

source: NW04

CG converges in $\mathrm{Rank}(A)$ iterations

write (don't compute!) SVD of $A = V\Lambda V^\top$ with

- $r = \mathrm{Rank}(A)$
- $\Lambda \in \mathbf{R}^r \times r$ diagonal and positive
- $V \in \mathbf{R}^{n\times r}$: orthonormal: $V^\top V = I_r$

CG converges in $\mathrm{Rank}(A)$ iterations

write (don't compute!) SVD of $A = V\Lambda V^\top$ with

- $r = \mathrm{Rank}(A)$
- $\Lambda \in \mathbf{R}^r \times r$ diagonal and positive
- $V \in \mathbf{R}^{n\times r}$: orthonormal: $V^\top V = I_r$

characteristic polynomial of Λ:

$$\xi(s) = \det(sI_r - \Lambda) = (s-\lambda_1)\cdots(s-\lambda_r) = s^r + \alpha s^{r-1} + \cdots + \alpha_r$$

Cayley-Hamilton theorem

$$\begin{aligned}\xi(\Lambda) = 0 &= \Lambda^r + \alpha_1\Lambda^{r-1} + \cdots + \alpha_r I_r \\ \Lambda^{-1} &= -(1/\alpha_r)(\Lambda^{r-1} + \alpha_1\Lambda^{r-2} + \cdots + \alpha_{r-1} I_r)\end{aligned}$$

CG converges in $\mathrm{Rank}(A)$ iterations

write (don't compute!) SVD of $A = V\Lambda V^\top$ with

- $r = \mathrm{Rank}(A)$
- $\Lambda \in \mathbf{R}^r \times r$ diagonal and positive
- $V \in \mathbf{R}^{n\times r}$: orthonormal: $V^\top V = I_r$

characteristic polynomial of Λ:

$$\xi(s) = \det(sI_r - \Lambda) = (s-\lambda_1)\cdots(s-\lambda_r) = s^r + \alpha s^{r-1} + \cdots + \alpha_r$$

Cayley-Hamilton theorem

$$\begin{aligned} \xi(\Lambda) = 0 &= \Lambda^r + \alpha_1\Lambda^{r-1} + \cdots + \alpha_r I_r \\ \Lambda^{-1} &= -(1/\alpha_r)(\Lambda^{r-1} + \alpha_1\Lambda^{r-2} + \cdots + \alpha_{r-1} I_r) \end{aligned}$$

write $A^{-1} = V\Lambda^{-1}V^\top$ in terms of this decomposition:

$$\begin{aligned} A^{-1} = V\Lambda^{-1}V^\top &= = -(1/\alpha_r)(V\Lambda^{r-1}V^\top + \alpha_1 V\Lambda^{r-2}V^\top + \cdots + \alpha_r \\ &= -(1/\alpha_r)(A^{r-1} + \alpha_1 A^{r-2} + \cdots + \alpha_{r-1} I) \end{aligned}$$

in particular, $x_\star = A^{-1}b \in \mathcal{K}_r$

Outline

QR

Conjugate gradient

Preconditioned CG

Iterative refinement

Preconditioning CG

for any $P \succ 0$,

$$\begin{aligned} Ax = b \quad \Longleftrightarrow \quad & P^{-1/2}Ax = P^{-1/2}b \\ & P^{-1/2}AP^{-1/2}z = P^{-1/2}b \end{aligned}$$

where $x = P^{-1/2}z$.

Preconditioning CG

for any $P \succ 0$,

$$\begin{aligned} Ax = b \quad \iff \quad & P^{-1/2}Ax = P^{-1/2}b \\ & P^{-1/2}AP^{-1/2}z = P^{-1/2}b \end{aligned}$$

where $x = P^{-1/2}z$.

- preconditioning works well when $\kappa(P^{-1/2}AP^{-1/2}) \ll \kappa(A)$

Preconditioning CG

for any $P \succ 0$,

$$
\begin{aligned}
Ax = b \quad \iff \quad & P^{-1/2}Ax = P^{-1/2}b \\
& P^{-1/2}AP^{-1/2}z = P^{-1/2}b
\end{aligned}
$$

where $x = P^{-1/2}z$.

- preconditioning works well when $\kappa(P^{-1/2}AP^{-1/2}) \ll \kappa(A)$

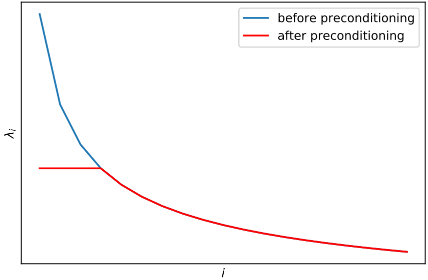
how to precondition?

- common heuristic: Jacobi preconditioning $P = \mathbf{diag}(A)$
- incomplete Cholesky (best for structured sparsity)

An optimal low-rank preconditioner

- suppose $\lfloor A \rfloor_s = V_s \Lambda_s V_s^T$ is a best rank-s apx to $A \in \mathbf{S}_+^n$.
- the best preconditioner using this information is

$$P_\star = \frac{1}{\lambda_{s+1}} V_s(\Lambda_s) V_s^T + (I - V_s V_s^T)$$



Outline

Iterative refinement

want to solve $Ax = b$.

given approximate solution $Ax^{(0)} \approx b$, for $k = 1, \ldots,$

- compute residual $r^{(k)} = b - Ax^{(k)}$
- use any method to solve $A\delta^{(k)} = r^{(k)}$
- $x^{(k+1)} = x^{(k)} + \delta^{(k)}$