Numerical Linear Algebra Primer

Ryan Tibshirani
Convex Optimization 10-725

Last time: proximal Newton method

Consider

$$\min_x \; g(x) + h(x)$$

where g, h convex, g twice differentiable, and h "simple". Proximal Newton method: let $x^{(0)} \in \mathbb{R}^n$, and repeat:

$$v^{(k)} = \operatorname*{argmin}_v \; \nabla g(x^{(k-1)})^T v + \frac{1}{2} v^T \nabla^2 g(x^{(k-1)}) v + h(x^{(k-1)} + v)$$
$$x^{(k)} = x^{(k-1)} + t_k v^{(k)}, \quad k = 1, 2, 3, \ldots$$

Step sizes are typically chosen by backtracking

- Iterations here are typically very expensive (computing $v^{(k)}$ is generally difficult)
- But typically very few iterations are needed until convergence: under appropriate conditions, get local quadratic convergence

Outline

Today:

- Flops for basic operations
- Solving linear systems
- Matrix factorizations
- Sensitivity analysis
- Indirect methods

Complexity of basic operations

Flop (floating point operation):

- One addition, subtraction, multiplication, division of floating point numbers
- Serves as a basic unit of computation
- We are interested in rough, not exact flop counts

Vector-vector operations: given $a, b \in \mathbb{R}^n$:

- Addition, $a + b$: costs n flops
- Scalar multiplication, $c \cdot a$: costs n flops
- Inner product, $a^T b$: costs $2n$ flops

Flops do not tell the whole story: setting every element of a to 1 costs 0 flops

Matrix-vector product: given $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^n$, consider Ab:

- In general: costs $2mn$ flops
- For s-sparse A: costs $2s$ flops
- For k-banded $A \in \mathbb{R}^{n\times n}$: costs $2nk$ flops
- For $A = \sum_{i=1}^r u_i v_i^T \in \mathbb{R}^{m\times n}$: costs $2r(m+n)$ flops
- For $A \in \mathbb{R}^{n\times n}$ a permutation matrix: costs 0 flops

Matrix-matrix product: for $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, consider AB:

- In general: costs $2mnp$ flops
- For s-sparse A: costs $2sp$ flops (less if B is also sparse)

Matrix-matrix-vector product: for $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, $c \in \mathbb{R}^p$, consider ABc:

- Costs $2np + 2mn$ flops if done properly (or $2mnp + 2mp$ if done improperly!)

Solving linear systems

For nonsingular $A \in \mathbb{R}^{n \times n}$, consider solving linear system $Ax = b$:

- In general: costs about n^3 flops—we'll see more on this later
- For diagonal A: costs n flops

$$x = (b_1/a_1, \ldots b_n/a_n)$$

- For lower triangular A ($A_{ij} = 0$, $j > i$): costs about n^2 flops

$$\begin{aligned} x_1 &= b_1/A_{11} \\ x_2 &= (b_2 - A_{21}x_1)/A_{22} \\ &\vdots \\ x_n &= (b_n - A_{n,n-1}x_{n-1} \ldots - A_{n1}x_1)/A_{nn} \end{aligned}$$

  This is called forward substitution
- For upper triangular A: costs about n^2, by back substitution

- For s-sparse A, often costs $\ll n^3$ flops, but exact (worse-case) flop counts are not known for abitrary sparsity structures
- For k-banded A: costs about nk^2 flops—more later
- For orthogonal A: we have $A^{-1} = A^T$, so $x = A^T b$ costs $2n^2$ flops
- For permutation A: again $A^{-1} = A^T$, so $x = A^T b$ costs 0 flops. Example:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad A^{-1} = A^T = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrix factorizations

As you've probably learned, we can solve $Ax = b$ by, e.g., Gaussian elimination. More typically, it is useful to instead factorize A:

$$A = A_1 A_2 \ldots A_k$$

and then compute $x = A_k^{-1} \ldots A_2^{-1} A_1^{-1} b$. Usually $k = 2$ or 3, and:

- Computing the factorization is expensive, about n^3 flops
- Applying $A_1^{-1}, \ldots A_k^{-1}$ is cheaper, about n^2 flops
- This is because $A_1, \ldots A_k$ are structured: either orthogonal, triangular, diagonal, or permutation matrices

Note: this is especially useful when we will be solving many linear systems in A. To solve t linear systems, after initial factorization, costs about tn^2

QR decomposition

Any $A \in \mathbb{R}^{m \times n}$, with $m \geq n$, has a QR decomposition:

$$A = QR$$

with $Q \in \mathbb{R}^{m \times n}$ orthogonal (i.e., $Q^T Q = I$), and $R \in \mathbb{R}^{n \times n}$ upper triangular. Can compute this in $2mn^2 - n^3/3$ flops

Property: number of nonzero diagonal elements of R is the rank of A, corresponding columns of Q span $\mathrm{col}(A)$

Assuming A is nonsingular and square, we can now solve $Ax = b$:

- Compute $y = Q^T b$, in $2n^2$ flops
- Solve $Rx = y$, in n^2 flops (back substitution)

So solving costs $3n^2$ flops

Cholesky decomposition

More specialized, any $A \in \mathbb{S}^n_{++}$, has a Cholesky decomposition:

$$A = LL^T$$

with $L \in \mathbb{R}^{n\times n}$ lower triangular. Can compute this in $n^3/3$ flops. (Could compute Cholesky from QR, but this would be inefficient)

From Cholesky factors, we can solve $Ax = b$:

- Compute $y = L^{-1}b$, in n^2 flops (forward substitution)
- Compute $x = (L^T)^{-1}y$, in n^2 flops (back substitution)

So solving costs $2n^2$ flops

An important extension for k-banded A: computing Cholesky takes $nk^2/4$ flops, and solving takes $2nk$ flops

Least squares problems and Cholesky

Given $y \in \mathbb{R}^n$, $X \in \mathbb{R}^{n \times p}$, consider the least squares problem:

$$\min_{\beta \in \mathbb{R}^p} \; \|y - X\beta\|_2^2$$

Assuming X has full column rank, solution is $\hat{\beta} = (X^T X)^{-1} X^T y$. How expensive?

- Compute $X^T y$, in $2pn$ flops
- Compute $X^T X$, in $p^2 n$ flops
- Compute Cholesky of $X^T X$, in $p^3/3$ flops
- Solve $(X^T X)\beta = X^T y$, in $2p^2$ flops

Thus in total, about $np^2 + p^3/3$ flops (or np^2 flops if $n \gg p$)

Least squares problems and QR

Same problem, now with QR. Key identity:

$$\|x\|_2^2 = \|P^T x\|_2^2 = \|Q^T x\|_2^2 + \|\tilde{Q}^T x\|_2^2$$

where $P = [Q\ \tilde{Q}] \in \mathbb{R}^{n \times n}$ is orthogonal. Applied to $x = y - X\beta$:

$$\|y - X\beta\|_2^2 = \|Q^T y - R\beta\|_2^2 + \|\tilde{Q}^T y\|_2^2$$

Second term does not depend on β. So for least squares solution:

- Compute $X = QR$, in $2np^2 - p^3/3$ flops
- Compute $Q^T y$, in $2pn$ flops
- Solve $R\beta = Q^T y$, in p^2 flops (back substitution)

Thus in total, about $2np^2 - p^3/3$ flops (or $2np^2$ flops if $n \gg p$)

Linear systems and stability

Consider first the linear system $Ax = b$, for nonsingular $A \in \mathbb{R}^{n \times n}$. The singular value decomposition (SVD) of A:

$$A = U \Sigma V^T$$

where $U, V \in \mathbb{R}^{n \times n}$ are orthogonal, and $\Sigma \in \mathbb{R}^{n \times n}$ is diagonal with elements $\sigma_1 \geq \ldots \geq \sigma_n > 0$

Even if A is full rank, it could be "near" a singular matrix B, i.e.,

$$\mathrm{dist}(A, \mathcal{R}_k) \;=\; \min_{\mathrm{rank}(B)=k} \; \|A - B\|_{\mathrm{op}}$$

could be small, for some $k < n$. An easy SVD analysis shows that $\mathrm{dist}(A, \mathcal{R}_k) = \sigma_{k+1}$. If this is small, then solving $x = A^{-1}b$ could pose problems

Sensitivity analysis

Precise sensitivity analysis: fix some $F \in \mathbb{R}^{n\times n}$, $f \in \mathbb{R}^n$. Solve the “perturbed linear system”:

$$(A + \epsilon F)x(\epsilon) = (b + \epsilon f)$$

Theorem: The solution to the perturbed system, abbreviating $x = x(0)$, satisfies

$$\frac{\|x(\epsilon) - x\|_2^2}{\|x\|_2} \le \kappa(A)(\rho_A + \rho_b) + O(\epsilon^2)$$

where $\kappa(A) = \sigma_1/\sigma_n$ is the condition number of A, and ρ_A, ρ_b are the relative errors $\rho_A = |\epsilon|\|F\|_{\text{op}}/\|A\|_{\text{op}}$, $\rho_b = |\epsilon|\|f\|_2/\|b\|_2$

Proof:

- By implicit differentiation,

$$x'(0) = A^{-1}(f - Fx)$$

- Using a Taylor expansion around 0,

$$x(\epsilon) = x + \epsilon A^{-1}(f - Fx) + O(\epsilon^2)$$

- Rearranging gives

$$\frac{\|x(\epsilon) - x\|_2}{\|x\|_2} \leq |\epsilon| \|A^{-1}\|_{\mathrm{op}} \left(\frac{\|f\|_2}{\|x\|_2} + \|F\|_{\mathrm{op}} \right) + O(\epsilon^2)$$

  Multiplying and dividing by $\|A\|_{\mathrm{op}}$ proves the result, since $\kappa(A) = \|A\|_{\mathrm{op}} \|A^{-1}\|_{\mathrm{op}}$

Cholesky versus QR for least squares

Linear systems: worse conditioning means great sensitivity. What about for least squares problems?

$$\min_{\beta \in \mathbb{R}^p} \|y - X\beta\|_2^2$$

- Recall Cholesky solves $X^T X\beta = X^T y$. Hence we know that sensitivity scales with $\kappa(X^T X) = \kappa(X)^2$
- Meanwhile, QR operates on X, never forms $X^T X$, and can show that sensitivity scales with $\kappa(X) + \rho_{\mathrm{LS}} \cdot \kappa(X)^2$, where $\rho_{\mathrm{LS}} = \|y - X\hat{\beta}\|_2^2$

Summary: Cholesky is cheaper (and uses less memory), but QR is more stable when ρ_{LS} is small and $\kappa(X)$ is large

Indirect methods

So far we've been talking about direct methods for linear systems. These return the exact solution (in perfect computing environment)

Indirect methods (iterative methods) produce $x^{(k)}$, $k = 1, 2, 3, \ldots$ converging to a solution x. Most often used for very large, sparse systems

When to use direct versus indirect? (tim davis)

Jacobi and Gauss-Seidl

Given $A \in \mathbb{S}^n_{++}$, two basic iterative approaches for solving $Ax = b$:

- Jacobi iterations: initialize $x^{(0)} \in \mathbb{R}^n$, repeat

$$x_i^{(k)} = \Big(b_i - \sum_{j \neq i} A_{ij} x_j^{(k-1)}\Big) / A_{ii}, \; i = 1, \ldots n$$

  for $k = 1, 2, 3, \ldots$

- Gauss-Seidl iterations: initialize $x^{(0)} \in \mathbb{R}^n$, repeat

$$x_i^{(k)} = \Big(b_i - \sum_{j < i} A_{ij} x_j^{(k)} - \sum_{j > i} A_{ij} x_j^{(k-1)}\Big) / A_{ii}, \; i = 1, \ldots n$$

  for $k = 1, 2, 3, \ldots$. Gauss-Seidl uses most recent info possible

- Gauss-Seidl iterations always converge, but Jacobi iterations do not

Gradient descent

As $A \in \mathbb{S}^n_{++}$, note that the function

$$\phi(x) = \frac{1}{2} x^T A x - b^T x$$

is convex, and its minimizer satisfies $0 = \nabla \phi(x) = Ax - b$. That is, minimizing ϕ above is equivalent to solving $Ax = b$

So let's just apply good old gradient descent: initialize $x^{(0)}$, repeat:

$$x^{(k)} = x^{(k-1)} + t_k r^{(k-1)}, \quad \text{where} \quad r^{(k-1)} = b - Ax^{(k-1)}$$

for $k = 1, 2, 3, \ldots$. What step sizes to use? Abbreviate $x = x^{(k-1)}$, $r = r^{(k-1)}$, best choice is

$$t_k = \underset{t \geq 0}{\mathrm{argmin}} \; \phi(x + tr) = \frac{r^T r}{r^T A r}$$

Convergence analysis

As ϕ is strongly convex, gradient descent has linear convergence. Can make this even more precise:

Theorem: Gradient descent with exact step sizes satisfies

$$\|x^{(k)} - x\|_A \leq \sqrt{1 - \kappa(A)^{-1}} \|x^{(k-1)} - x\|_A$$

where $\|x\|_A^2 = x^T A x$ and $\kappa(A) = \lambda_1(A)/\lambda_n(A)$ is the condition number of A

Proof: direct calculation (similar to our strong convexity analysis)

Important note: the contraction factor depends adversely on $\kappa(A)$. To get $\|x^{(k)} - x\|_A \leq \epsilon \|x^{(0)} - x\|_A$, we require $O(\kappa(A) \log(1/\epsilon))$ iterations

Conjugate gradient

For large $\kappa(A)$, the contours of ϕ are elongated ellipsoids. Roughly put, gradient descent will spend a lot of time traversing back and forth "across the valley", rather than "down the valley"

Said differently, there is not enough diversity in the directions $r^{(k)}$ used by gradient descent

Conjugate gradient method: very clever idea due to Hestenes and Stiefel (1952). Replace gradient descent directions $r^{(k)}$ with

$$p^{(k)} \in \mathrm{span}\{Ap^{(1)}, \ldots Ap^{(k-1)}\}^{\perp}$$

We can see these directions are constructed to be diverse. Note: we say p, q are A-conjugate provided $p^T Aq = 0$. This explains the name

Intuition: for any p, as before

$$\underset{t \geq 0}{\operatorname{argmin}} \ \phi(x + tp) = \frac{p^T r}{p^T A p}$$

Plugging this in to $x^{(k)} = x^{(k-1)} + t_k p$ gives

$$\phi(x^{(k)}) = \phi(x^{(k-1)}) - \frac{1}{2} \frac{(p^{(k)})^T (r^{(k-1)})}{(p^{(k)})^T A p^{(k)}}$$

We see in order to make enough progress, $p^{(k)}$ must be sufficiently aligned with $r^{(k-1)}$. Recall, we also require A-conjugacy

Turns out these two considerations are simultaneously met with

$$p^{(k)} = r^{(k-1)} + \beta_k p^{(k-1)}, \quad \text{where} \quad \beta_k = -\frac{(p^{(k-1)})^T A r^{(k-1)}}{(p^{(k-1)})^T A p^{(k-1)}}$$

Convergence analysis

Theorem: Conjugate gradient method satisfies

$$\|x^{(k)} - x\|_A \le 2\left(\frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1}\right)^k \|x\|_A$$

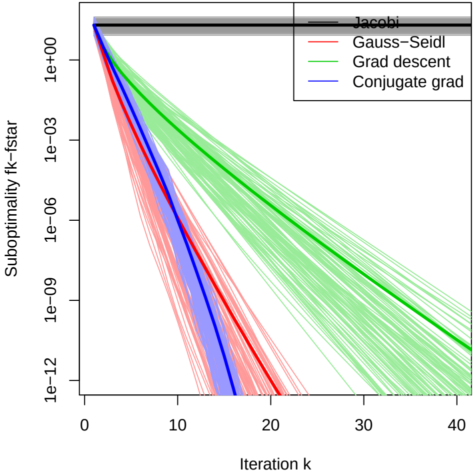
where as before $\|x\|_A^2 = x^T A x$ and $\kappa(A) = \lambda_1(A)/\lambda_n(A)$ is the condition number of A. Further, it finds the exact solution x in at most n iterations

Proof: interesting (modern proof invokes Chebyshev polynomials)

We see that conjugate gradient too enjoys linear convergence but with a contraction factor that has a better dependence on $\kappa(A)$: to get $\|x^{(k)} - x\|_A \le \epsilon \|x^{(0)} - x\|_A$, we need $O(\sqrt{\kappa(A)}\log(1/\epsilon))$ iterations

Example

Comparison of iterative methods for least squares problems: 100 random instances with $n = 100$, $p = 20$



Some advanced topics

So many more interesting things to learn ...

- Updating/downdating matrix factorizations
- Sparse matrix factorizations (SuiteSparse)
- Successive over-relaxation and acceleration
- Preconditioned conjugate gradient
- Laplacian (SDD) linear systems

References and further reading

- S. Boyd, Lecture notes for EE 264A, Stanford University, Winter 2014-2015
- S. Boyd and L. Vandenberghe (2004), "Convex optimization", Appendix C
- T. Davis (2006), "Direct methods for sparse linear systems", see `http://faculty.cse.tamu.edu/davis/suitesparse.html`
- G. Golub and C. van Loan (1996), "Matrix computations", Chapters 1–5, 10
- N. Vishnoi (2013), "$Lx = b$; Laplacian solvers and algorithmic applications"