10-725: Convex Optimization **Fall 2019**

Lecture 18: October 28

Lecturer: Ryan Tibshirani *Scribes: Aleksandr Podkopaev, Jingjing Tang, Tianwei Yue*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

18.1 Basic Operations

Complexity of certain operations can be expressed in terms of floating point operations (*flops*). A flop serves as a basic unit of computation. The operations that can be denoted by flops including:

- Addition / subtraction of floating point numbers
- Multiplication / division of floating point numbers

This measure of complexity serves rather as a rough approximation of the complexity of the problem. There are some prospects of computation cost not considered in the framework of floating point operations, *e.g.*, consider nullifying a vector in $\mathbb{R}^n$, which has computational cost, though is not counted as 0 flops (*i.e.* assigned cost is 0 flops). Finally, in practical situations, we're interested in rough, not exact flop counts to measure the complexity of the problem.

Vector-vector operations As an example, consider two vectors $a, b \in \mathbb{R}^n$. Then:

- **Addition:** $a + b$ - costs n flops for n element-wise additions.
- **Scalar multiplication:** $c \cdot a$ - costs n flops.
- **Inner product:** $a^\top b$ - costs $2n$ flops, including n multiplications and $n - 1$ additions, respectively.

Matrix-vector product Consider $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^n$. Assume that A has rows a_i (represented as column vectors). Then matrix-vector product:

$$Ab = \begin{pmatrix} a_1^\top b \\ \vdots \\ a_m^\top b \end{pmatrix}$$

is a collection of m vector-vector products. Hence, matrix-vector multiplication complexity is:

- **General matrix** A**:** $2mn$ flops.
- s**-sparse matrix** A**:** $2s$ flops. The reduction in cost is coming from the fact that $(Ab)_i = \sum_j a_{ij} b_j$, $(i, j) \in S$ where S is the index set of non-zero elements of A. The bound follows from observing that $|S| = s$ and the worst case is when non-zero elements are in the same row. Here we do not consider questions related to storage leading to efficient use of the sparsity in practice.

- **k-banded matrix $A \in \mathbb{R}^{n\times n}$:** $2nk$ flops. The bound follows since the number of flops per row is $2k$.
- **Low-rank matrix A ($A = UV^\top, U \in \mathbb{R}^{m\times r}, V \in \mathbb{R}^{n\times r}$)**: costs $2r(m+n)$. Indeed, observe that:

$$Ab = (UV^\top)b = U(V^\top b)$$

  and the cost is $2rm + 2rn = 2r(m+n)$ as a result.
- **Permutation matrix $A \in \mathbb{R}^{n\times n}$:** 0 flops. Indeed, there are no floating point operations involved.

Matrix-matrix product Consider $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times p}$ and matrix product AB. Matrix-matrix multiplication is as complex as solving a linear system. The complexity is:

- **General case:** $2mnp$ flops. Indeed, bound follows by observing that each of p columns of matrix B has has to multiplied by matrix A.
- **s-sparse A:** $2sp$ flops. Follows from the same observation. Moreover, the cost can be further reduced if B is also sparse.

Matrix-matrix vector product Consider $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times p}, c \in \mathbb{R}^p$ and the product ABc. If done improperly, i.e. if we consider product $(AB)c$, then the total cost is $2mnp + 2mp$. However, if we consider product $A(Bc)$, the the cost is $2np + 2mn$. Even though, mathematically they are the same thing, but the first one is cubic cost while the second one is quadratic cost.

18.2 Solving linear systems

For non-singular matrix $A \in \mathbb{R}^{n\times n}$ and a vector $b \in \mathbb{R}^n$ consider solving a linear system $Ax = b$. In general the cost is of order n^3 flops. The natural question is when we can solve linear systems quickly? It appears that the complexity of solving linear systems can be significantly reduced for matrices with some special properties.

- **Diagonal A:** the cost is n flops. Indeed, we just have to solve $a_i x_i = b_i, i \in \{1, \dots, n\}$ and the solution is given by (assuming that diagonal entries are non-zero for the worst case):

$$x = \begin{pmatrix} \frac{b_1}{a_1} & \cdots & \frac{b_n}{a_n} \end{pmatrix}^\top$$

- **Lower triangular matrix A ($A_{ij} = 0,\ j > i$)**: the cost is n^2 flops. Indeed, we have to solve system:

$$\begin{cases} A_{11}x_1 = b_1 \\ A_{21}x_1 + A_{22}x_2 = b_2 \\ \cdots \\ A_{n1}x_1 + \cdots + A_{n,n}x_n = b_n \end{cases} \quad \Longrightarrow \quad \begin{cases} x_1 = \frac{b_1}{A_{11}} \\ x_2 = \frac{b_2 - A_{21}x_1}{A_{22}} \\ \cdots \\ x_n = \frac{b_n - A_{n,n-1}x_{n-1} - \cdots - A_{n,1}x_1}{A_{n,n}} \end{cases}$$

  (which is known as forward substitution). The cost is: $1 + 3 + \cdots + (2n-1) = n^2$ flops.
- **Upper triangular matrix A:** the cost is also n^2. Can be done similarly, but by back substitution.
- **s-sparse matrix** A: the cost is often $\ll n^3$, but exact flop counts are not known for general sparsity structures.

- **k-banded A:** the cost is nk^2 flops (QR or Cholesky decomposition are needed to provide the bound).
- **Orthogonal matrix A:** the cost is $2n^2$. This is due to the fact that $A^{-1} = A^\top$ and we simply need to perform matrix-vector multiplication
- **Permutation matrix A:** the cost is 0. This is due to the fact that $A^{-1} = A^\top$ and $A^\top b$ costs 0 flops for permutation matrix (n assignment operations that have zero flops cost)

18.3 Matrix factorizations

System $Ax = b$ can be solved by, for example, Gaussian elimination. However, usually it is more efficient to, firstly, factorize A:

$$A = A_1 A_2 \cdots A_k$$

i.e. decompose it into product of matrices with some special structure. Usually A is decomposed into the product of 2 or 3 matrices ($k = 2$ or 3). Notice that:

- Computing the factorization is expensive, about n^3 flops
- However, applying $A_1^{-1}, \ldots, A_k^{-1}$ might be cheaper because A_i's have certain structure (orthogonal, triangular, diagonal or permutation matrices). This is especially useful when one has to solve many systems involving the same A. After initial factorization, the cost of solving t linear systems is reduced from tn^3 to tn^2.

18.3.1 QR decomposition

Any rectangular matrix $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ can be represented as:

$$A = QR$$

where $Q \in \mathbb{R}^{m \times n}$ is orthogonal ($Q^\top Q = I_m$) and $R \in \mathbb{R}^{n \times n}$ is upper triangular matrix.

- This decomposition can be computed via Gram–Schmidt procedure or via using Householder transformations. The computational cost is $2mn^2 - n^3/3$ flops.
- Number of non-zero diagonal elements of R is the rank of A.
- Columns of Q form a basis in a column-span of matrix A: span(col(A)).
- Assume that A is square, non-singular matrix, we can solve $Ax = b$ in $3n^2$ flops using:
  - Compute (in $2n^2$ flops)
    $$y = Q^\top b$$
  - Solve (in n^2 flops) by back substitution:
    $$Rx = y$$

18.3.2 Cholesky decomposition

More specialized, any $A \in \mathbb{S}^n_{++}$ (square, symmetric, positive-definite matrices) can be decomposed as:

$$A = LL^\top$$

where L is lower triangular matrix. Such decomposition can be computed in $n^3/3$ flops. After computing the decomposition, linear system $Ax = b$ can be solved as follows:

- Compute:
$$y = L^{-1}b$$
by forward substitution (n^2 flops).
- Compute:
$$x = (L^\top)^{-1}y$$
by back substitution (again n^2 flops)

As a result, solving a linear system costs $2n^2$ flops. An important example is solving a linear system with k-banded matrix A as an example. In this case, computing Cholesky decomposition takes $nk^2/4$ flops, and solving taks $2nk$ flops. It is very efficient when k is much smaller than n.

18.4 Cholesky vs QR for Least Squares

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

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

Assuming that X has full column rank, solution has form:

$$\widehat{\beta} = (X^\top X)^{-1}X^\top y$$

Natural question in this setting is how expensive it would be to obtain the solution using different methods. We, firstly, focus on *direct* methods. Indirect methods, for example, are useful in large-scale problems, when one of potential problems is, for example, that the data does not fit into the memory

Cholesky decomposition for Least Squares Equivalently, we are solving linear system:

$$(X^\top X)\beta = X^\top y$$

If X has full column rank, then $X^\top X$ is positive definite matrix. Then the complexity of obtaining the solution is:

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

Thus, in total, the complexity is about $np^2 + p^3/3$. Assuming $n \gg p$, the dominating term is np^2.

QR decomposition for Least Squares If columns of matrix $P \in \mathbb{R}^{n\times n}$ form orthonormal basis, then for any $x \in \mathbb{R}^n$:

$$\|Px\|_2^2 = x^\top P^\top P x = x^\top x = \|x\|_2^2$$

Assume that we have QR decomposition of matrix X: $X = QR$. If $Q \in \mathbb{R}^{n\times p}$, then we can complete columns to the basis in $\mathbb{R}^n$ by (Gram-Schmidt say) adding $\widetilde{Q} \in \mathbb{R}^{n\times(n-p)}$. Then consider:

$$P = \begin{pmatrix} Q & \widetilde{Q} \end{pmatrix}$$

Then:

$$\|x\|_2^2 = \left\| \begin{pmatrix} Qx \\ \widetilde{Q}x \end{pmatrix} \right\|_2^2 = \left\| Qx \right\|_2^2 + \left\| \widetilde{Q}x \right\|_2^2$$

Consider $x = y - X\beta$:

$$\|y - X\beta\|_2^2 = \|Q^\top y - Q^\top X\beta\|_2^2 + \|\widetilde{Q}^\top y - \widetilde{Q}^\top X\beta\|_2^2 =$$

$$= \|Q^\top y - R\beta\|_2^2 + \|\widetilde{Q}^\top y\|_2^2$$

Observe that the second term does not depend on β and, thus, the solution is given by solving:

$$(R^\top R)\widehat{\beta} = R^\top Q^\top y \iff R\widehat{\beta} = Q^\top y$$

As a result, Least Squares solution can be obtained via the following procedure:

- Compute $X = QR$, in $2np^2 - p^3/3$ flops.
- Compute $Q^\top y$, in $2pn$ flops
- Solve $R\beta = Q^\top y$, in $p^2 n$ flops by substitution.

Thus in total, needs $2np^2 - p^3/3 flops$. Assuming $n \gg p$, the dominating term is $2np^2$.

We observe that solving least squares using Cholesky decomposition is cheaper than using QR, but it is also less stable. Two methods have different sensitivity, which means that the quality of their solutions are different (due to different numerical errors). Cholesky and QR have different dependence on the condition number of matrix X and they actually solve different linear systems.

18.4.1 Linear systems and stability

Consider, firstly, the linear system $Ax = b$ for non-singular $A \in \mathbb{R}^n$. The *singular value decomposition* (SVD) of A is:

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

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

Rank of a matrix is unstable in a sense that even if A is full rank, it could be "near" a singular matrix B:

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

could be small, for some $k < n$. Easy SVD analysis (see Eckart-Young theorem for more details) shows that $\text{dist}(A, \mathcal{R}_k) = \sigma_{k+1}$. If this is small, then solving $x = A^{-1}b$ could pose numerical problems.

18.4.2 Sensitivity Analysis

Fix some $F \in \mathbb{R}^{n\times n}, f \in \mathbb{R}^n$. Consider solving a *perturbed linear system*:

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

Then the following result is known:

Theorem 18.1 *Denote $x(0) = x$. The solution to the perturbed system satisfies:*

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

where $\kappa(A) = \sigma_1/\sigma_n$ is the condition number of A, and ρ_A, ρ_b are the relative errors:

$$\rho_A = \frac{|\varepsilon|\|F\|_{op}}{\|A\|_{op}}, \qquad \rho_b = \frac{|\varepsilon|\|f\|_2}{\|b\|_2}$$

Proof:

1. By differentiation:
$$Fx(\varepsilon) + (A + \varepsilon F)x'(\varepsilon) = f \Longrightarrow x'(0) = A^{-1}(f - Fx)$$

2. By Taylor's expansion around 0:
$$x(\varepsilon) = x + \varepsilon A^{-1}(f - Fx) + O(\varepsilon^2)$$

3. Rearranging results in:
$$\frac{\|x(\varepsilon) - x\|_2}{\|x\|_2} \leq |\varepsilon|\|A^{-1}\|_{op}\left(\frac{\|f\|_2}{\|x\|_2} + \|F\|_{op}\right) + O(\varepsilon^2)$$

Multiplying and dividing by $\|A\|_{op}$ gives the result since $\kappa(A) = \|A\|_{op}\|A^{-1}\|_{op}$. ■

18.4.3 Implications for Least Squares problem

As we observed, for linear systems worse conditioning means great sensitivity.

- Using Cholesky decompostion, involves solving $X^\top X\beta = X^\top y$. Hence, sensitivity scales with $\kappa(X^\top X) = \kappa^2(X)$
- QR decomposition operates on X directly and never forms $X^\top X$. Hence, sensitivity scales with $\kappa(X) + \rho_{LS} \cdot \kappa^2(X)$ where $\rho_{LS} = \|y - X\beta\|_2^2$

To summarize, solving using Cholesky decomposition is cheaper (and uses less memory), but using QR decomposition is more stable when ρ_{LS} is small and $\kappa(X)$ is large.

18.5 Indirect methods

So far, we have have seen direct methods for solving system of linear systems, which give exact solutions (in perfect computing environment). However,

- Direct methods scale badly with growing dimension of the problem, *i.e.*, they are impractical for very large and sparse systems
- For many problems a direct solution might be infeasible or might not even exist.

In such cases we look for indirect (iterative) methods, to speed up with a slight cost of accuracy accordingly. Iterative methods produce a sequence of $x^{(k)}, k = 1, 2, 3, \cdots$ converging to the solution $x^\star$.

Tip (Tim Davis): when problem can fit in your computer – use direct methods.

18.5.1 Jacobi and Gauss-Seidl

Given a matrix $A \in \mathbb{S}^n_{++}$, two basic approaches for solving linear systems can be viewed as coordinate descent algorithm applied to linear systems,

$$Ax = b, \ \sum_{j \neq i} A_{ij} x_j + A_{ii} x_i = b_i$$

and thus reduce the linear system to a sequence of uni-variate problems.

- Jacobi iteration: initial $x^{(0)}$, repeat

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

  for $k = 1, 2, 3, ...$ It is easily parallelizable, but does not guarantee to converge in general.

- Gauss-Seidl iteration: initial $x^{(0)}$, repeat

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

  for $k = 1, 2, 3, ...$ Note that the difference is that Gauss-Seidl method uses the most recent available information. It is always converges, but cannot be parallel.

Note: in the case of the matrix A being strictly diagonally dominant, both Jacobi iteration and Gauss-Seidl iterations are guaranteed to converge. A square matrix A is said to be strictly diagonally dominant if the magnitude of each diagonal entry is larger than the sum of the magnitudes of all the other non-diagonal entries in the same row, *i.e.*

$$|A_{ii}| \geq \sum_{j \neq i} |A_{ij}| \quad \text{for all } i.$$

18.5.2 Gradient descent

Given $A \in \mathbb{S}^n_{++}$,

$$\phi(x) = \frac{1}{2}x^\top Ax - b^\top x$$

is convex and the minimizer satisfies: $Ax = b$. Thus, minimizing ϕ is equivalent to solving $Ax = b$. Consider gradient descent:

$$x^{(k)} = x^{(k-1)} + t_k r^{(k-1)}$$

where $r^{(k-1)} = b - Ax^{(k-1)}$ for $k = 1, 2, 3, ...$
What step sizes to use?

$$x = x^{(k-1)},\ r = r^{(k-1)}$$

$$t_k = \arg\min_{t \geq 0}\ \phi(x + tr) = \frac{r^\top r}{r^\top Ar}$$

Convergence analysis: When ϕ is strongly convex, gradient descent has linear convergence.

More precisely, we have the following results on the rate of convergence.

Theorem: Gradient descent with exact step sizes satisfies

$$\|x^{(k)} - x\|_A \leq (\sqrt{1 - \kappa(A)^{-1}})^k \|x^{(0)} - x\|_A$$

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

The theorem indicates that

- The contraction factor (of each iteration step) depends adversely on $\kappa(A)$
- It requires $O(\kappa(A))\log(1/\varepsilon))$ iterations to get $\|x^{(k)} - x\|_A \leq \varepsilon\|x^{(0)} - x\|_A$, *i.e.*, linear convergence.

To show the linear convergence, using the theorem, we actually want $\left(1 - \frac{1}{\kappa}\right)^{k/2} = \varepsilon$. Solving for k gives:

$$k = \frac{-2\log\varepsilon}{-\log(1 - \frac{1}{\kappa})} \leq \frac{2\log\frac{1}{\varepsilon}}{\frac{1}{\kappa}} = 2\kappa\log(1/\varepsilon)$$

where first-order Taylor's expansion has been applied:

$$-\log(1 - x) = x + \frac{x^2}{2} + \frac{x^3}{3} + \cdots \geq x.$$

Thus we require $O(\kappa(A)\log(1/\varepsilon))$ iterations.

18.5.3 Conjugate gradient

As mentioned before that the contraction factor of gradient descent depends adversely on $\kappa(A)$, then for large $\kappa(A)$, the convergence is rather slow. The contours of ϕ are elongated ellipsoids, and gradient descent will spend much time traversing back and forth rather than descent in the contour. That is to say, there is not enough diversity in the directions $r(k)$ taken by gradient descent.

In order to avoid directions that do not provide a objective descent, here we introduce conjugate gradient methods to construct diverse directions. At each step k, replace the gradient descent direction $r(k)$ with the A-conjugated directions of all previous descent directions, *i.e.*,

$$p^{(k)} \in \text{span}\{Ap^{(1)}, \cdots, Ap^{(k-1)}\}^{\perp}.$$

Note: we say p, q are A-conjugate provided $p^T Aq = 0$.

Descent direction: Specifically, how to compute the descent direction p? As before, we have

$$t = \arg\min_{t \geq 0} \ \phi(x + tp) = \frac{p^\top r}{p^\top Ap}.$$

For step k where $x^{(k)} = x^{(k-1)} + t_k p$, plugging this in to $\phi(x^{(k)})$ gives

$$\phi\left(x^{(k)}\right) = \phi\left(x^{(k-1)}\right) - \frac{1}{2}\frac{\left(p^{(k)}\right)^T \left(r^{(k-1)}\right)}{\left(p^{(k)}\right)^T Ap^{(k)}}.$$

That is to say, in order to make enough progress, $p^{(k)}$ must be sufficiently aligned with $r^{(k-1)}$. Recall, we also require A-conjugacy. And it turns out that these two considerations are simultaneously met with

$$p^{(k)} = r^{(k-1)} + \beta_k p^{(k-1)}, \quad \text{where} \quad \beta_k = -\frac{\left(p^{(k-1)}\right)^T Ar^{(k-1)}}{\left(p^{(k-1)}\right)^T Ap^{(k-1)}}.$$

Theorem (Convergence): Conjugate gradient method satisfies

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

where as before $\|x\|_A^2 = x^\top Ax$ 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.

Note: We see that conjugate gradient also enjoys linear convergence (the same as gradient descent) but with a contraction factor that has a better dependence on $\kappa(A)$: To get $\|x^{(k)} - x\|_A \leq \epsilon \|x^{(0)} - x\|_A$, we require $O(\sqrt{\kappa(A)})\log(1/\epsilon))$ iterations.

To see the above statement, solve $2\left(\frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1}\right)^k = \varepsilon$ for k, which gives:

$$k = \frac{-2\log\varepsilon}{-\log(1 - \frac{1}{\sqrt{\kappa(A)}+1})} \leq \frac{2\log\frac{1}{\varepsilon}}{\frac{1}{\sqrt{\kappa(A)}+1}} = 2(\sqrt{\kappa(A)} + 1)\log(1/\varepsilon) = O\left(\sqrt{\kappa(A)})\log(1/\varepsilon)\right).$$

A few final notes on conjugate descent method:

- Though still linear convergence, it can reduce the linear dependence on condition number $\kappa(A)$ to sqrt (compared to gradient descent) $\Rightarrow$ less affected by problem conditioning.
- If A is sparse, we win in computation of β_k.
- Preconditioning is a way to improve the condition number before solving a linear system.