**10-725/36-725: Convex Optimization** **Spring 2015**

# Lecture 9: Numerical Linear Algebra Primer (February 11st)

*Lecturer: Ryan Tibshirani* *Scribes: Avinash Siravuru, Guofan Wu, Maosheng Liu*



In the past few lectures, we have reviewed various first order methods used for minimizing convex optimization problems. Although first order methods only need computation of the gradient at each step, *second-order methods* would require solving linear equations that will be covered later. Several efficient non-iterative matrix algorithms exist for this purpose. We will look at some of them in this lecture and estimate their computational complexity.

## 9.1 Complexity of basic operations

Complexity can be expressed in terms of *floating point operations* or *flops* required to find the solution, expressed as a function of the problem dimension. A *Flop* serves as a basic unit of computation. It could denote an addition, subtraction, multiplication or division of two floating point numbers. In practise, division is more expensive to compute. However, we approximate it to one flop to simplify overall calculation. In this way, the number of flops indicates the cost of performing a sequence of operations. Note that, the flop count is just a rough measure of how expensive an algorithm can be. Many more aspects need to be taken into account to accurately estimate practical runtime. It is still a quite useful in predicting algorithmic cost.

As an illustration, flop counts of some basic linear algebra operations are given below:

### 9.1.1 Vector-vector operations

Given vectors $a, b \in \mathbb{R}^n$, it can be seen that:

- $a + b$ - Vector addition requires $n$ flops for $n$ elementwise additions.
- $c \cdot a \quad (c \in \mathbb{R})$ - Scalar multiplication also requires $n$ flops for as many elementwise multiplications.
- $a^T b$ - Inner Product requires approximately $2n$ flops for $n$ multiplications and $n - 1$ additions, respectively.

As stressed earlier, flops don't give the complete picture always. For example, it takes zero flops to assign an arbitrary n-dimensional vector to variable $a$ as defined above.

### 9.1.2 Matrix-vector operations

Given a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $b \in \mathbb{R}^n$, let's examine the cost of computing the product $Ab$:

- In general, it costs $2mn$ flops. $2n$ flops are required to compute the inner product between each row of $A$ and $b$. $m$ such computations yield the resultant vector $Ab$.
- For $s$-sparse A, it takes $2s$ flops. Note that, this can be obtained by replacing $mn$ in the general case with $s$. For a general matrix, $mn$ denotes the number of non-zero elements, which is equal to $s$, in the sparse case. Alternatively, recollect that flops only count multiplications and divisions. Therefore, if all the non-zero elements in $A$ could be collected and stored more efficiently, we only need to multiply them with their corresponding elements from $b$ and add them up accordingly. This will be $s$ flops each for addition and multiplication.
- For $k$-banded case (assuming $A \in \mathbb{R}^{n\times n}$), it costs $2kn$. $k$-banded matrices are a special case of sparse matrices with a defined structure. Each row contains atmost $k$ non-zero elements with the diagonal element necessarily at the centre (except at corners). It can be seen that since there are only $k$ elements in each row, $2k$ flops suffice to compute $A_i^T b$, where $A_i$ is the $i^{th}$ row of $A$. Performing this operation across $n$ rows costs $2kn$ flops.
- For $A = \sum_{i=1}^{r} u_i v_i^T \in \mathbb{R}^{m\times n}$, it requires $2r(m+n)$ flops. Observe that, $U \in \mathbb{R}^{m\times r}, V \in \mathbb{R}^{n\times r}$ and $UV^T b$ can be resolved into two matrix-vector operations $p = V^T b$ and $Up$, each requiring $2nr$ and $2mr$ flops, respectively.
- For $A \in \mathbb{R}^{n\times n}$, a permutation matirx, it takes 0 flops to reorder elements in $b$.

### 9.1.3 Matrix-matrix operations

Choose $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$ and cosider the computation of $AB$:

- In general, it takes $2mnp$ flops. Evidently, this comes from performing matrix-vector product on every column of $B$.
- For an $s$-sparse $A$, it costs $2sp$ flops. The cost can be further reduced if $B$ is also sparse.

In summary, matrix computations are very cheap for sparse and banded matrices. They are *free* when permutation matrices are used.

### 9.1.4 Matrix-matrix-vector operations

The cost of computing a matrix-matrix-vector operation can vary significantly depending on the order in which the operands are multiplied. For $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$ and $c \in \mathbb{R}^n$, the cost of computing $ABc$ can cost as high as $2mnp + 2mp$ flops, when multiplied as $(AB)c$, and as low as $2np + 2mn$ flops, when multiplied as $A(Bc)$

In the next section, we will apply the concept of flops to estimate the cost of solving linear equations.

## 9.2 Solving linear systems using Gaussian elimination

Here we consider a non-singular square matrix $A \in \mathbb{R}^{n\times n}$ and a vector $b \in \mathbb{R}^n$. Our objective is to solve the linear equation, $Ax = b$. In others words, we intend to determine the cost of computing $x = A^{-1}b$.

- In general, it costs $n^3$ flops. Note that, the flop counts grow with the increase in the operation complexity. We have seen that vector-vector computations takes $n$ flops, matrix-vector takes $n^2$ flops (for a square matrix) and solving linear systems costs $n^3$. The complexities are very different and this is pronounced when $n$ is large. However, the complexity of solving linear systems can be reduced for some matrices having special properties.
- For a diagonal matrix, it just costs $n$ flops, one each for element-wise divisions. $x = (b_1/a_1, \ldots, b_n/a_n)$
- For lower tringular matrices, which have non-zero elements only on the diagonal and below it (i.e., $A_{ij} = 0 \; \forall \; j > i$), we use forward substitution, as shown in (9.1), to compute the inverse more efficiently. It can be easily verified that this recursion costs $n^2$ flops.

$$x_i = A_{ii}^{-1}(b_i - \sum_{j=1}^{i-1} A_{ij}x_i) \quad \forall i \in n \tag{9.1}$$

- For upper triangular matrices, we apply a similar procedure called backward substitution and it also costs $n^2$ flops.
- For an $s$-sparse $A$, it often costs $\ll n^3$. However, it is hard to determine the exact order of flops. It heavily depends on the sparsity structure of the matrix. In the worst-case, it could even be of the order of $n^3$.
- For the $k$-banded A, having a structured sparsity, we can get a more accurate measure of the worst case flop count. It costs $nk^2$ flops. $k$ is generally very small and therefore they can be solved in linear time.
- For an orthogonal matrix $A$, we know that $A^{-1} = A^T$, and $x = A^T b$. This reduces to a simple matrix-vector operation costing $2n^2$ flops.
- For a permutation matrix $A$, again $A^{-1} = A^T$, and $x = A^T b$ costs 0 flops. Each row of $A$ has only one element and $x$ can be obtained from $n$ assignment operations that are free of cost!

In the next section, we see how factorizing a matrix before solving the system of linear equations, helps in reducing cost. Some well-known factorization techniques are introduced and their relative complexities are compared.

## 9.3 Numerical matrix decomposition

Although Gaussian elimination is universal for solving a linear equation, it is not efficient enough for cases when the coefficient matrix $A$ possesses special structure such as sparsity. Here we are going to introduce another two very useful alternatives, the Cholesky decomposition and QR decomposition. A detailed discussion on their properties could be found in Chapter 4 and 5 in [GL96].

### 9.3.1 Cholesky decomposition

When the matrix $A$ is symmetric and positive definite, i.e $A \in \mathbb{S}^n_{++}$, there exists a unique lower triangular matrix $L$ such that $A = LL^T$. Moreover, the matrix $L$ is non-singular. Since Cholesky decomposition is a special case of Gaussian elimination for the positive definite matrices, its computation requires $n^3/3$ flops. To solve a linear equation $Ax = b$ using Cholesky decomposition, the flop number is given by:

- Get Cholesky decomposition $A = LL^T$, $n^3/3$ flops.
- Compute $y = L^{-1}b$ by forward substitution, $n^2$ flops.
- Compute $x = (L^T)^{-1}y$ by backward substitution, $n^2$ flops.

So in general, to solve a $n$ dimensional linear equation by a given Cholesky decomposition only needs $2n^2$ flops.

### 9.3.2 QR decomposition

QR decomposition works for a more general case even when the matrix under consideration is not *square*. The construction of QR decomposition depends on a so-called *Householder transformation*[GL96]. By subsequently applying this Householder transformation, we are able to decompose a matrix $A \in \mathbb{R}^{m\times n}$ into the form as

$$A = QR$$

where $m \geq n,\ Q \in \mathbb{R}^{m\times n}, Q^TQ = I_n,\ R \in \mathbb{R}^{n\times n}$ is upper triangular.

And this transformation is called QR decomposition. Note that there's some interesting properties about the factor matrix $Q$ and $R$:

- The column vectors of $Q = [Q_1, Q_2, \cdots, Q_n]$ actually forms the *orthonormal basis of a n dimensional subspace* of $\mathbb{R}^m$. So it can be treated as orthogonal in a general sense.
- Moreover, if we expand the columns of $Q$ to the whole space as $[Q_1, Q_2, \cdots, Q_m]$, then it holds that the column span of $\tilde{Q} = [Q_{r+1}, \cdots, Q_m]$ actually forms an *orthogonal complementary of* $\text{col}(Q)$. Then by the fact that orthogonal matrix preserves vector's norm, we have

$$x^Tx = x^T \begin{bmatrix} Q & \tilde{Q} \end{bmatrix} \begin{bmatrix} Q^T \\ \tilde{Q}^T \end{bmatrix} x = x^T(QQ^T + \tilde{Q}\tilde{Q}^T)x = ||Q^Tx||_2^2 + ||\tilde{Q}^Tx||_2 \tag{9.2}$$

  which can simply the optimization problem in many cases.
- The diagonal elements of $R$ are relevant to the rank of $A$. If $\text{rank}(A) \geq r$, then the first $r$ diagonal entries of $R$ are nonzero and $\text{span}(Q_1, \cdots, Q_r) = \text{col}(A)$ where $r \leq n$.

To compute the QR decomposition of a $n \times p$ matrix $A$, it requires about $2(n - p/3) \cdot p^2$ flops. When $A$ is square, the number of flops $4n^3/3$ which is more expensive than Cholesky decomposition.

### 9.3.3 Computational cost of Cholesky and QR on least square

Here we perform an analysis on the computational cost of both sides. The case studied here is least square problem shown below as:

$$\min_{\beta\in\mathbb{R}^p} ||y - X\beta||_2^2 \Rightarrow \beta = (X^TX)^{-1}(X^Ty)$$

where $X \in \mathbb{R}^{n\times p}, y \in \mathbb{R}^n$.

To solve this linear equation given by the analytical solution, necessary flop numbers are shown in Table 9.1. This shows that Cholesky decomposition is *computationally cheaper* than QR decomposition.

| Cholesky decomposition | | QR decomposition | |
|---|---|---|---|
| Step | Flop Number | Step | Flop Number |
| Compute $z = X^T y$ | $2pn$ | Compute $X = QR$ | $2(n - p/3)p^2$ |
| Compute $A = X^T X$ | $p^2 n$ | Reduce to minimizing $\|\|Q^T y - R\beta\|\|_2^2$ by (9.2) | 0 |
| Compute $A = LL^T$ | $p^3/3$ | Compute $z = Q^T y$ | $2pn$ |
| Solve $Ax = z$ | $2p^2$ | Solve $R\beta = z$ forward subs | $p^2$ |
| Total Number | $\simeq (n + p/3)p^2$ | Total Number | $\simeq 2(n - p/3)p^2$ |

Table 9.1: Computational cost between Cholesky and QR decomposition

## 9.4 Linear systems and sensitivity

From the previous section, it seems that Cholesky decomposition is "always" better than QR decomposition computationally. However, as we take the numerical *robustness,* the performance of QR will win over by sensitivity analysis. To start with, consider the linear system $Ax = b$, with nonsingular $A \in \mathbb{R}^{n\times n}$. The singular value decomposition of A is $A = U\Sigma V^T$, where $U, V \in \mathbb{R}^{n\times n}$ are orthogonal, and $\Sigma \in \mathbb{R}^{n\times n}$ is diagonal with elements $\sigma_1 \geq ... \geq \sigma_n > 0$.

$A$ could be near a singular matrix $B$ even if it's full rank, i.e.,

$$dist(A, \Re_k) = \min_{rank(B)=k} \|A - B\|_{op}$$

could be small for some $k < n$. We can show with SVD analysis that $dist(A, \Re_k) = \sigma_{k+1}$. If the value is small, solving $x = A^{-1}b$ could be problematic.

Applying SVD we can see that:

$$x = A^{-1}b = V\Sigma^{-1}U^T b = \sum_{i=1}^{n} \frac{v_i u_i^T b}{\sigma_i}$$

If $\sigma_i > 0$ is small, close to set of rank $i - 1$ matrices, that would pose some problem.

In precise sensitivity analysis: fix some $F \in \mathbb{R}^{n\times n}$, $f \in \mathbb{R}^n$, solve:

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

**Theorem 9.1** *The solution to the perturbed system satisfies:*

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

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

**Proof:** Differentiating the equation above, let $\epsilon = 0$, and solving for $\frac{dx}{d\epsilon}$. We have:

$$\frac{dx}{d\epsilon}(0) = A^{-1}(f - Fx)$$

where $x = x(0)$.

Apply Taylor expansion around 0,

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

Rearrange and we arrive at the inequality,

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

Multiplying and dividing by $\|A\|_{op}$, and note that $\kappa(A) = \|A\|_{op}\|A^{-1}\|_{op}$, which proves the result. ■

## 9.5 Cholesky versus QR for least squares

In linear systems worse conditioning means great sensitivity.

For least squares problems: $\min_{\beta\in\mathbb{R}^p} \|y - X\beta\|_2^2$, Cholesky solves $X^TX\beta = X^Ty$, hence the sensitivity scales with $\kappa(X^TX) = \kappa(X)^2$. While QR operates on X without forming $X^TX$, that sensitivity scales with $\kappa(X) + \rho_{LS} \cdot \kappa(X)^2$, where $\rho_{LS} = \|y - X\hat{\beta}\|_2^2$.

In summary, Cholesky is *cheaper* and use less memory, while QR is *more stable* when $\rho_{LS}$ is small and $\kappa(X)$ is large.

## 9.6 Some advanced topics

- **Updating matrix factorizations**: can often be done efficiently after a simple change. E.g., QR of $A \in \mathbb{R}^{m\times n}$ can be updated in $O(m^2)$ flops after adding or deleting a row, and $O(mn)$ flops after adding or deleting a column.
- **Underdetermined least squares**: if $X \in (R)^{n\times p}$ and $rank(X) < p$, the criterion $\|y - X\beta\|_2^2$ has infinitely many minimizers. One with smallest $l_2$ norm can be computed using QR.
- **Matrix factorizations**: if $X \in (S)_{++}^n$ is $k$-banded, then we can compute its Choleksy decomposition in $nk^2/4$ flops, and apply it in $2nk$ flops.
- **Sparse matrix factorizations**: although it is lacking in theoretical analysis, in practice the performance is extremely good. To find more details about the state-of-art, we refer to [TD06], [NV13] in the references.

## 9.7 Alternative indirect methods

So far, we have have seen direct methods for solving system of linear equations, which give exact solutions. However, they may be impractical for very large and sparse systems. In such cases we trade accuracy with some speed-up. *Recursive (indirect) methods*, introduced in this section, are more useful in such situations. These methods recursively produce $x^{(k)}, k = 1, 2, 3, \ldots$, coverging to the solution $x^*$.

**Jacobi iterations:** It is the most basic approach. Let $A \in \mathbb{S}^n_{++}$, initialize $x_0 \in \mathbb{R}^n$, and repeat for $k = 1, 2, 3, \ldots$

$$x^{k+1} = (b_i - \sum_{i \neq j} A_{ij} x_j^k)/A_{ii} \tag{9.3}$$

The matrix $A$ is divided into two matrices, one containing only diagonal elements and the other containing all the non-diagonal elements. A contraction map is constructed such that the optimal solution is the fixed point of (9.3).

**Gauss-Seidel iterations:** It is an improvement to the above method and uses only the most recent iterates, i.e., $\sum_{j>i} A_{ij} x_j^{k+1} + \sum_{j<i} A_{ij} x_j^k$, instead of the sum in (9.3). Gauss-Seidel iterations converge always while Jacobi iteration don't.

**Gradient Descent:** Let $f = x^T Ax - b^T x$. Now, repeat:

$$r^{(k)} = b = Ax^{(k)} \tag{9.4}$$

$$x^{(k+1)} = x^{(k)} + t_{exact}.r^{(k)} \tag{9.5}$$

Since $A \in \mathbb{S}^n_{++}$, the criterion $f$ is strongly convex, therefore linear convergence. But this contraction depends adversely on $\kappa(A)$. This implies that the gradient direction $r^{(k)}$ are not diverse enough across iterations.

**Conjugate gradient:** It is more improved algorithm which replaces the gradient directions above with better selections that satisfy, $p^{(k)} \in span\{Ap^{(1)}, \ldots, Ap^{(k-1)}\}^{\perp}$. Note these directions are constructed to be diverse. Conjugate gradient method still uses one $A$ multiplication per iteration, and in principle, it takes $n$ iterations or much less. In practice, this is not true (numerical errors), and *preconditioning* is used.

# References

[TD06] T. Davis , "Direct methods for sparse linear systems", free C++ package at *http://faculty.cse.tamu.edu/davis/suitesparse.html*.

[GL96] G. Golub and C.V. Loan, " Matrix Computation ", Chapter 4-5, 10.

[NV13] N. Vishnoi, " $Lx = b$, Laplacian solver and their applications. "

[BV04] S. Boyd and L. Vandenberghe, " Convex Optimization ", Appendix C.