Convex Optimization

Stephen Boyd   Lieven Vandenberghe

Revised slides by Stephen Boyd, Lieven Vandenberghe, and Parth Nobel

B. Numerical linear algebra background

Outline

Flop counts and BLAS

Solving systems of linear equations

Block elimination

Flop count

- **flop** (floating-point operation): one addition, subtraction, multiplication, or division of two floating-point numbers
- to estimate complexity of an algorithm
  - express number of flops as a (polynomial) function of the problem dimensions
  - simplify by keeping only the leading terms
- not an accurate predictor of computation time on modern computers, but useful as a rough estimate of complexity

Basic linear algebra subroutines (BLAS)

vector-vector operations ($x, y \in \mathbf{R}^n$) (BLAS level 1)

- inner product $x^T y$: $2n - 1$ flops ($\approx 2n$, $O(n)$)
- sum $x + y$, scalar multiplication αx: n flops

matrix-vector product $y = Ax$ with $A \in \mathbf{R}^{m \times n}$ (BLAS level 2)

- $m(2n - 1)$ flops ($\approx 2mn$)
- $2N$ if A is sparse with N nonzero elements
- $2p(n + m)$ if A is given as $A = UV^T$, $U \in \mathbf{R}^{m \times p}$, $V \in \mathbf{R}^{n \times p}$

matrix-matrix product $C = AB$ with $A \in \mathbf{R}^{m \times n}$, $B \in \mathbf{R}^{n \times p}$ (BLAS level 3)

- $mp(2n - 1)$ flops ($\approx 2mnp$)
- less if A and/or B are sparse
- $(1/2)m(m + 1)(2n - 1) \approx m^2 n$ if $m = p$ and C symmetric

BLAS on modern computers

- there are good implementations of BLAS and variants (*e.g.*, for sparse matrices)
- CPU single thread speeds typically 1–10 Gflops/s (10^9 flops/sec)
- CPU multi threaded speeds typically 10–100 Gflops/s
- GPU speeds typically 100 Gflops/s–1 Tflops/s (10^{12} flops/sec)

Outline

Flop counts and BLAS

Solving systems of linear equations

Block elimination

Complexity of solving linear equations

- $A \in \mathbf{R}^{n \times n}$ is invertible, $b \in \mathbf{R}^n$
- solution of $Ax = b$ is $x = A^{-1}b$
- solving $Ax = b$, *i.e.*, computing $x = A^{-1}b$
  - almost never done by computing A^{-1}, then multiplying by b
  - for general methods, $O(n^3)$
  - (much) less if A is structured (banded, sparse, Toeplitz, ...)
  - *e.g.*, for A with half-bandwidth k ($A_{ij} = 0$ for $|i - j| > k$, $O(k^2 n)$
- it's super useful to recognize matrix structure that can be exploited in solving $Ax = b$

Linear equations that are easy to solve

- diagonal matrices: n flops; $x = A^{-1}b = (b_1/a_{11}, \ldots, b_n/a_{nn})$
- lower triangular: n^2 flops via **forward substitution**

$$\begin{aligned} x_1 &:= b_1/a_{11} \\ x_2 &:= (b_2 - a_{21}x_1)/a_{22} \\ x_3 &:= (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33} \\ &\vdots \\ x_n &:= (b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1})/a_{nn} \end{aligned}$$

- upper triangular: n^2 flops via **backward substitution**

Linear equations that are easy to solve

- orthogonal matrices ($A^{-1} = A^T$):
  - $2n^2$ flops to compute $x = A^T b$ for general A
  - less with structure, *e.g.*, if $A = I - 2uu^T$ with $\|u\|_2 = 1$, we can compute $x = A^T b = b - 2(u^T b)u$ in $4n$ flops

- permutation matrices: for $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ a permutation of $(1, 2, \ldots, n)$

$$a_{ij} = \begin{cases} 1 & j = \pi_i \\ 0 & \text{otherwise} \end{cases}$$

  - interpretation: $Ax = (x_{\pi_1}, \ldots, x_{\pi_n})$
  - satisfies $A^{-1} = A^T$, hence cost of solving $Ax = b$ is 0 flops
  - example:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \qquad A^{-1} = A^T = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Factor-solve method for solving $Ax = b$

- factor A as a product of simple matrices (usually 2–5):

$$A = A_1 A_2 \cdots A_k$$

- *e.g.*, A_i diagonal, upper or lower triangular, orthogonal, permutation, ...
- compute $x = A^{-1}b = A_k^{-1} \cdots A_2^{-1} A_1^{-1} b$ by solving k 'easy' systems of equations

$$A_1 x_1 = b, \qquad A_2 x_2 = x_1, \qquad \ldots \qquad A_k x = x_{k-1}$$

- cost of factorization step usually dominates cost of solve step

Solving equations with multiple righthand sides

- we wish to solve

$$Ax_1 = b_1, \qquad Ax_2 = b_2, \qquad \ldots \qquad Ax_m = b_m$$

- cost: one factorization plus m solves
- called **factorization caching**
- when factorization cost dominates solve cost, we can solve a modest number of equations at the same cost as one (!!)

LU factorization

- every nonsingular matrix A can be factored as $A = PLU$ with P a permutation, L lower triangular, U upper triangular
- factorization cost: $(2/3)n^3$ flops

Solving linear equations by LU factorization.

given a set of linear equations $Ax = b$, with A nonsingular.

1. *LU factorization.* Factor A as $A = PLU$ ($(2/3)n^3$ flops).
2. *Permutation.* Solve $Pz_1 = b$ (0 flops).
3. *Forward substitution.* Solve $Lz_2 = z_1$ (n^2 flops).
4. *Backward substitution.* Solve $Ux = z_2$ (n^2 flops).

- total cost: $(2/3)n^3 + 2n^2 \approx (2/3)n^3$ for large n

Sparse LU factorization

- for A sparse and invertible, factor as $A = P_1 L U P_2$
- adding permutation matrix P_2 offers possibility of sparser L, U
- hence, less storage and cheaper factor and solve steps
- P_1 and P_2 chosen (heuristically) to yield sparse L, U
- choice of P_1 and P_2 depends on sparsity pattern and values of A
- cost is usually much less than $(2/3)n^3$; exact value depends in a complicated way on n, number of zeros in A, sparsity pattern
- often practical to solve very large sparse systems of equations

Cholesky factorization

- every positive definite A can be factored as $A = LL^T$
- L is lower triangular with positive diagonal entries
- Cholesjy factorization cost: $(1/3)n^3$ flops

Solving linear equations by Cholesky factorization.

given a set of linear equations $Ax = b$, with $A \in \mathbf{S}^n_{++}$.

1. *Cholesky factorization.* Factor A as $A = LL^T$ ($(1/3)n^3$ flops).
2. *Forward substitution.* Solve $Lz_1 = b$ (n^2 flops).
3. *Backward substitution.* Solve $L^T x = z_1$ (n^2 flops).

- total cost: $(1/3)n^3 + 2n^2 \approx (1/3)n^3$ for large n

Sparse Cholesky factorization

- for sparse positive define A, factor as $A = PLL^TP^T$
- adding permutation matrix P offers possibility of sparser L
- same as
  - permuting rows and columns of A to get $\tilde{A} = P^TAP$
  - then finding Cholesky factorization of $\tilde{A}$
- P chosen (heuristically) to yield sparse L
- choice of P only depends on sparsity pattern of A (unlike sparse LU)
- cost is usually much less than $(1/3)n^3$; exact value depends in a complicated way on n, number of zeros in A, sparsity pattern

Example

- sparse A with upper arrow sparsity pattern

$$A = \begin{bmatrix} * & * & * & * & * \\ * & * & & & \\ * & & * & & \\ * & & & * & \\ * & & & & * \end{bmatrix} \qquad L = \begin{bmatrix} * & & & & \\ * & * & & & \\ * & * & * & & \\ * & * & * & * & \\ * & * & * & * & * \end{bmatrix}$$

L is full, with $O(n^2)$ nonzeros; solve cost is $O(n^2)$

- reverse order of entries (*i.e.*, permute) to get lower arrow sparsity pattern

$$\tilde{A} = \begin{bmatrix} * & & & & * \\ & * & & & * \\ & & * & & * \\ & & & * & * \\ * & * & * & * & * \end{bmatrix} \qquad L = \begin{bmatrix} * & & & & \\ & * & & & \\ & & * & & \\ & & & * & \\ * & * & * & * & * \end{bmatrix}$$

L is sparse with $O(n)$ nonzeros; cost of solve is $O(n)$

$\mathrm{LDL}^{\mathrm{T}}$ factorization

- every nonsingular symmetric matrix A can be factored as

$$A = PLDL^TP^T$$

  with P a permutation matrix, L lower triangular, D block diagonal with 1×1 or 2×2 diagonal blocks

- factorization cost: $(1/3)n^3$

- cost of solving linear equations with symmetric A by $\mathrm{LDL}^{\mathrm{T}}$ factorization: $(1/3)n^3 + 2n^2 \approx (1/3)n^3$ for large n

- for sparse A, can choose P to yield sparse L; cost $\ll (1/3)n^3$

Outline

Flop counts and BLAS

Solving systems of linear equations

Block elimination

Equations with structured sub-blocks

- express $Ax = b$ in blocks as

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

  with $x_1 \in \mathbf{R}^{n_1}$, $x_2 \in \mathbf{R}^{n_2}$; blocks $A_{ij} \in \mathbf{R}^{n_i \times n_j}$
- assuming A_{11} is nonsingular, can eliminate x_1 as

$$x_1 = A_{11}^{-1}(b_1 - A_{12}x_2)$$

- to compute x_2, solve

$$(A_{22} - A_{21}A_{11}^{-1}A_{12})x_2 = b_2 - A_{21}A_{11}^{-1}b_1$$

- $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$ is the **Shur complement**

Bock elimination method

Solving linear equations by block elimination.

given a nonsingular set of linear equations with A_{11} nonsingular.

1. Form $A_{11}^{-1}A_{12}$ and $A_{11}^{-1}b_1$.
2. Form $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$ and $\tilde{b} = b_2 - A_{21}A_{11}^{-1}b_1$.
3. Determine x_2 by solving $Sx_2 = \tilde{b}$.
4. Determine x_1 by solving $A_{11}x_1 = b_1 - A_{12}x_2$.

---

dominant terms in flop count

- step 1: $f + n_2 s$ (f is cost of factoring A_{11}; s is cost of solve step)
- step 2: $2n_2^2 n_1$ (cost dominated by product of A_{21} and $A_{11}^{-1}A_{12}$)
- step 3: $(2/3)n_2^3$

total: $f + n_2 s + 2n_2^2 n_1 + (2/3)n_2^3$

Examples

- for general A_{11}, $f = (2/3)n_1^3$, $s = 2n_1^2$

  $$\#\text{flops} = (2/3)n_1^3 + 2n_1^2 n_2 + 2n_2^2 n_1 + (2/3)n_2^3 = (2/3)(n_1 + n_2)^3$$

  so, no gain over standard method
- block elimination is useful for structured A_{11} ($f \ll n_1^3$)
- for example, A_{11} diagonal ($f = 0$, $s = n_1$): #flops $\approx 2n_2^2 n_1 + (2/3)n_2^3$

Structured plus low rank matrices

- we wish to solve $(A + BC)x = b$, $A \in \mathbf{R}^{n\times n}$, $B \in \mathbf{R}^{n\times p}$, $C \in \mathbf{R}^{p\times n}$
- assume A has structure (*i.e.*, $Ax = b$ easy to solve)
- first **uneliminate** to write as block equations with new variable y

$$\begin{bmatrix} A & B \\ C & -I \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

- now apply block elimination: solve

$$(I + CA^{-1}B)y = CA^{-1}b,$$

  then solve $Ax = b - By$
- this proves the **matrix inversion lemma:** if A and $A + BC$ are nonsingular,

$$(A + BC)^{-1} = A^{-1} - A^{-1}B(I + CA^{-1}B)^{-1}CA^{-1}$$

Example: Solving diagonal plus low rank equations

- with A diagonal, $p \ll n$, $A + BC$ is called **diagonal plus low rank**
- for covariance matrices, called a **factor model**
- method 1: form $D = A + BC$, then solve $Dx = b$
  - storage n^2
  - solve cost $(2/3)n^3 + 2pn^2$ (**cubic** in n)
- method 2: solve $(I + CA^{-1}B)y = CA^{-1}b$, then compute $x = A^{-1}b - A^{-1}By$
  - storage $O(np)$
  - solve cost $2p^2n + (2/3)p^3$ (**linear** in n)