Lecture: Numerical Linear Algebra Background

http://bicmr.pku.edu.cn/~wenzw/opt-2017-fall.html

Acknowledgement: this slides is based on Prof. Lieven Vandenberghe's and Michael Grant's lecture notes

Introduction

- matrix structure and algorithm complexity
- solving linear equations with factored matrices
- LU, Cholesky, LDL^T factorization
- block elimination and the matrix inversion lemma
- sparse numerical linear algebra

Matrix structure and algorithm complexity

cost (execution time) of solving $Ax = b$ with $A \in \mathbb{R}^{n\times n}$

- for general methods, grows as n^3
- less if A is structured (banded, sparse, Toeplitz, ...)

flop counts

- 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, and simplify by keeping only the leading terms
- not an accurate predictor of computation time on modern computers
- useful as a rough estimate of complexity

vector-vector operations ($x, y \in \mathbb{R}^n$)

- inner product $x^T y$: $2n - 1$ flops (or $2n$ if n is large)
- sum $x + y$, scalar multiplication αx: n flops

matrix-vector product $y = Ax$ with $A \in \mathbb{R}^{m \times n}$

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

matrix-matrix product $C = AB$ with $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times p}$

- $mp(2n - 1)$ flops (or $2mnp$ if n large)
- 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

Basic Linear Algebra Subroutines (BLAS)

written by people who had the foresight to understand the future benefits of a standard suite of “kernel” routines for linear algebra.
created and orgnized in three *levels*:

- *Level 1*, 1973-1977: $O(n)$ vector operations: addition, scaling, dot product, norms
- *Level 2*, 1984-1986: $O(n^2)$ matrix-vector operations: matrix-vector product, trigular matrix-vector solves, rank-1 and symmetric rank-2 updates
- *Level 3*, 1987-1990: $O(n^3)$ matrix-matrix operations: matrix-matrix products, trigular matrix solves, low-rank updates

BLAS operations

Level 1	addition/scaling	$\alpha x, \quad \alpha x + y$
	dot products, norms	$x^T y, \quad \|x\|_2, \quad \|x\|_1$
Level 2	matrix/vector products	$\alpha A x + \beta y, \quad \alpha A^T x + \beta y$
	rank 1 updates	$A + \alpha x y^T, \quad A + \alpha x x^T$
	rank 2 updates	$A + \alpha x y^T + \alpha y x^T$
	triangular solves	$\alpha T^{-1} x, \quad \alpha T^{-T} x$
Level 3	matrix/matrix products	$\alpha A B + \beta C, \quad \alpha A B^T + \beta C$
		$\alpha A^T B + \beta C, \quad \alpha A^T B^T + \beta C$
	rank-k updates	$\alpha A A^T + \beta C, \quad \alpha A^T A + \beta C$
	rank-k updates	$\alpha A^T B + \alpha B^T A + \beta C$
	triangular solves	$\alpha T^{-1} C, \quad \alpha T^{-T} C$

Level 1 BLAS naming convention

BLAS routines have a Fortran-inspired naming convention:

cblas_	X	XXXX
prefix	data type	operation

data types:

s	single precision real	d	double precision real
c	single precision complex	z	double precision complex

operations:

axpy	$y \leftarrow \alpha x + y$	dot	$r \leftarrow x^T y$
nrm2	$r \leftarrow \|x\|_2 = \sqrt{x^T x}$	asum	$r \leftarrow \|x\|_1 = \sum_i \lvert x_i \rvert$

example:

cblas_ddot   double precision real dot product

BLAS naming convention: Level 2/3

cblas_	X	XX	XXX
prefix	data type	structure	operation

matrix structure:

tr	triangular	tp	packed triangular	tb	banded triangular
sy	symmetric	sp	packed symmetric	sb	banded symmetric
hy	Hermitian	hp	packed Hermitian	hn	banded Hermitian
ge	general			gb	banded general

operations:

mv	$y \leftarrow \alpha Ax + \beta y$	sv	$x \leftarrow A^{-1}x$ (triangular only)
r	$A \leftarrow A + xx^T$	r2	$A \leftarrow A + xy^T + yx^T$
mm	$C \leftarrow \alpha AB + \beta C$	r2k	$C \leftarrow \alpha AB^T + \alpha BA^T + \beta C$

example:

cblas_dtrmv	double precision real triangular matrix-vector product
cblas_dsyr2k	double precision real symmetric rank-2k update

Using BLAS efficiently

always choose a higher-level BLAS routine over multiple calls to a lower-level BLAS routine

$$A \leftarrow A + \sum_{i=1}^{k} x_i y_i^T, \quad A \in \mathbb{R}^{m \times n}, x_i \in \mathbb{R}^m, y_i \in \mathbb{R}^n$$

two choices: k seperate calls to the Level 2 routine `cblas_dger`

$$A \leftarrow A + x_1 y_1^T, \quad \ldots \quad A \leftarrow A + x_k y_k^T$$

or a single call to the Level 3 routine `cblas_dgemm`

$$A \leftarrow A + XY^T, \quad X = [x_1 \ldots x_k], \quad Y = [y_1 \ldots y_k]$$

the Level 3 choice will perform much better

Is BLAS necessary?

why use BLAS when writing your own routines is so easy?

$$A \leftarrow A + XY^T, \quad A \in \mathbb{R}^{m\times n}, x_i \in \mathbb{R}^{m\times p}, y_i \in \mathbb{R}^{n\times p}$$

$$A_{ij} \leftarrow A_{ij} + \sum_{k=1}^{p} X_{ik} Y_{jk}$$

```
void matmutadd( int m, int n, int p, double* A,
        const double* X, const double* Y ) {
  int i, j, k;
  for ( i = 0 ; i < m ; ++i )
    for ( j = 0 ; j < n ; ++j )
      for ( k = 0 ; k < p ; ++k )
        A[i + j * n] += X[i + k * p] * Y[j + k * p];
}
```

Is BLAS necessary?

- tuned/optimized BLAS will ran faster than your home-brew version — often $10\times$ or more
- BLAS is tuned by selecting block sizes that fit well with your processor, cache sizes
- ATLAS (automatically tuned linear algebra software)

  `http://math-atlas.sourceforge.net`

  uses automated code generation and testing methods to *generate* an optimized BLAS library for a specific computer

Linear Algebra PACKage (LAPACK)

LAPACK contains routines for solving linear systems and performing common matrix decompositions and factorizations

- first release: February 1992; latest version (3.0): May 2000
- supercedes predecessors EISPACK and LINPACK
- supports same data types (single/double precision, real/complex) and matrix structure types (symmetric, banded, ...) as BLAS
- uses BLAS for internal computations
- routines divided into three categories: *auxiliary* routines, *computational* routines, and *driver* routines

LAPACK computational routines

compitational routines perform single, specific tasks

- factorizations: $LU, LL^T/LL^H, LDL^T/LDL^H, QR, LQ, QRZ,$ generalized QR and RQ
- symmetric/Hermitian and nonsymmetric eigenvalue decomposition
- singular value decompositions
- generalized eigenvalue and singular value decomposition

LAPACK driver routines

driver routines call a sequence of computational routines to solve standard linear algebra problems, such as

- linear equations: $AX = B$
- linear least square: minimize$_x \|b - Ax\|_2$
- linear least-norm:

$$\begin{array}{ll} \text{minimize}_x & \|c - Ax\|_2 \\ \text{subject to} & Bx = d \end{array} \qquad \begin{array}{ll} \text{minimize}_y & \|y\|_2 \\ \text{subject to} & d = Ax + By \end{array}$$

Linear equations that are easy to solve

diagonal matrices ($a_{ij} = 0$ if $i \neq j$): n flops

$$x = A^{-1}b = (b_1/a_{11}, \dots, b_n/a_{nn})$$

lower triangular ($a_{ij} = 0$ if $j > i$): n^2 flops

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

called forward substitution

upper triangular ($a_{ij} = 0$ if $j < i$): n^2 flops via backward substitution

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:

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

where $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ is a permutation of $(1, 2, \ldots, n)$

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

example:

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

The factor-solve method for solving $Ax = b$

- factor A as a product of simple matrices (usually 2 or 3):

$$A = A_1 A_2 \cdots A_k$$

(A_i diagonal, upper or lower triangular, etc)

- compute $x = A^{-1}b = A_k^{-1} \cdots A_2^{-1} A_1^{-1} b$ by solving k 'easy' 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

equations with multiple righthand sides

$$A x_1 = b_1, \qquad A x_2 = b_2, \qquad \ldots, \qquad A x_m = b_m$$

cost: one factorization plus m solves

LU factorization

every nonsingular matrix A can be factored as

$$A = PLU$$

with P a permutation matrix, L lower triangular, U upper triangular

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).

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

sparse LU factorization

$$A = P_1 L U P_2$$

- adding permutation matrix P_2 offers possibility of sparser L, U (hence, 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

Cholesky factorization

every positive definite A can be factored as

$$A = LL^T$$

with L lower triangular

cost: $(1/3)n^3$ flops

> *Solving linear equations by Cholesky factorization.*
>
> **given** a set of linear equations $Ax = b$, with $A \in \mathbb{S}^n_{++}$.
>
> 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).

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

sparse Cholesky factorization

$$A = PLL^T P^T$$

- adding permutation matrix P offers possibility of sparser L
- 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

LDL^T factorization

every nonsingular symmetric matrix A can be factored as

$$A = PLDL^T P^T$$

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

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

- cost of solving symmetric sets of linear equations by LDL^T actorization: $(1/3)n^3 + 2n^2 \approx (1/3)n^3$ for large n
- for sparse A, can choose P to yield sparse L; cost $\ll (1/3)n^3$

Equations with structured sub-blocks

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{1}$$

- variables $x_1 \in \mathbb{R}^{n_1}$, $x_2 \in \mathbb{R}^{n_2}$; blocks $A_{ij} \in \mathbb{R}^{n_i \times n_j}$
- if A_{11} is nonsingular, can eliminate x_1: $x_1 = A_{11}^{-1}(b_1 - A_{12}x_2)$; to compute x_2, solve

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

Solving linear equations by block elimination.

given a nonsingular set of linear equations (1), 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

- general A_{11} ($f = (2/3)n_1^3$, $s = 2n_1^2$): no gain over standard method

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

- block elimination is useful for structured A_{11} ($f \ll n^3$)
  for example, diagonal ($f = 0$, $s = n_1$): #flops $\approx 2n_2^2 n_1 + (2/3)n_2^3$

Structured matrix plus low rank term

$$(A + BC)x = b$$

- $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times p}$, $C \in \mathbb{R}^{p\times n}$
- assume A has structure ($Ax = b$ easy to solve)

first write as

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

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 $A + BC$ nonsingular,

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

example: A diagonal, B, C dense

- method 1: form $D = A + BC$, then solve $Dx = b$

  cost: $(2/3)n^3 + 2pn^2$
- method 2 (via matrix inversion lemma): solve

$$(I + CA^{-1}B)y = CA^{-1}b, \tag{2}$$

  then compute $x = A^{-1}b - A^{-1}By$

  total cost is dominated by (2): $2p^2n + (2/3)p^3$ ($i.e.$, linear in n)

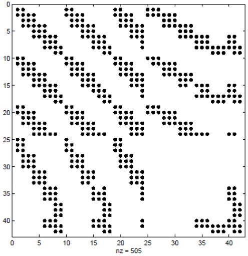
Underdetermined linear equations

if $A \in \mathbb{R}^{p \times n}$ with $p < n$, $\operatorname{rank} A = p$,

$$\{x | Ax = b\} = \{Fz + \hat{x} | z \in \mathbb{R}^{n-p}\}$$

- $\hat{x}$ is (any) particular solution
- columns of $F \in \mathbb{R}^{n \times (n-p)}$ span nullspace of A
- there exist several numerical methods for computing F (QR factorization, rectangular LU factorization, ...)

Sparse matries



- $A \in \mathbb{R}^{m \times n}$ is sparse if it has "enough zeros that it pays to take advantage ot them" (J. Wilkinson)
- usually this means n_{NZ},number of elements known to be nonzero, is small: $n_{\mathrm{NZ}} \ll mn$

Sparse matries

sparse matrices can save memory and time

- storing $A \in \mathbb{R}^{m \times n}$ using double precision numbers
  – dense: $8mn$ bytes
  – sparse: $\approx 16 n_{\mathsf{NZ}}$ bytes or less, depending on storage format
- operation $y \leftarrow y + Ax$
  – dense: mn flops
  – sparse: n_{NZ} flops
- operation $x \leftarrow T^{-1}x, T \in \mathbb{R}^{n \times n}$ triangular, nonsigular:
  – dense: $n^2/2$ flops
  – sparse: n_{NZ} flops

Representing sparse matrices

- several methods used
- simplest (but typically not used) is to store the data as list of (i, j, A_{ij}) triples
- column compressed format: an array of pairs (A_{ij}, i), and an array of pointers into this array that indicate the start os a new column
- for high end work, exotic data structures are used
- sadly, no universal standard (yet)

Sparse BLAS?

sadly there is not (yet) a standard sparse matrix BLAS library

- the "official" *sparse BLAS*

  `http://www.netlib.org/blas/blast-forum`

  `http://math.nist.gov/spblas`

- C++: Boost uBlas, Matrix Template Library, SparseLib++
- MKL from intel
- Pyhton: SciPy, PySparse, CVXOPT

Sparse factorization

library for factoring/solving systems with sparse matrices

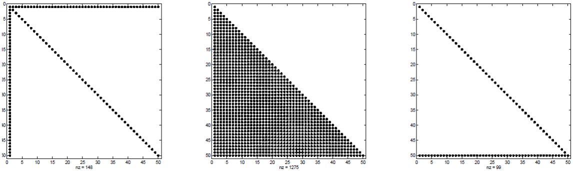
- most comprehensive: SuiteSparse (Tim Davis)

  `http://www.cise.ufl.edu.research/sparse/SuiteSparse`

- others include SuperLU, TAUCS, SPOOLES
- typically include
  -$A = PLL^T P^T$ Cholesky
  -$A = PLDL^T P^T$ for symmetric indefinite systems
  -$A = P_1 LU P_2^T$ for general (nonsymmetric) matrices
  P, P_1, P_2 are permutations or *orderings*

Sparse orderings

sparse orderings can have a *dramatic* effect on the sparsity of a factorization



- left: spy diagram of original NW arrow matrix
- center: spy diagram of Cholesky factor with no permutation $(P = I)$
- right: spy diagram of Cholesky factor with the best permutation (permute $1 \to n$)

Sparse orderings

- general problem of choosing the ordering that produces the sparest factorization is hard
- but, several simple heuristics are very effective
- more exotic ordering methods, $e.g.$, nested disection, can work very well

Symbolic factorization

- for Cholesky factorization, the ordering can be chosen based only on the sparsity pattern of A, and *not* its numerical values
- facatorization can be divided into two stages: *symbolic* factorization and *numerical* factorization
  - when solving *multiple* linear systems with identical sparsity patterns, symbolic factorization can be computed just once
  - more effort can go into selecting an ordering, since it will be amortizzed across multiple numerical factorizations
- ordering for LDL^T factorization usually has to be done on the fly, $i.e.$, based on the data

Computing dominant eigenpairs/singular pairs

Eigenvalue pairs

- ARPACK (eigs in matlab)
- LOBPCG
- Arrabit
- SLEPc

Singular value pairs

- PROPACK, a good implementation is lansvd in `http://www.math.nus.edu.sg/~mattohkc/NNLS.html`
- LMSVD with warm-starting: `https://ww2.mathworks.cn/matlabcentral/fileexchange/46875-lmsvd-m`

Other plantform of Lapack

Parallel and distributed computation

- Scalapack
  `http://www.netlib.org/scalapack/`
- Elemental
  `https://github.com/elemental/Elemental`

GPU:

- MAGMA
  `http://icl.cs.utk.edu/magma/`
- PLASMA
  `https://bitbucket.org/icl/plasma`

Other methods

we list some other areas in numerical linear algebra that have received significant attention:

- *iterative* methods for sparse and structure linear systems
- parallel and distributed methods (MPI)
- fast linear operations: fast Fouroer transforms (FFTs), convolutions, state-space linear system simulations

there is considerable existing research, and accompanying public demain (or freely licensed) code