Introduction to LAPACK

Eric J. Kostelich

ARIZONA STATE UNIVERSITY
SCHOOL OF **MATHEMATICAL & STATISTICAL SCIENCES**

Oct. 1, 2018

Overview of LAPACK and BLAS

- The Basic Linear Algebra Subroutines are a suite of computational kernels developed in the 1970s and '80s
- Objective: Provide a consistent user interface for the innermost loops of linear algebra algorithms
- The code itself may be hand-optimized assembler or written in a higher-level language
- Level-1, -2, and -3 routines correspond to $O(n)$, $O(n^2)$, and $O(n^3)$ algorithms, respectively
- All routines come in 4 versions: single- and double-precision real and single- and double-precision complex

Level-1 routines

- Example: dot product of n-vectors x and y:

  dprod = sdot(n, x, 1, y, 1)

- Naming convention: Single and Double real; single Complex; Zouble complex

- Example: To compute $\sum_{i=1}^{n} \bar{x}_i y_i$:

  innerprod = zdotc(n, x, 1, y, 1)

Level-1 routines, 2

- Givens rotations (srot family)
- SAXPY: $\mathbf{y} \leftarrow \mathbf{y} + a\mathbf{x}$

```
call saxpy(n, a, x, incx, y, incy)
```

- Dot products, Euclidean norm
- Sum of absolute values, location of maximum absolute value

The incx convention

- Suppose $\mathbf{A}$ is an $m \times n$ matrix
- To add c times the second column to the first, you can say

```
call saxpy(m, c, a(1,2), 1, a(1,1), 1)
```

- With an implicit interface, Fortran passes a reference (pointer) to the first element of each column
- Consecutive elements in each column are stride 1 apart

The incx convention

- Given $m \times n$ matrix $\mathbf{A}$, to add c times the second row to the first, you can say

```
call saxpy(n, c, a(2,1), m, a(1,1), m)
```

- Consecutive elements in each row are stride m
- There are n elements in each row (n columns)
- Memory accesses are not optimal on a cache-based architecture, but the interface is flexible

Level-2 BLAS

- Matrix-vector multiply routines for general, banded, symmetric, symmetic banded, and triangular matrices
- Example: General double-precision matrix-vector multiply $\mathbf{y} \leftarrow \alpha\mathbf{A}\mathbf{x} + \beta\mathbf{y}$:

```
real(DOUBLE), parameter:: ONE = 1.0
call dgemv('N', m, n, ONE, a, m, x, 1, ONE, y, 1)
```

- To calculate $\mathbf{y} \leftarrow \alpha\mathbf{A}^{\mathrm{T}}\mathbf{x} + \beta\mathbf{y}$, replace 'N' with 'T' and adjust dimensions and increments appropriately

The lda convention

- General calling interface for dgemv:

  call dgemv(trans, m, n, alpha, a, lda, x, incx, beta, y, incy)

- lda is the Leading Dimension of A
- If A is declared A(m,n), then lda = m

The lda convention, 2

- Alternatively, it may be more efficient to allocate space for the largest **A** that is likely to be encountered, then work with the leading $m \times n$ block for smaller problems
- Example: allocate(a(1000,1000)) and then consider an actual **A** that is 209×209:

  call dgemv('N', 209, 209, alpha, a, 1000, x, 1, beta, y, 1)

Level-3 BLAS

- Matrix-matrix multiply routines for general, symmetric, Hermitian, and triangular matrices
- Solvers for triangular systems with multiple right-hand sides
- Rank-k and -$2k$ updates
- Example: General double-precision matrix multiply $\mathbf{C} \leftarrow \alpha \mathbf{A}_{m\times n}\mathbf{B}_{n\times k} + \beta \mathbf{C}_{m\times k}$:

```
real(DOUBLE), parameter:: ONE = 1, ZERO = 0
call dgemm('N', 'N', m, n, k, ONE, a, lda, b, ldb,
ZERO, c, ldc)
```

Level-3 BLAS, 2

- You can also compute $\mathbf{A}^T\mathbf{B}$, $\mathbf{A}\mathbf{B}^T$, and $\mathbf{A}^T\mathbf{B}^T$ provided that the dimensions of the transposes are consistent
- Important: Vendors can be expected to use cache-optimized and (in Intel's case) multithreaded implementations

Fortran 90+ constructs

- These are standard and can be used on any contemporary compiler
- s=dot_product(x,y) for vector dot product: $\mathbf{x} \cdot \mathbf{y}$
- y=matmul(A,x) for matrix-vector multiplication
- C=matmul(A,B) for matrix-matrix multiplication
- C=matmul(transpose(A),B) for $\mathbf{A}^T\mathbf{B}$
- norm2(x) for 2-norm $\|\mathbf{x}\|$
- Some compilers convert matmul constructs into a call to [sdcz]gemm

Other efforts

- The ATLAS project provides "automatically tuned" BLAS
- The MAGMA project provides C interfaces for dense linear algebra on hybrid GPU architectures
- PLASMA is a partial replacement for LAPACK for multicore architectures, including Intel Phi, written in C

LAPACK overview

- Full documentation can be found in the LAPACK Users' Guide
- A printed version is available from SIAM
- LAPACK supports dense and banded matrices, but not general sparse matrices
- Three categories of routines: driver routines for solving standard problems, computational routines for specific tasks (e.g., factorization), and auxiliary routines for low-level tasks
- "Freely available" (but is copyrighted)

Numerical solution of linear systems

- Row reduce by Gaussian elimination to form $\mathbf{A} = \mathbf{LU}$
- Given $\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 \\ 3 & 5 & 6 \\ -2 & 2 & 7 \end{pmatrix}$, encode as

$$\mathbf{A}' = \begin{pmatrix} 1 & 1 & 1 \\ \boxed{-3} & 2 & 3 \\ \boxed{2} & 4 & 9 \end{pmatrix} \longrightarrow \mathbf{A}'' = \begin{pmatrix} 1 & 1 & 1 \\ \boxed{-3} & 2 & 3 \\ \boxed{2} & \boxed{-2} & 3 \end{pmatrix}$$

- Reversing the order of operations on $\mathbf{A}''$ yields $\mathbf{A}$

The encoding produces $\mathbf{L}$ and $\mathbf{A''} = \mathbf{U}$

- Given $\mathbf{A''} = \begin{pmatrix} 1 & 1 & 1 \\ \boxed{-3} & 2 & 3 \\ \boxed{2} & \boxed{-2} & 3 \end{pmatrix}$, define

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ -2 & 2 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{U} = \begin{pmatrix} 1 & 1 & 2 \\ 0 & 2 & 3 \\ 0 & 0 & 3 \end{pmatrix}$$

- Then $\mathbf{LU} = \mathbf{A}$
- We can solve $\mathbf{Ax} = \mathbf{b}$ by solving $\mathbf{Ly} = \mathbf{b}$ and $\mathbf{Ux} = \mathbf{y}$
- Once $\mathbf{L}$ and $\mathbf{U}$ are found, we can solve for arbitrarily many $\mathbf{b}$'s

LAPACK: Linear systems

- LAPACK contains routines to calculate *LU* factorizations and to do the back substitutions
- Alternatively, there are driver routines
- SGESV solves $\mathbf{AX} = \mathbf{B}$ for a GEneral matrix $\mathbf{A}$
- SGESVX solves $\mathbf{AX} = \mathbf{B}$ or $\mathbf{A}^T\mathbf{X} = \mathbf{B}$, estimates condition numbers, equilibrates $\mathbf{A}$ as necessary, and refines the solution

Example usage: SGESV

- To solve $\mathbf{AX} = \mathbf{B}$, the calling sequence is

  call sgesv(n, nrhs, a, lda, ipiv, b, ldb, info)

- $\mathbf{A}$ is $n \times n$ and is overwritten with its LU decomposition on return
- $\mathbf{B}$ is $n \times$ nrhs and is overwritten with the solution $\mathbf{X}$
- You have to supply storage for the integer n-vector ipiv, which tracks column swaps in the LU reduction
- info is 0 on successful completion and nonzero otherwise (e.g., exactly singular $\mathbf{A}$)

Linear least squares

- Suppose we wish to do a simple linear regression (straight-line fit) to the data $\{(x_i, y_i)\}_{i=1}^{n}$:

$$y_i = b_1 + b_2 x_i + \epsilon_i$$

- In matrix form, we have

$$\mathbf{y}_{n\times 1} = \mathbf{X}_{n\times 2}\mathbf{b}_{2\times 1} + \mathbf{e}_{n\times 1}$$

Linear least squares, 2

- Componentwise:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

- If we wanted to fit a quadratic model of the form $y_i = b_1 + b_2 x_i + b_3 x_i^2$, we'd add a third column to $\mathbf{X}$ consisting of the x_i^2's

The normal equations

- The normal equations for linear least squares imply

$$\mathbf{b} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

- In practice you do not want to compute $\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}$ directly!
- It can be subject to considerable roundoff error due to ill conditioning
- Instead, solve the problem using the QR factorization of $\mathbf{X}$

The *QR* factorization

- Given a linearly independent set of vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, we can generate an orthonormal set $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n\}$ with the same span using Gram-Schmidt orthogonalization
- Let $\mathbf{u}_1 = \mathbf{x}_1 / \|\mathbf{x}_1\|$ and

$$\mathbf{u}_2 = \frac{\mathbf{x}_2 - (\mathbf{x}_2 \cdot \mathbf{u}_1)\mathbf{u}_1}{\|\mathbf{x}_2 - (\mathbf{x}_2 \cdot \mathbf{u}_1)\mathbf{u}_1\|}$$

- This process may be repeated inductively

The *QR* factorization, 2

- The *QR* factorization of $\mathbf{X}$ is

$$\mathbf{X}_{n\times k} = \mathbf{Q}_{n\times k}\mathbf{R}_{n\times k}$$

where $\mathbf{Q}$ is orthonormal ($\mathbf{Q}^{\mathrm{T}}\mathbf{Q} = \mathbf{I}_{k\times k}$) and $\mathbf{R}$ is upper triangular. Then

$$\mathbf{X}^{\mathrm{T}}\mathbf{X} = (\mathbf{QR})^{\mathrm{T}}(\mathbf{QR}) = \mathbf{R}^{\mathrm{T}}\mathbf{Q}^{\mathrm{T}}\mathbf{QR} = \mathbf{R}^{\mathrm{T}}\mathbf{R}$$

and

$$\mathbf{X}^{\mathrm{T}} = (\mathbf{QR})^{\mathrm{T}} = \mathbf{R}^{\mathrm{T}}\mathbf{Q}^{\mathrm{T}}$$

The *QR* factorization, 3

- Therefore, the normal equations

$$\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)\mathbf{b} = \mathbf{X}^{\mathrm{T}}\mathbf{y}$$

  reduce to

$$\mathbf{R}^{\mathrm{T}}\mathbf{R}\mathbf{b} = \mathbf{R}^{\mathrm{T}}\mathbf{Q}^{\mathrm{T}}\mathbf{y} \longrightarrow \mathbf{R}\mathbf{b} = \mathbf{Q}^{\mathrm{T}}\mathbf{y}$$

  since $\mathbf{R}$ is nonsingular
- Let $\mathbf{z} = \mathbf{Q}^{\mathrm{T}}\mathbf{y}$ and solve the triangular system $\mathbf{R}\mathbf{b} = \mathbf{z}$
- The *QR* decomposition may be done with partial pivoting to reduce the effect of roundoff error

LAPACK driver routine for least squares

- Given $\mathbf{y}_{n\times 1} = \mathbf{X}_{n\times k}\mathbf{b}_{k\times 1}$ with $n \geq k$, we can minimize $\|\mathbf{Xb} - \mathbf{y}\|$ with

```
call sgels('N', n, k, 1, X, ldx, y, ldy, work, lwork, info)
```

- work is a work array of length lwork, which depends on the size of the problem
- y on entry is vector of observations and is overwritten with the k regression parameters on return, followed by the sum of squares of the solution vector