BLAS AND LAPACK

T. Daniel Crawford, Virginia Tech

Linear Algebra in Computational Chemistry

$$\mathbf{AV} = \mathbf{\Lambda V}$$

- Eigenvalue Problems
  - The Schrödinger equation $\hat{H}\Psi = E\Psi$
  - Configuration interaction (Hamiltonian matrix)
  - Normal mode analysis (Hessian matrix)
  - Moments of inertia (Inertia tensor)

Linear Algebra in Computational Chemistry

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

- Linear Equation Systems
  - Orbital response equations (CPHF)
  - Hessian matrix inversion
  - Geometry step (potential-energy-surface scanning)

Linear Algebra in Computational Chemistry

$$\mathbf{AB} = \mathbf{C}$$

- Matrix-Matrix Multiplication
  - Algebraic (basis-set) problems cast as matrix problems
  - Coupled cluster theory (many-body methods)
  - Self-consistent field theory (Roothaan's algorithm)

BLAS

- Basic Linear Algebra Subprograms
  - BLAS1:
    - vector norms
    - dot products
    - vector scaling
    - addition of a scalar multiple of one vector to another (AXPY):

$$\mathbf{y} = \alpha\mathbf{x} + \mathbf{y}$$

BLAS

- Basic Linear Algebra Subprograms
  - BLAS2:
    - matrix-vector operations:

$$\mathbf{y} = \alpha \mathbf{A}\mathbf{x} + \beta \mathbf{y}$$

BLAS

- Basic Linear Algebra Subprograms
  - BLAS3:
    - matrix-matrix operations:

$$\mathbf{C} = \alpha\mathbf{A}\mathbf{B} + \beta\mathbf{C}$$

    - GEMM = GEneral Matrix Multiply:

LAPACK

- Linear Algebra PACKage for solving:
  - Systems of linear equations:

$$\mathbf{Ax} = \mathbf{b}$$

  - Singular value problems;

$$\mathbf{A} = \mathbf{U\Sigma V}$$

  - Eigenvalue problems:

$$\mathbf{AV} = \mathbf{\Lambda V}$$

QR Algorithm

$$\mathbf{AV} = \mathbf{\Lambda V}$$

QR Decomposition: A square matrix may be factored into a product of an orthogonal matrix and an upper-triangular matrix:

$$\mathbf{A} = \mathbf{QR}$$

$$\mathbf{Q}^T\mathbf{Q} = \mathbf{1}$$

$$\mathbf{A}_0 \equiv \mathbf{A} = \mathbf{Q}_0\mathbf{R}_0$$

$$\mathbf{A}_1 \equiv \mathbf{R}_0\mathbf{Q}_0 = \mathbf{Q}_0^T\mathbf{Q}_0\mathbf{R}_0\mathbf{Q}_0 = \mathbf{Q}_0^T\mathbf{A}_0\mathbf{Q}_0 = \mathbf{Q}_0^{-1}\mathbf{A}\mathbf{Q}_0$$

$$\ldots$$

$$\mathbf{A}_{k+1} \equiv \mathbf{R}_k\mathbf{Q}_k = \mathbf{Q}_k^T\mathbf{Q}_k\mathbf{R}_k\mathbf{Q}_k = \mathbf{Q}_k^T\mathbf{A}_k\mathbf{Q}_k = \mathbf{Q}_k^{-1}\mathbf{A}_k\mathbf{Q}_k$$

Naming Conventions

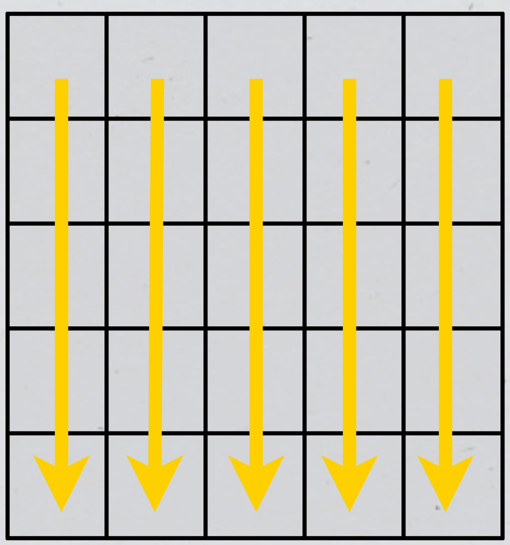
- Precision:
  - S = single precision real (float)
  - D = double-precision real (double)
  - C = single-precision complex
  - Z = double-precision complex
- Matrix type: GE = general; SY = symmetric
- Drivers: SV = solve; EV = eigenvalues; SVD = duh

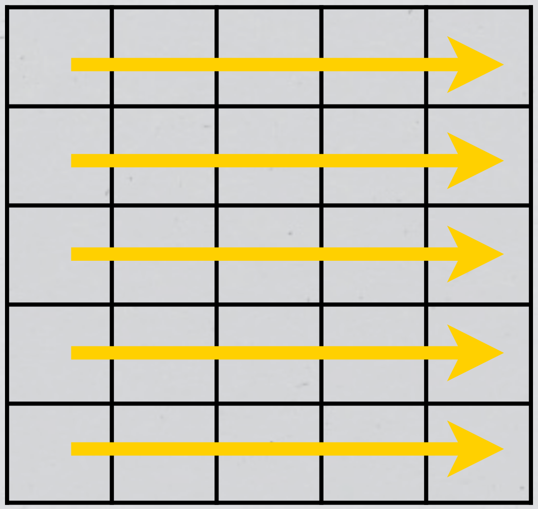
Library Interface: DGEMM

- C = alpha * A * B + beta * C
    - TRANSA = ′n′ (normal) or ′t′ (transpose)
    - TRANSA = ′n′ (normal) or ′t′ (transpose)
    - M = rows of matrix C and of matrix A or A-transpose
    - N = columns of matrix C and of matrix B or B-transpose
    - K = columns of A/A-transpose and rows of B/B-transpose
    - ALPHA = (double) scalar
    - A = (double) array (or double * in C/C++)
    - LDA = row-dim. (Fortran) or col.-dim. (C/C++) of matrix A
    - B = (double) array (or double * in C/C++)
    - LDB = row-dim. (Fortran) or col.-dim. (C/C++) of matrix B
    - BETA = (double) scalar
    - C = (double) array (or double * in C/C++)
    - LDC = row-dim. (Fortran) or col.-dim. (C/C++) of matrix C.

Fortran vs. C/C++

- The ordering of the elements in memory of a matrix is different between Fortran and C/C++:
  - In Fortran: consecutive elements follow the **columns**
  - In C/C++: consecutive elements follow the **rows**


Fortran


C/C++

Fortran vs. C/C++

- Given that the standard BLAS and LAPACK interfaces are defined to be Fortran77/90, some adjustments by C/C++ programs are necessary to call the functions correctly.

$$\mathbf{C} = \mathbf{A}\mathbf{B}$$

$$\mathbf{C}^T = (\mathbf{A}\mathbf{B})^T = \mathbf{B}^T\mathbf{A}^T$$

- Thus, when calling DGEMM() from C/C++, we reverse the ordering of the matrices, **but keep the same 'n' and 't'** arguments.

Library Interface: DSYEV

- A * V = W * V
  - JOBZ = ′n′ (eigenvalues only) or ′v′ (also eigenvectors)
  - UPLO = ′l′ (lower-triangle) or ′u′ (upper-triangle) (not important for full matrix)
  - N = dimension of A
  - A = (double) array (or double * in C/C++) containing the matrix. This is replaced by the eigenvectors on exit.
  - LDA = row-dim. (Fortran) or col.-dim. (C/C++) of matrix A
  - W = (double) array (or double * in C/C++) containing the eigenvalues in ascending order on exit.
  - WORK = (double) array (or double * in C/C++) containing memory for temporary use by the function.
  - LWORK = (int) length of WORK.
  - INFO = 0 (if successful) on exit. (Other values indicate incorrect arguments or lack of convergence).

Optimized BLAS/LAPACK

Maximum optimization of BLAS/LAPACK is vital for all computational chemistry software, and many implementations exist:

- **Netlib**: The original source of the code. Should never be used for production-level computations. (free)
- **Intel Math Kernel Library (MKL)**: Hand optimized using evil and ancient magic for Intel processors. ($$)
- **IBM Engineering and Scientific Subroutine Library (ESSL)**: Optimized for PowerPC architectures. ($$)
- **Goto BLAS**: Hand-optimized in assembler by Kazushige Goto. Astonishingly fast for Intel Nehalem and AMD Opteron. (BSD)
- **Automatically Tuned Linear Algebra Subroutines (ATLAS)**: Self-tuning at compile time for a given architecture. (BSD)