# Linear Algebra Libraries

Ramses van Zon

March 3, 2022

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

# How to deal with linear algebra in code

As much as possible, rely on existing, mature software libraries to perform linear algebra computations. By doing so you...

- Focus on your code details.
- Reduce the amount of code to write and debug
- Libraries are tuned and optimized, i.e., your code will run faster
- More options to switch methods if necessary.

1

# BLAS

# Basic Linear Algebra Subroutines

- A well-defined standard interface for linear algebra routines.
- Many highly-tuned implementations exist for various platforms.
  (Atlas, OpenBLAS, PLASMA, cuBLAS, ...)
- Interface vs. Implementation!
  Trick is designing a sufficiently general interface.
- Higher-order operations (e.g. factorizations, solving) defined in LAPACK, on top of BLAS.

# Typical BLAS routines

- Level 1:
  - `sdot` (dot product, single)
  - `zaxpy` ($ax + y$, dbl complex)
- Level 2:
  - `dgemv` (dbl matrix*vec)
  - `dsymv` (dbl symmetric matrix*vec)
- Level 3:
  - `sgemm` (general matrix-matrix)
  - `ctrmm` (triangular matrix-matrix)

Somewhat cryptic names, interfaces.

**Prefixes**

**S**: Single **C**: Complex
**D**: Double **Z**: Double Complex Matrix

**Types**

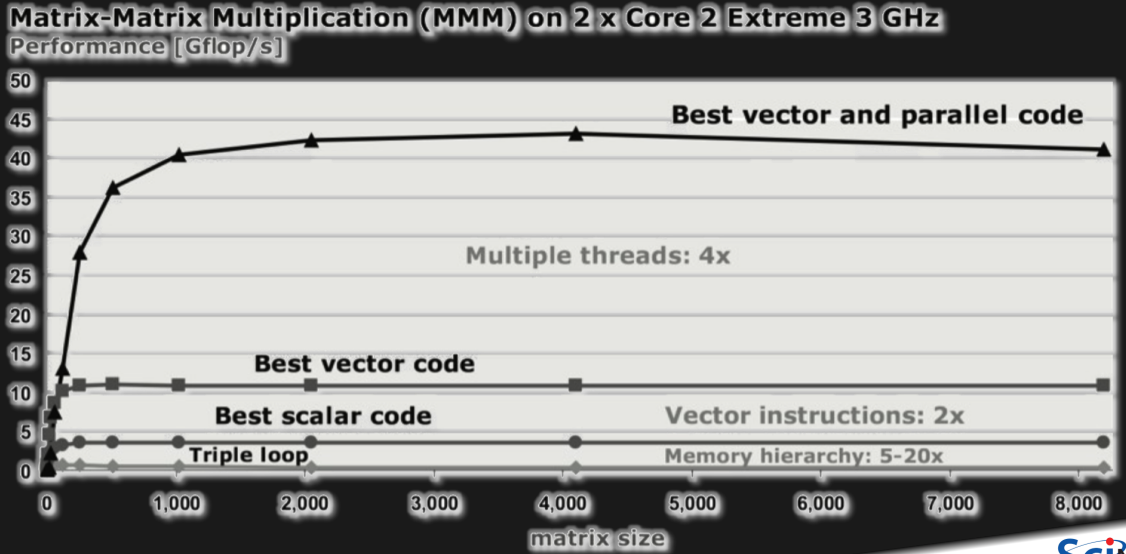
**GE**: General **GB**: General Banded
**HY**: Hermetian **HB**: Hermetian Banded
**SY**: Symmetric **SB**: Symmetric Banded
**TR**: Triangular **TB**: Triangular Banded
**TP**: Triangular Packed

# Why using Linear Algebra Packages?

- Why bother?
- Finding, downloading, installing the library
- Figuring out how to link
- C/Fortran issues
- Why not just write it?
  (It's not rocket science)

```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        for (k=0; k<N; k++)
            c[i][j] = a[i][k]*b[k][j];
```

# Never, ever, write your own...



2

# Using BLAS

# Using BLAS

- Netlib provides *reference* implementation
- Most vendors provide optimized versions
- Commercial: Intel (MKL), AMD (BLIS), IBM (ESSL)
- Open Source: ATLAS, OpenBLAS
- Fortran functions
- C interface using CBLAS and LAPACKE

# Using BLAS

## Install OpenBLAS (on Teach)

```
$ module load gcc/9
$ mkdir -p $SCRATCH/build
$ cd $SCRATCH/build
$ git clone git://github.com/xianyi/OpenBLAS.git
$ cd OpenBLAS
$ make
$ make install PREFIX=$HOME/MyOpenBLAS
```

Put the following in your $HOME/.bashrc

```
export BLAS_INC=$HOME/MyOpenBLAS/include
export BLAS_LIB=$HOME/MyOpenBLAS/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BLAS_LIB
export LIBRARY_PATH=$LIBRARY_PATH:$BLAS_LIB
export CPATH=$CPATH:$BLAS_INC
```

Log out and in again.

3

# BLAS Examples

# BLAS Example: DSCAL ( x $\leftarrow \alpha$ x)

```cpp
// dscalex.cpp
#include <iostream>
#include <cblas.h>

int main() {
   double x[] = { 1.0, 2.0, 3.0 };

   double coeff = 4.323;
   int one = 1;
   int n = 3;
   cblas_dscal(n, coeff, &x[0], one);
   for (int i = 0; i < n; i++)
      std::cout << " " << x[i];
   std::cout << "\n";
}
```

```
$ module load gcc/9
$ g++ -c -O2 -std=c++17 dscalex.cpp -o dscalex.o
$ g++ dscalex.o -o dscalex -lopenblas
$./dscalex
 4.323 8.646 12.96
```

# BLAS Example: DSCAL ( x $\leftarrow \alpha$ x)

```cpp
// dscalex.cpp
#include <iostream>
#include <cblas.h>
#include <rarray>
int main() {
   rvector<double> x(3);
   x = 1.0, 2.0, 3.0;
   double coeff = 4.323;
   int one = 1;
   int n = 3;
   cblas_dscal(n, coeff, &x[0], one);

   std::cout << x;
   std::cout << "\n";
}
```

```
$ module load gcc/9 rarray
$ g++ -c -O2 -std=c++17 dscalex.cpp -o dscalex.o
$ g++ dscalex.o -o dscalex -lopenblas
$./dscalex
{4.323,8.646,12.969}
```

# BLAS Example: DSCAL ( x $\leftarrow \alpha$ x)

## Documentation

- https://www.netlib.org/blas/blast-forum
- `man dscal`

```
NAME
       DSCAL - a vector by a constant

SYNOPSIS
       SUBROUTINE DSCAL(N,DA,DX,INCX)

           DOUBLE PRECISION          DA
           INTEGER                   INCX,N
           DOUBLE PRECISION          DX(*)

PURPOSE
         scales a vector by a constant.
         uses unrolled loops for increment equal to one.
```

Function name + order and names of arguments

Says that DA is a double precision scalar

Says that INCX and N are integers.

Says that DX is a double precision array

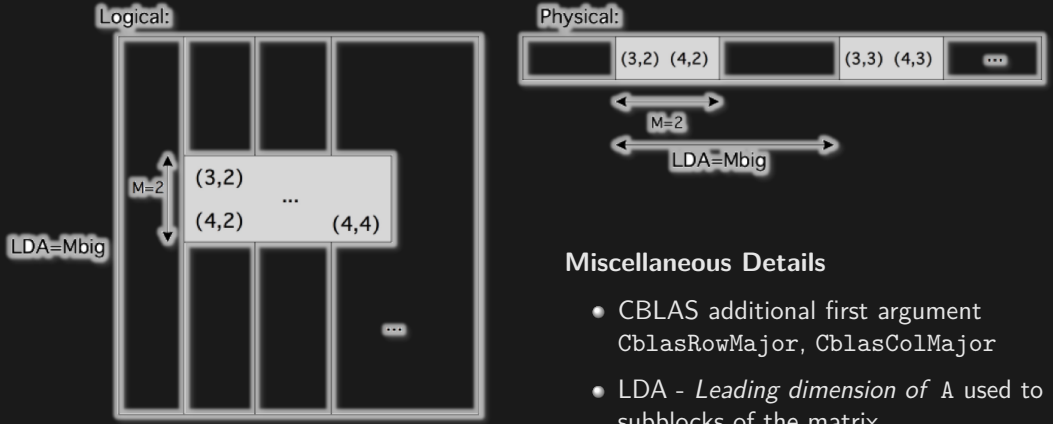
Yep, that's the Fortran version, but the C/C++ versions are (nearly) the same.

# Matrix Multiply Documentation

- man dgemm

```
NAME
       DGEMM - performs one of the matrix-matrix operations   C := alpha*op( A )*op( B ) + beta*C,
SYNOPSIS
       SUBROUTINE DGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)

           DOUBLE PRECISION     ALPHA,BETA
           INTEGER              K,LDA,LDB,LDC,M,N
           CHARACTER            TRANSA,TRANSB
           DOUBLE PRECISION     A(LDA,*),B(LDB,*),C(LDC,*)

PURPOSE
       DGEMM  performs one of the matrix-matrix operations

       where  op( X ) is one of
          op( X ) = X   or   op( X ) = X',

       alpha and beta are scalars, and A, B and C are matrices, with op( A ) an m by k matrix,
       op( B ) a  k by n matrix and C an m by n matrix.
```

# LDA? Leading Dimension?




**Miscellaneous Details**

- CBLAS additional first argument `CblasRowMajor`, `CblasColMajor`
- LDA - *Leading dimension of* `A` used to access subblocks of the matrix.

The *leading dimension* is the number of elements to skip to get to the next row (when row major) or column (when column major).

For full matrices, this is the number of columns (for row major) or the number of rows (for column major).

# BLAS Example: DGEMM ( C $\leftarrow \alpha$ A·B + $\beta$ C )

```
// dgemmex.cpp
#include <iostream>
#include <cblas.h>
#include <rarray>
int main() {
   int m = 5, k = 5, n =5;
   double alpha = 1.0, beta = 0.0;
   rmatrix<double> A(m,k);
   rmatrix<double> B(k,n);
   rmatrix<double> C(m,n);
   for (int i=0; i<(m*k); i++)  A[i/k][i%k] = (double)(i+1);
   for (int i=0; i<(k*n); i++)  B[i/n][i%n] = (double)(-i-1);
   C.fill(0.0);
   cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
               m, n, k, alpha, &A[0][0], k, &B[0][0], n, beta, &C[0][0], n);
   printmatrix("A", A);
   printmatrix("B", B);
   printmatrix("C", C);
}
void printmatrix(const char* Xname, rmatrix<double> X) {
  std::cout<<"Matrix "<<Xname<<" : "<<X.extent(0)<<" by "<<X.extent(1)<<"\n"<<X<<"\n";
}
```

# BLAS Example: DGEMM ( C $\leftarrow \alpha$ A·B + $\beta$ C )

```
// dgemmex.cpp
#include <iostream>
#include <cblas.h>
#include <rarray>
int main() {
   int m = 5, k = 5, n =5;
   double alpha = 1.0, beta = 0.0;
   rmatrix<double> A(m,k);
   rmatrix<double> B(k,n);
   rmatrix<double> C(m,n);
   for (int i=0; i<(m*k); i++)  A[i/k][i%k] = (double)(i+1);
   for (int i=0; i<(k*n); i++)  B[i/n][i%n] = (double)(-i-1);
   C.fill(0.0);
   cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
               m, n, k, alpha, A.data(), k, B.data(), n, beta, C.data(), n);
   printmatrix("A", A);
   printmatrix("B", B);
   printmatrix("C", C);
}
void printmatrix(const char* Xname, rmatrix<double> X) {
  std::cout<<"Matrix "<<Xname<<" : "<<X.extent(0)<<" by "<<X.extent(1)<<"\n"<<X<<"\n";
}
```

# BLAS Example: DGEMM ( C $\leftarrow \alpha$ A·B + $\beta$ C )

```
Matrix A : 5 by 5
{
{1,2,3,4,5},
{6,7,8,9,10},
{11,12,13,14,15},
{16,17,18,19,20},
{21,22,23,24,25}
}
Matrix B : 5 by 5
{
{-1,-2,-3,-4,-5},
{-6,-7,-8,-9,-10},
{-11,-12,-13,-14,-15},
{-16,-17,-18,-19,-20},
{-21,-22,-23,-24,-25}
}
Matrix C : 5 by 5
{
{-215,-230,-245,-260,-275},
{-490,-530,-570,-610,-650},
{-765,-830,-895,-960,-1025},
{-1040,-1130,-1220,-1310,-1400},
{-1315,-1430,-1545,-1660,-1775}
}
```

4

# LAPACK

# Linear Algebra PACKage (LAPACK)

LAPACK contains a variety of subroutines for solving linear systems, matrix decompositions, and factorizations.

- Internally uses BLAS calls
- Supports the same data types (single/double precision, real/complex and matrix structure types (symmetric, banded, etc.) as BLAS
- Three categories: auxiliary routines, computational routines, and driver routines
- C interface with prefix `LAPACKE_`
  https://www.netlib.org/lapack/lapacke.html

# Linear Algebra PACKage (LAPACK)

Computational routines are designed to perform single, specific computational tasks:

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

# LAPACK Example: DGESV (Solve A x = b)

```
NAME
       DGESV - computes the solution to a real system of linear equations  A * X = B,

SYNOPSIS
       SUBROUTINE DGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO )

           INTEGER          INFO, LDA, LDB, N, NRHS
           INTEGER          IPIV( * )
           DOUBLE PRECISION A( LDA, * ), B( LDB, * )
PURPOSE
       DGESV computes the solution to a real system of linear equations
          A  *  X = B, where A is an N-by-N matrix and X and B are N-by-NRHS matrices.
       The LU decomposition with partial pivoting and row interchanges is used to factor A as
          A = P * L * U,
       where P is a permutation matrix, L is unit lower triangular, and U is upper triangular.
       The factored form of A is then used to solve the system of equations A * X = B.

ARGUMENTS
      N        (input) INTEGER
               The number of linear equations, i.e., the order of the matrix A.  N >= 0.

      NRHS     (input) INTEGER
               The number of right hand sides, i.e., the number of columns of the matrix B.
```

# LAPACK Example: DGESV (Solve A x = b)

```cpp
// dgesvex.cpp
#include <iostream>
#include <lapacke.h>
#include <rarray>
int main() {
   const int N=3, NRHS=2, LDA=N, LDB=NRHS;
   rvector<int> ipiv(N);
   int info;
   rmatrix<double> A(N, N);
   A =  6.80, -6.05, -0.45,
       -2.11, -3.30,  2.58,
        5.66,  5.36, -2.70;
   rmatrix<double> b(N,NRHS);
   b = 4.02, -1.56,
       6.19,  4.00,
      -8.22, -8.67;
   info = LAPACKE_dgesv(LAPACK_ROW_MAJOR, N, NRHS,
                        A.data(), LDA, ipiv.data(), b.data(), NRHS);
   std::cout << "Solution x:\n" << b << "\n"
             << "Details of LU factorization\n" << A << "\n"
             << "Pivot indices\n" << ipiv << "\n";
}
```

# LAPACK Example: DGESV (Solve A x = b)

```
$ g++ -std=c++17 -O2 dgesvex.cpp -o dgesvex -lopenblas
$ ./dgesvex
```

```
Solution x:
{
{-0.0517981,-0.892398},
{-0.819976,-0.736171},
{1.30806,-0.121056}
}
Details of LU factorization
{
{6.8,-6.05,-0.45},
{0.832353,10.3957,-2.32544},
{-0.310294,-0.49802,1.28225}
}
Pivot indices
{1,3,3}
```

# What about non-dense matrices?

*E.g.:*

$$\begin{pmatrix} -2 & 1 & & & & & & \\ 1 & -2 & 4 & & & & & \\ & 4 & -2 & 4 & & & & \\ & & 4 & -2 & 4 & & & \\ & & & 4 & -2 & 4 & & \\ & & & & & \ddots & & \\ & & & & 4 & -2 & 1 & \\ & & & & & 1 & -2 & \end{pmatrix}$$

**For different types, use different functions**

- Banded: `DGBSV`
- Tri-diagonal: `DGTSV`
- Symmetric positive definite: `DPOSV`

# LAPACK Example: DGTSV (Solve Ax=b)

```
// dgtsvex.cpp
#include <iostream>
#include <lapacke.h>
#include <rarray>
int main() {
    const int N=5, NRHS=3;
    int ldb=NRHS, info;
    rvector<double> dl(N-1); dl = 1, 4, 4, 1;
    rvector<double> d(N);    d  = -2, -2 ,-2 ,-2, -2;
    rvector<double> du(N);   du = 1, 4, 4, 1;
    rmatrix<double> b(N, NRHS);
    b = 3, -1.56,  9.81,
        5,  4.00, -4.09,
        5, -8.67, -4.57,
        5,  1.75, -8.61,
        3,  2.86,  8.99;
     info = LAPACKE_dgtsv(LAPACK_ROW_MAJOR, N, NRHS,
                          dl.data(), d.data(), du.data(), b.data(), ldb);
    std::cout << "Solutions x:\n" << b << "\n";
}
```

# LAPACK Example: DGTSV (Solve Ax=b)

```
$ g++ -std=c++17 -O2 dgtsvex.cpp -o dgtsvex -lopenblas
$ ./dgtsvex
```

```
Solutions x:
{
{-0.931034,0.285747,-6.09874},
{1.13793,-0.988506,-2.38747},
{2.05172,0.43431,-0.691552},
{1.13793,-0.961839,0.899195},
{-0.931034,-1.91092,-4.0454}
}
```

# Sparse BLAS ?

Unfortunately there is not just one mature, standard sparse matrix BLAS library.

Some potential options:

- "Official'' Sparse BLAS: a reference implementation is not yet available
  https://www.netlib.org/blas/blast-forum
- NIST Sparse BLAS: An alternative BLAS system; a reference implementation is available
  https://math.nist.gov/spblas
- MKL sparse BLAS routines:

  https://www.intel.com/content/www/us/en/develop/documentation/oneapi-mkl-dpcpp-developer-reference/top/blas-and-sparse-blas-routines/sparse-blas-routines.html
- Various linear algebra packages may offer support for sparse matrices.

# References


- https://www.cs.colorado.edu/~jessup/lapack
- https://web.cs.ucdavis.edu/~bai/publications/baidemmeletal06.pdf

# Conclusions

- Linear algebra pops up everywhere
- Statistics, data fitting, graph problems, PDE/coupled ODE solves, signal processing...
- Exploit structure in your matrices
- Chose best method based on system properties (condition number, sparsity, etc..)
- Many very highly tuned packages for any sort of problem that can be cast into matrices
- LAPACK, BLAS, etc..