Week 8 — November 1, 2018

Homework

- Read section 12.8 (pp. 170-178) in "Writing Scientific Software"
- Optional: read Appendix A in "Writing Scientific Software" if you need to brush up your linear algebra knowledge

Introduction

Linear algebra is an essential tool in computational mathematics, and software for numerical linear algebra is a key component in many mathematical software packages. Matrix Laboratory, or MATLAB, is a well-known software package that makes it easy for the user to manipulate matrices and vectors, and it provides a comprehensive set of numerical linear algebra routines for various matrix decompositions and computations. MATLAB is an interpreted language, and it is great for fast prototyping, experimentation, and research. However, MATLAB is proprietary and has strict hardware requirements which means that it is not well-suited for all applications. However, implementing a set of high-quality routines for numerical linear that one can use in a stand-alone program is time-consuming and by all means nontrivial, but luckily there are a number of free libraries that a both free and of high quality.

In this week's exercises, we will work with two standard external libraries that provide a set of numerical linear algebra routines, namely the BLAS (an abbreviation of *Basic Linear Algebra Subroutines*) and LAPACK (an abbreviation of *Linear Algebra PACKage* and pronounced *L-A-PACK*). The BLAS and LAPACK libraries were developed in late 1970s in FORTRAN, a programming language that was very popular at the time (and still is today in some communities). The reference implementations, which are available at netlib.org, are still FORTRAN code, but their technical specification has been standardized, so today there exists a number optimized implementations such as MKL (Intel's *Math Kernel Library*), ACML (*AMD Core Math Library*), ATLAS (*Automatically Tuned Linear Algebra Software*), and OpenBLAS. While these libraries may not be implemented in FORTRAN, they all adhere to the BLAS specification, and the compiled libraries can be used in many different programming languages simply by linking against the compiled library.

The BLAS provides a number of routines that are specified in the BLAS Technical Forum Standard. As described in "Writing Scientific Software", the BLAS routines are divided into three categories:

- BLAS level 1 routines implement basic vector operations (scaling a vector, adding two vectors, etc.)
- BLAS level 2 routines implement basic matrix-vector operations (matrix-vector multiplication, solving triangular systems of equations, etc.)
- BLAS level 3 routines implement matrix-matrix operations (matrix-matrix multiplication, etc.).

The BLAS routines have mnemonic names. For example, the routine for adding a scalar multiple of a vector to another vector is called **daxpy** which is a mnemonic for remembering *double (precision) a x plus y*. The same operation for single precision also exists, and it has the name **saxpy** (*single (precision) a x plus y*). You can find a complete list of the routines that are part of the BLAS library on the BLAS website.

The LAPACK library provides a number of matrix factorization routines (LU, Cholesky, QR, etc.), routines for computing eigenvalues and singular values, routines for solving linear equations, and routines for solving linear least-squares problems. Many routines in the LAPACK library make use of routines from the BLAS library, and hence both libraries are necessary when using LAPACK (MKL, ACML, and OpenBLAS contain both BLAS and LAPACK routines). The LAPACK routines also have mnemonic names. For example, the **dgels** routine solves a least-squares problem (LS) with an unstructured or general (GE) coefficient matrix with input in double precision (D).

Exercises

1. Download the ZIP file `week8.zip` from DTU Inside. It includes two examples, `example_blas.c` and `example_cblas.c`, that demonstrate how to scale an array of doubles using the BLAS routine `dscal`. The ZIP file also contains a Makefile that can be used to compile the examples. The first example, `example_blas.c`, includes the `dscal` prototype explicitly in the source file:

```
/* DSCAL (scale array)                                    */
void dscal_(
    const int * n,          /* length of array     */
    const double * a,       /* scalar a            */
    double * x,             /* array x             */
    const int * incx        /* array x, stride     */
);
```

   The `dscal` routine scales n elements of an array by a scalar a (i.e., corresponding to the first two arguments). The n elements that will be scaled are `x[0]`, `x[*incx]`, `x[2*(*incx)`, ..., `x[(n-1)*(*incx)`. In other words, the first n elements of the array `x` will be scaled if `*incx` is equal to 1, every other element (starting with `x[0]`) will be scaled if `*incx` is equal to 2, and every kth element will be scaled if `*incx` is equal to k. Finally, notice that with the exception of the third argument `x`, all other arguments are specified using the `const` type qualifier. This tells the compiler that the routine is only allowed to modify whatever `x` points to.

   The second example, `example_cblas.c`, uses CBLAS which is a C interface to the BLAS library. CBLAS provides a header file, `cblas.h`, that includes a prototype for each routine in the CBLAS library. The CBLAS equivalent of the BLAS routine `dscal` is `cblas_dscal` which has the following prototype:

```
/* CBLAS_DSCAL (scale array)                              */
cblas_dscal(
```

```
    const int n,            /* length of array     */
    const double a,         /* scalar a            */
    double * x,             /* array x             */
    const int incx          /* array x, stride     */
);
```

Notice that unlike the BLAS prototype, most of the arguments in the CBLAS prototype are not pointers. If you would like to use the CBLAS library, you may include the following preprocessor directives

```
#if defined(__APPLE__) && defined(__MACH__)
#include <Accelerate/Accelerate.h>
#else
#include <cblas.h>
#endif
```

to include the correct header file in your code.

A remark for Windows users: The BLAS and LAPACK libraries are included on many Unix/Linux systems, including the DTU Unix system and macOS. Unfortunately this is not the case with Windows. To install OpenBLAS using MSYS2, open an MSYS2 shell and execute the following command:

```
$ pacman -S mingw64/mingw-w64-x86_64-openblas
```

If the installation fails, try performing an update with the following command:

```
$ pacman -Suyy
```

If you are asked to close the MSYS2 shell in order to complete the update, then (i) close the window, (ii) open an new MSYS2 shell, and (iii) repeat the update command (`pacman -Suyy`). Now retry installing OpenBLAS.

Ask the instructor or a TA for help if you encounter any problems, or do the exercises on the DTU Unix system.

2. Go to CodeJudge and complete the "Week 08" exercises.