Module 9 — November 4, 2021

Homework

- Read the note about BLAS and LAPACK.

Exercises

1. Download the ZIP file `week9.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:

```c
/* 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:

```c
/* 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. Users of WSL can install OpenBLAS by opening the WSL shell and executing the command:

```
$ sudo apt-get install libopenblas-dev
```

Users of MSYS2/MinGW can install OpenBLAS by opening the MSYS2 shell and executing the 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 "Module 09" exercises.