Math 4370/6370, Fall 2022
Due 7 October

Homework 2 – C++ Programming

Place your code for this homework into a directory entitled `hw2` within your GitHub repository. Write a `Makefile` that will compile your class, functions and `main()` routines into executables named `vec2d_test.exe` and `vec2d_b_test.exe` on a standard Linux system using the `g++` compiler. Check this in and push your changes to GitHub before midnight on the due date. Ensure that your work is well commented, fully debugged, and thoroughly tested.

I. Two-Dimensional `vec2d` class (part 1):

Study the simple one-dimensional `vec1d` class that has been provided. The object of this class is to create a "mathematical" vector object for one-dimensional data sets $v \in \mathbb{R}^n$ that can be used in algorithms, instead of always having to manually write loops for every mathematical operation on a one-dimensional array of data.

In some of your projects for this class, the ability to treat two-dimensional solution data as a mathematical "vector" may be quite useful; this is the primary goal of this homework. Create a two-dimensional vector class named `vec2d`, in files named `vec2d.hpp` and `vec2d.cpp`, that implements the same mathematical operations as `vec1d`, but on two-dimensional data sets $v \in \mathbb{R}^{m \times n}$. In `vec2d`, store the data as a two-dimensional array of doubles, e.g.

```
double **data;
```

where to allocate the data you use code similar to

```
data = new double*[m];
for (i=0; i<m; i++)
   data[i] = new double[n];
```

In your `vec2d` class, *the following routines must retain their current argument order* (modulo the conversion from `vec1d` to `vec2d`): `LinearSum`, `Scale`, `Copy`, `Constant`, `Min`, `Max`, `Dot`, `TwoNorm`, `RmsNorm` and `MaxNorm`.

The following routines must be modified as specified to accommodate the two-dimensional shape of the desired `vec2d` object:

```
double** GetData() const {return data;};
vec2d(long int m, long int n);     // constructor
vec2d Linspace(double a, double b, long int m, long int n);
vec2d Random(long int m, long int n);
```

All other functions may be retained/modified as you see fit, and you may add any new routines that you wish (since you may be using this in future codes for this class). You are not required to create any "fancy" overloaded C++ data accessors.

Create a `vec2d_test.cpp` routine similar to the one provided, so that you can guarantee the accuracy of your routines. This should use the provided `GramSchmidt2d.cpp` *without modification*. For the portion that runs timings of the Gram-Schmidt process, your code should implement (and separately time) this test using a variety of two-dimensional vector sizes $(m, n) \in$

$\{(10000, 1000), (1000, 10000), (100, 100000), (10, 1000000), (100000, 100), (1000000, 10)\}$.

Run your tests and write down the resulting timings for each (m, n) pair. *Note: these tests may take up to 1 GB of RAM; be certain to run these on a computer with sufficient memory.*

II. Two-Dimensional `vec2d` class (part 2):

Create another two-dimensional vector class, `vec2d_b`, in files named `vec2d_b.hpp` and `vec2d_b.cpp`, that again implements all of the same mathematical operations for two-dimensional data sets $v \in \mathbb{R}^{m \times n}$. However, in this class you should store the data as a one-dimensional array of doubles, e.g.

```
double *data;
```

where to allocate the data you use the much simpler code

```
data = new double [m*n];
```

When taking this approach, you must manually determine whether to store the data in row-major vs column-major order (often referred to as "C" versus "Fortran" ordering). I recommend that you use row-major ordering, i.e. the $v_{i,j}$ entry would be stored in `data[i*n+j]`, but this is not required.

Follow similar instructions as in the preceding section for the functions: `LinearSum`, `Scale`, `Copy`, `Constant`, `Min`, `Max`, `Dot`, `TwoNorm`, `RmsNorm`, `MaxNorm`, `Linspace`, `Random` and the constructor. The `GetData` function must be identical to the one in `vec1d`. All other functions may be retained/modified as you see fit. You may also add any new routines that you wish (since you may be using this in future codes for this class). You are again not required to create any overloaded C++ data accessors.

Create the file `vec2d_b_test.cpp` similar to your previous one, but that uses this second two-dimensional vector class, and links against the provided `GramSchmidt2d_b.cpp` file (*again without modification*).

Run your tests and write down the resulting timings for each (m, n) pair.

III. General Instructions and Commentary:

General instructions:

- Be certain that all of your functions are written efficiently, and that they access the stored memory in the optimal order.
- While you may add new functions to your classes, carefully follow the preceding instructions regarding function names, class names and function arguments. All codes will be tested using my own custom `main()` routines, and any deviation from the instructions will cause your program to fail compilation.

Comment on any performance differences you notice between the two different storage formats for your two-dimensional vectors.