AMATH 483 / 583 (Roche) - Homework Set 3

Due Friday April 25, 5pm PT

April 18, 2025

HW3 (80 points) Here are some coding exercises. For the performance measurements, you must have a theoretical flop count for each operation you implement built into your test codes. The compilation of test codes used for grading is:

```
g++ -std=c++17 -o xtestrefdblas -DDBLS -I. k8r_refBLAS.cpp -L. -lrefBLAS
g++ -std=c++17 -o xtestreftblas -DTBLS -I. k8r_refBLAS.cpp -L. -lrefBLAS
g++ -std=c++17 -o xtesterrors   -DTST  -I. k8r_refBLAS.cpp -L. -lrefBLAS
```

1. (+15) Level 1 BLAS (Basic Linear Algebra Subprograms). Given the following specification, write a C++ function that computes $y \leftarrow \alpha x + y$, where $x, y \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$. Write a C++ code that calls the function and measures the performance for $n = 2$ to $n = 512$. Let each n be measured *ntrial* times and plot the average performance in FLOPs for each case versus n, $ntrial \geq 3$. You may initialize your problem with any non-zero values you desire (random numbers are good). The correctness of your function will be tested against a test system with known result, so please test prior to submission. Check for and flag incorrect cases. Submit C++ files **ref_daxpy.cpp, ref_daxpy.hpp**, and performance plot.

```
void daxpy(double a, const std::vector<double> &x, std::vector<double> &y);
```

2. (+15) Level 2 BLAS. Given the following specification, write a C++ function that computes $y \leftarrow \alpha Ax + \beta y$, where $A \in \mathbb{R}^{m\times n}$,$x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, $\alpha, \beta \in \mathbb{R}$. Write a C++ code that calls the function and measures the performance for the case $m = n$, and $n = 2$ to $n = 512$. Let each n be measured *ntrial* times and plot the average performance for each case versus n, $ntrial \geq 3$. You may initialize your problem with any non-zero values you desire (random numbers are good). The correctness of your function will be tested against a general m, n test system with known result, so please test prior to submission. Check for and flag incorrect cases. Submit C++ files **ref_dgemv.cpp, ref_dgemv.hpp**, and performance plot.

```
void dgemv(double a, const std::vector<std::vector<double>> &A,
           const std::vector<double> &x, double b, std::vector<double> &y);
```

3. (+15) Level 3 BLAS. Given the following specification, write a C++ function that computes $C \leftarrow \alpha AB + \beta C$, where $A \in \mathbb{R}^{m\times p}$, $B \in \mathbb{R}^{p\times n}$, $C \in \mathbb{R}^{m\times n}$, $\alpha, \beta \in \mathbb{R}$. Write a C++ code that calls the function and measures the performance for square matrices of dimension $n = 2$ to $n = 512$. Let each n be measured *ntrial* times and plot the average performance for each case versus n, $ntrial \geq 3$. You may initialize your problem with any non-zero values you desire (random numbers are good). The correctness of your function will be tested against a general m, p, n test system with known result, so please test prior to submission. Check for and flag incorrect cases. Submit C++ files **ref_dgemm.cpp, ref_dgemm.hpp**, and performance plot.

```
void dgemm(double a, const std::vector<std::vector<double>> &A,
           const std::vector<std::vector<double>> &B, double b,
           std::vector<std::vector<double>> &C)
```

4. (+20) Template L1, L2, L3 BLAS. Given the following specifications, write C++ template functions that compute the L1, L2, L3 BLAS functions from the previous problems. Write a C++ code that calls each function. The correctness of your functions will be tested against a general test systems with known results, so please test prior to submission. Check for and flag incorrect cases. Submit C++ file(s) **ref_axpyt.cpp, ref_axpyt.hpp, ref_gemvt.cpp, ref_gemvt.hpp, ref_gemmt.cpp, ref_gemmt.hpp**.

```
template <typename T>
void axpy(T a, const std::vector<T> &x, std::vector<T> &y);

template <typename T>
void gemv(T a, const std::vector<std::vector<T>> &A,
               const std::vector<T> &x,
          T b, std::vector<T> &y);

template <typename T>
void gemm(T a, const std::vector<std::vector<T>> &A,
               const std::vector<std::vector<T>> &B,
          T b, std::vector<std::vector<T>> &C);
```

5. (+15) Shared object library. Compile the functions from problems 2-5 into a library called `librefBLAS.so`, and create a header file `refBLAS.hpp` that contains the specification for each function. (Hint - you already have a bunch of `.hpp` files -use them!) Write a C++ code that includes this header file and calls each function from the previous problems to convince yourself that it works. You will submit the compilation commands used to create the `.so` file in file `README.txt`, the C++ function file(s) needed for the compilation, and header file. I will build your shared object library using your instructions (compilation details) and run it against a test code.

0.1 Shared object library

Consider C++ source files `foo.cpp` and `bar.cpp` which contain functions that will be used by `main.cpp`. To create a shared object library that is composed of the object codes `foo.o` and `bar.o` compile as follows (assuming you are using GNU C++ compiler):

- g++ -c -fPIC foo.cpp -o foo.o
- g++ -c -fPIC bar.cpp -o bar.o
- g++ -shared -o libfoobar.so foo.o bar.o

This creates a shared object library called `libfoobar.so`. Note that `PIC` means *position independent code*. This is binary code that can be loaded and executed at any address without being modified by the linker during the compilation. For the `main.cpp` to use this library, it must be linked correctly as follows:

- g++ -o xmain main.cpp -L. -lfoobar

The `-L. -lfoobar` flags are used to link the shared library `libfoobar.so` during the compilation stage. The executable `xmain` can now be run.

0.2 FLOPs

*FL*oating point *OP*erations per *s*econd, this is a metric used to understand the performance of numerically intensive software. If a code block for instance of size n theoretically does $f(n)$ floating point operations (which we can know if we know the algorithm or problem), and it takes t seconds to complete the code block, then $FLOPs = f(n)/t$.

0.3 Timing code

Here is a code snippet that demonstrates how to use caliper based timing in C++. You may find it helpful.

```
#include <chrono>
int main()
{
    // timer foo
    auto start = std::chrono::high_resolution_clock::now();
    auto stop = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::nanoseconds>(stop - start);
    long double elapsed_time = 0.L;
    long double avg_time;
    const int ntrials = 3;

    // loop on problem size
    for (int i = 2; i <= 128; i++)
    {
        //perform an experiment
        for (int t = 0; t < ntrials; t++)
        {
            start = std::chrono::high_resolution_clock::now();
            // do work(size i, trial t)
            stop = std::chrono::high_resolution_clock::now();
            duration = std::chrono::duration_cast<std::chrono::nanoseconds>(stop - start);
            elapsed_time += (duration.count() * 1.e-9); // Convert duration to seconds
        }
        avg_time = elapsed_time / static_cast<long double>(ntrials);
        //save or report findings

        // zero time again
        elapsed_time = 0.L;
    }
```