# AMATH 483 / 583 - HW2

## Contents



# 1 Information for problems

## 1.1 Timing code segments

```
#include <chrono>
#include <iostream>

int main() {
    // Start the clock
    auto start = std::chrono::high_resolution_clock::now();

    // Code segment to time
    // ...

    // Stop the clock
    auto stop = std::chrono::high_resolution_clock::now();

    // Calculate the duration of the code segment
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(stop - start);

    // Output the duration to the console
    std::cout << "Time taken by code segment: " << duration.count() << " microseconds" <<

    return 0;
}
```

## 1.2 Loop Unroll

The loop unrolling technique aims to optimize the performance of a program by reducing the overhead of the loop itself. The idea is to execute multiple iterations of the loop at once, thereby reducing the number of instructions executed by the processor.

```
// Example loop that sums the elements in an array
int sum(int arr[], int len) {
    int result = 0;
    for (int i = 0; i < len; i++) {
        result += arr[i];
    }
```

```
    return result;
}

// Loop unrolled version that sums the elements in an array
int sum_unrolled(int arr[], int len) {
    int result = 0;
    for (int i = 0; i < len; i += 4) {
        result += arr[i];
        result += arr[i + 1];
        result += arr[i + 2];
        result += arr[i + 3];
    }
    for (; i < n; i++)
    { //don't forget about the remaining array elements
        result += arr[i];
    }
    return result;
}
```

### 1.3 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.

### 1.4 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 $FLOPs = f(n)/t$.

## 2 Problems

This assignment is due Wednesday April 19 2023 by midnight PDT.

We will explore some coding exercises in this homework. For the performance measurements, you must have a theoretical flop count for each operation you implement. **This flop count must be included in your test codes.**

1. 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 $n = 1024$. Let each $n$ be measured *ntrial* times and plot the average performance for each case versus $n$, $ntrial \geq 3$. (performance is FLOPs don't forget) 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++ function file, main source file, and performance plot.

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

2. Level 1 BLAS Loop Unrolled. Given the following specification, write a C++ function that computes $y \leftarrow \alpha x + y$, where $x, y \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$. Your function should unroll the loop at least to depth 4, and accept a block size parameter. Write a C++ code that calls the function and measures the performance for $n = 2048$ and study the block sizes $1, 2, 4, 8, 16, 32, 64$. Measure *ntrial* times for each block size 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 test system with known result, so please test prior to submission. Check for and flag incorrect cases. Submit C++ function file, main source file, and performance plot.

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

3. 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 $n = 1024$. 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++ function file, main source file, 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);
```

4. 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 $n = 1024$. 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++ function file, main source file, 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)
```

5. 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 separately, and measure and plot the performance as above for each method, except use type `float` for these experiments. You may initialize with any non-zero values you desire (random numbers are good). 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++ function file(s), main source file, and performance plots.

```
template <typename T>
void axpy(double alpha, 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);
```

6. Shared object library. Compile all your functions into a library called `librefBLAS.so`. Create a header file `refBLAS.hpp` that contains the specification of each function you created in this homework. Write a C++ code that includes this header file and calls each function from the previous problems to convince yourself that it works. Submit the details of your compilation in file `README.txt`, C++ function file(s), and header file. We will build your shared object library using your instructions (compilation details) and run it against a test code.

# 3 Extra Credit

1. (+2) Write the 6-bit representation of $+\infty$ and $-\infty$.
2. (+1) Define the vector 1-norm, 2-norm, and $\infty$-norm.
3. (+2) Given matrix $A = \begin{pmatrix} 1 & 2 \\ 0 & 2 \end{pmatrix}$, evaluate the action of $A$ on the unit balls of $\mathbb{R}^2$ defined by the 1-norm, 2-norm, and $\infty$-norm (induced matrix norms). Submit your work and drawings.