AMATH 483 / 583 (Roche) - Homework Set 6

Due Monday May 19, 5pm PT

May 12, 2025

Homework 6 (75 points)

1. (+20) **Length of a graph** of $f(x)$ on $[a,b]$ is $L = \int_a^b \sqrt{1+(f'(x))^2}dx$. Given $f(x) = ln(x) - \frac{1}{8}x^2$ on $[1,6]$. (a) Find the length of f on the interval analytically. You will need this to calculate the error in your codes. (b) Write a C++ code parallelized using the **C++ threads** library that numerically evaluates the length of this function on the interval using Riemann sum. Your code should accept the number of points used to partition the interval and the number of threads to spawn. (c) Plot the strong scaling efficiency on 1,2,4,8,16 threads of your code for $n = 1.e8$ partition points, i.e. time versus thread count. (d) Plot the *log* of the numerical error for $10, 100, ...1.e6$ partition points (increasing by factors of $10x$ each time). Submit your code and plots.

2. (+15) **Length of a graph** of $f(x)$ on $[a,b]$ is $L = \int_a^b \sqrt{1+(f'(x))^2}dx$. Given $f(x) = ln(x) - \frac{1}{8}x^2$ on $[1,6]$. (a) Write a C++ code parallelized using **MPI** that numerically evaluates the length of this function on the interval using Riemann sum. Your code should accept the number of points used to partition the interval. (b) Plot the strong scaling efficiency of your code on $1, 2, 4, 8, 16, 32$ processes for $n = 1.e8$ partition points using Hyak. (c) Plot the *log* of the numerical error for $10, 100, ...1.e6$ partition points (increasing by factors of $10x$ each time). Submit your code and plots.

3. (+30) **Elevator scheduler**. I provide `hw6-elevator.cpp` and snippet `hw6-elevator.hpp` codes. Edit the header file to implement the two functions:

   ```
   void elevator(int id); void person(int id);
   ```

   Please don't change the global variables, but feel free to add what is needed for your functions. You will submit the file `hw6-elevator.hpp`. The exercise is to write a threaded elevator scheduler for a 50-story business building with 6 elevators. Up to 7500 people may visit the building in a day. Your functions should use the C++ threads library to simulate the behavior of the elevators and visitors of the building. The driver code takes as input the number of people to service (hint: makes it a bounded buffer). After servicing all person requests, have each elevator report the number of passengers it transported, and be certain that your code exits cleanly. Please print events to *cout* in a manner that is thread safe. For each person, report their ID and current floor and destination floor (i.e. *Person 0 wants to go from floor 7 to floor 9*), report the elevator number they get on (i.e. *Person 1 entered elevator 0*), report when they arrive at their target floor (i.e. *Person 98 arrived at floor 18*). For each elevator, report the direction of movement from current floor to next floor (i.e. *Elevator 2 moving from floor 12 to floor 19*), join all the elevator threads once all visitors have arrived at their floors, and report success (i.e. *Job completed!*).

   - randomly select the current floor and destination floor for each person
   - I let each elevator start from the ground floor if it matters
   - issue a person thread in the *detach* state (i.e. *thread(person, i).detach();*)
   - each elevator will be it's own joinable thread
   - keep track of the current floor and direction of each elevator
   - keep track of the destinations of the visitors who are waiting for elevators on each floor
   - determine which elevator should respond to a given request, based on its current location, direction, and the number of passengers it is already carrying
   - the maximum occupancy of the elevators is fixed, say 10
   - move the elevators up or down to their next destination, picking up and dropping off passengers as necessary

- print events to screen as they occur
- add some sleep functions to mimic the action of the elevators moving between floors

4. (+20) **Communication Bandwidth**. (a) Use the following template declaration to write a **MPI** C++ function using point-to-point communication that implements broadcast. Submit the function in file `my_broadcast.hpp`. (b) Measure and plot the performance of your function and the `int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)` function on messages of size $8B, 16B, ..., 256MB$ bytes (doubling the message size each instance). Do this exercise using Hyak on 4 and 32 MPI processes, and make a plot of your measurements with bandwidth (bytes per second) on the y-axis, and the message size in bytes on the x-axis. Submit your plot.

   - ```
     template <typename T>
     void my_broadcast(T* data, int count, int root, MPI_Comm comm);
     ```
     ```

Homework 5 (80 points)

1. (+20) Ring. $\forall x, y \in \mathbb{Z}$ define binary operator $\oplus$ as $x \oplus y = x + y - 1$, and binary operator $\odot$ as $x \odot y = x + y - xy$. $(\mathbb{Z}, \oplus, \odot)$ is a ring.
   (a) Additive Identity. Find $z \in \mathbb{Z} \ni a \oplus z = z \oplus a = a, \forall a \in \mathbb{Z}$.
   (b) Additive Inverse. $\forall a \in \mathbb{Z}, \exists b \in \mathbb{Z} \ni a \oplus b = b \oplus a = z$. Find b.
   (c) Show that $\oplus$ commutes.
   (d) Multiplicative Identity. A ring with unity is a ring with member $u \in \mathbb{Z}, u \neq z \ni a \odot u = u \odot a = a, \forall a \in \mathbb{Z}$. Find u.

2. (+10) Recursive sequence. Let $T_0 = 2$, $T_1 = 3$, and $T_2 = 6$, and for $n \geq 3$ let $T_n = (n+4)T_{n-1} - 4nT_{n-2} + (4n-8)T_{n-3}$.
   (a) Derive a formula of the form $T_n = A_n + B_n$ where $(A_n), (B_n)$ are well-known sequences.
   (b) Use induction to prove your formula works.

3. (+20) Memory access time. Write C++ functions that perform **row and column swap** operations on a type double matrix stored in column major index order using a single vector container for the data. The specifications are provided here. Put the functions in file **matrix_swaps.hpp** (no need for header guards -just the code for your functions). Conduct a performance test for square matrix dimensions 16, 32, 64, ... 4096, measuring the time required to conduct row swaps and column swaps separately. Let each operation be measured *ntrial* times, *ntrial* ≥ 3. Make a single plot of your *row* and *column* swap timing measurements with time on the y-axis (could be $log_{10}(time)$) and the problem dimension on the x-axis. Submit files **mem_swaps.hpp** and the plot.

```
void swapRows(std::vector<double> &matrix, int nRows, int nCols, int i, int j);
void swapCols(std::vector<double> &matrix, int nRows, int nCols, int i, int j);
```

   You may test the swap capabilities on randomly selected index pairs using a function like this:

```
#include <utility>   // For std::pair
std::pair<int, int> getRandomIndices(int n)
{
    int i = std::rand() % n;
    int j = std::rand() % (n - 1);
    if (j >= i)
    {
        j++;
    }
    return std::make_pair(i, j);
}
// ... from inside main()
//std::pair<int, int> rowIndices = getRandomIndices(M);
//int i = rowIndices.first;
//int j = rowIndices.second;
//std::pair<int, int> colIndices = getRandomIndices(N);
// ...
```

4. (+10) **Linear algebra**.
   (a) (+5) **Basis**. Does the given set $S = \{1, 1-x, (1-x)^2\}$ form a basis for the space of polynomials up to degree 2, $x \in \mathbb{R}$? Show your work. (Hint: must be both linearly independent and span the space)
   (b) (+5) **Gram-Schmidt**. Given set $S = \{x_1(t), x_2(t), x_3(t)\} \ni x_1(t) = t^2, \quad x_2(t) = t, \quad x_3(t) = 1$ and $t \in [-1, 1] \in \mathbb{R}$, construct by hand an orthonormal basis for S using the inner product $(x, y) = \int_{-1}^{1} x(t)y(t)\,dt$.

5. (+20) **Strassen**. Use notes from the class lecture to implement the C++ template for the (recursive) Strassen matrix multiplication algorithm for matrices with even dimension, `C = strassenMultiply(A, B);`. Use the provided starter code `strassen.cpp` and implement in it the declared method `strassenMultiply`. Plot the average performance (FLOPs) multiplying `double` precision square matrices of even dimension from $n = 2$ to $n = 512$ as measured *ntrial* times, *ntrial* ≥ 3. Submit `strassen.cpp` and performance plot.

Homework 4 (90 points)

1. (+10) 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 ∞-norm (induced matrix norms). Submit your work and drawings.

2. (+20) Compiler Optimization of Matrix Multiplication Loop Permutations. Implement C++ templated gemm, $C \leftarrow \alpha AB + \beta C$ ($A \in \mathbb{T}^{m \times p}$, $B \in \mathbb{T}^{p \times n}$, $C \in \mathbb{T}^{m \times n}$, $\alpha, \beta \in \mathbb{T}$) for $\{kij\}$ and $\{jki\}$ loop permutations using the specifications provided here:

   - ```
     template<typename T>
     void mm_jki(T a, const std::vector<T>& A, const std::vector<T>& B, T b,
     std::vector<T>& C, int m, int p, int n);
     ```

   - ```
     template<typename T>
     void mm_kij(T a, const std::vector<T>& A, const std::vector<T>& B, T b,
     std::vector<T>& C, int m, int p, int n);
     ```

   You will explore matrix multiply performance applying compiler optimization levels *-O0* and *-O3* for square matrices of dimension $n = 2$ to $n = 512$, stride one to these functions. For reference, recall the loop order for $C_{i,j} \leftarrow \sum_{k=0}^{p-1} A_{i,k} * B_{k,j} \, \forall [i = 0, ..., m-1][j = 0, ..., n-1]$ is $\{ijk\}$, i outer loop, j middle loop, k inner loop. Let each n be measured *ntrial* times and plot the average performance in FLOPs (flop count / time(seconds)) for each case versus n, *ntrial* ≥ 3. Submit plots for both permutation variants that include both FP32 (*float*) and FP64 (*double*) compiler optimization results.

3. (+20) **Row major Matrix class**. Reference the file `matrix_class.hpp` for the starter code for a Matrix class template for row major index referencing and `std::vector<T>` for the matrix storage scheme. Please put your function implementations in file `matrix_class.hpp`, and submit `matrix_class.hpp`.

   - (+5) Matrix transpose for $A \in \mathbb{R}^{m \times n}$ is defined $A^T_{i,j} = A_{j,i}$ and so $A^T \in \mathbb{R}^{n \times m}$. This method returns a matrix as defined by the class.

     – `Matrix<T> transpose() const{}`

   - (+5) Matrix infinity norm for $A \in \mathbb{R}^{m \times n}$ is defined $\|A\|_\infty = \max_{0 \leq i \leq m} \sum_{j=1}^{j=n} |A_{i,j}|$. This method returns a number.

     – `T infinityNorm() const{}`

   - (+5) Write the method to operator overload multiplication $*$ for the matrix class I provided. This method returns a matrix as defined by the class.

     – `Matrix<T> operator*(const Matrix<T> &other) const{}`

   - (+5) Write the method to operator overload addition + for the matrix class I provided. This method returns a matrix as defined by the class.

     – ```
       template <typename T>
       Matrix<T> Matrix<T>::operator+(const Matrix<T> &other) const{}
       ```

4. (+10) **Extremum**. Consider the surface defined by $xy + 2xz = 5\sqrt{5}$ and $(x, y, z) \in \mathbb{R}^3$. Find (a) the coordinate instance(s) affiliated with the minimum distance from a point on the surface to the origin, and (b) the value of the minimum distance. You will find (may safely assume) the domain $[-3, 3] \times [-3, 3] \times [-3, 3] \in \mathbb{R}^3$ holds the correct coordinate instance(s).

5. (+10) **IO bandwidth**. Write a C++ function that `write`s type `double` square matrices in column major order to file in binary. Measure the time required to complete the `write` for matrices of dimension 32, 64, 128, ... 16384 (2GB). Write a C++ function that `read`s binary matrices from file to type `double` matrices in memory. Measure the time required to complete each `read` for the same dimensions. (a) Make a single plot of the *read* and *write* measurements with the bandwidth (bytes per second) on the y-axis, and the problem dimension on the x-axis. Submit your plot.
       ```

6. (+20) **File access time**. (a) Write C++ functions for the given function declarations that perform *row* and *column* swap operations on a type `double` matrix stored in a file in column major index order. Test the swapping capabilities for correctness. Put the functions you write in file **file_swaps.hpp**. (b) Conduct a performance test for square matrix dimensions 16, 32, 64, 128, ... 8192, measuring the time required to conduct file-based *row* and *column* swaps separately. Let each operation be measured *ntrial* times, $ntrial \geq 3$. Make a single plot of the *row* and *column* swap average times on the y-axis ($log_{10}(time)$) and the problem dimension on the x-axis. Submit your header file **file_swaps.hpp** and plot. You may find the code snippet helpful.

```
void swapRowsInFile(std::fstream &file, int nRows, int nCols, int i, int j);
void swapColsInFile(std::fstream &file, int nRows, int nCols, int i, int j);

// snippet
#include <iostream>
#include <fstream>
#include <vector>
#include <utility>
#include <algorithm>
#include <cstdlib>
#include <ctime>
#include <cstdio>
#include <chrono>
#include "file_swaps.hpp"

int main(int argc, char *argv[])
{
// Generate the matrix
std::vector<double> matrix(numRows * numCols);
// init matrix elements in column major order
// write the matrix to a file
std::fstream file(filename, std::ios::out | std::ios::binary);
file.write(reinterpret_cast<char *>(&matrix[0]), numRows * numCols * sizeof(double));
file.close();
// Open the file in read-write mode for swapping
std::fstream fileToSwap(filename, std::ios::in | std::ios::out | std::ios::binary);
// Get random indices i and j for row swapping
// Measure the time required for row swapping using file I/O
auto startTime = std::chrono::high_resolution_clock::now();
// Swap rows i and j in the file version of the matrix
swapRowsInFile(fileToSwap, numRows, numCols, i, j);
auto endTime = std::chrono::high_resolution_clock::now();
std::chrono::duration<double> duration = endTime - startTime;
// Close the file after swapping
fileToSwap.close();
//...
// after each problem size delete the test file
std::remove(filename.c_str());
// ...
}
```

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;
    }
```

Homework 2 (82 points)

1. (+10) Write a C ++ program that finds a practical measure of your machine's SP (32 bit) and DP (64 bit) floating point precision by taking the difference of 2 numbers and comparing these to zero in the same precision. What value do you obtain for $\epsilon_{\text{machine}}$ in both precisions? Hint: A loop (j) will be helpful here $1 - \left(1 + \frac{1}{2^j}\right)$ should do it.
2. (+12) What are the largest and smallest SP (32 bit) and DP (64 bit) numbers that can be represented in IEEE floating point representation? Show work in terms of sign, mantissa, and exponent.
3. (+5) Write a C++ program to multiply the integers $200 * 300 * 400 * 500$ on your computer? What is the result? Name the effect you observe.
4. (+5) Given C++ code segment below, what is the final value of *counter*?

```
unsigned int counter = 0;
for (int i = 0; i < 3; ++i) --counter;
```

5. (+10) Count and report how many IEEE SP (32 bit) normalized and denormalized floating point numbers there are. Please count and label infinities and NANs as well. Show work.
6. (+15) Consider a 6 bit floating point system with $s = 1$ (1 sign bit), $k = 3$ (3 bit exponent field), and $n = 2$ (2 bit mantissa).
   (a) Calculate by hand all the representable normalized numbers. Show work.
   (b) Calculate by hand all the representable denormalized numbers. Show work.
   (c) Plot both sets of numbers (ignoring NANs and infinities) as a number line to see the gaps of (un)representable numbers.
7. (+10) Conversions. Show work.
   (a) Write $(D3B701)_{16}$ as an integer in base-10.
   (b) Write $(1010000100111111)_2$ as an integer in base-16, i.e. as a hexadecimal number.
8. (+5) Are there $a, b, c \in \mathbb{Z}$ s.t. $6a + 9b + 15c = 107$? Show work.
9. (+10) Equivalence classes modulo n. $\forall a, b \in \mathbf{Z}$ then $a \equiv b \pmod{n}$ means $n \mid (a - b)$ or $a = b + k \cdot n$ and $k \in \mathbf{Z}$. $\mathbf{Z}_n$ is the set of equivalence classes $\{[0], [1], \ldots, [n-1]\}$. Is $(\mathbf{Z}_n, +, \cdot)$ a ring? Hint: If $s \in [i]$, then $n \mid (s - i)$. Show work (use ring properties).

Homework 1 (85 points)

1. (+15) Integrate the following integrals. Show work.
   (a) $\int_0^{\frac{\pi}{2}} ln\, sin(x)\, dx$.
   (b) $\int_0^{\pi} \frac{x\, sin(x)}{1+cos^2(x)}\, dx$. Use $\int_0^a f(x)\, dx = \int_0^a f(a-x)\, dx$.
   (c) $\int_0^{\frac{\pi}{2}} \frac{1}{1+(tan(x))^{\sqrt{2}}}\, dx$.
2. (+15) Show work.
   (a) Find $a, b \in \mathbb{R} \ni (1+i\sqrt{3})^{11} = a + ib$.
   (b) Find values of $(1+i\sqrt{3})^{\frac{1}{5}}$.
   (c) Solve for $w \in \mathbb{C}$ given $w^{\frac{4}{3}} + 2i = 0$.
3. (+15) Write the following as the ratio of integers. Show work.
   (a) $1 + 10^{-2} + 10^{-4} + 10^{-6} + \cdots$
   (b) $376.376376\ldots$
   (c) $.999\overline{9}$
4. (+10) Estimate the following as the ratio of integers using the secant approximation. Show work.
   (a) $(1.1)^{\frac{1}{3}}$
   (b) $\sqrt{8.5}$
5. (+10) Given $(x+y+z)^7$, find the expansion coefficients of the following terms. Show work.
   (a) $x^2y^2z^3$
   (b) x^3z^4
6. (+5) Given $(x+2y-3z+2w+5)^{16}$, find the expansion coefficient of the following term. Show work.
   (a) $x^2y^3z^2w^5$
7. (+10) Two numbers $a, b \in \mathbb{Z}$ are relatively prime when $gcd(a,b) = 1$, or $\exists x, y \in \mathbb{Z}$ with $ax + by = 1$. Recall, $c = gcd(a,b)$ when $c|a$ and $c|b$, and for any other divisor d of a, b then $d|c$.
   (a) For any $n \in \mathbb{Z}^+$, prove $8n+3$ and $5n+2$ are relatively prime. Show work. Hint: try Euclid's algorithm
   (b) Find the $gcd(250, 111)$ and show result as linear combination of these integers. Show work.
8. (+5) Write the prime factorization of 980220.

Final Exam - take home (80 points)

1. (+20) **C++ generic sparse vector representation**. Please submit file `mysparse.hpp` with your solutions and test these with the example driver routine provided.
   (a) (+10) Implement the constructor for the C++ template class `mysparse` using the starter code `mysparse.hpp`.
   (b) (+10) Implement the class method for the generic dot product for sparse vectors using the sparse data structure you construct. You can see how it will be used looking at the driver routine.

2. (+20) **Fourier transform diffusion equation solve**. Submit your written work and solution. Consider the diffusion equation $\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2}$ where $T(x,t)$ describes the temperature profile of a long metal rod.
   (a) Assume you know $T(x,0)$ and define the Fourier transform of $T(x,t)$ to be $\tau(k,t)$. Transform the original equation and initial conditions into k-space. Solve the resulting equation. Inverse transform the result to obtain the solution in terms of the original variables.
   (b) Find the temperature in the rod given initial conditions $\kappa = 10^3 \frac{m^2}{s}$ and

$$T(x,0) = \begin{cases} 0 & |x| > 1m \\ 100^o \text{ C} & |x| \leq 1m \end{cases} .$$

3. (+20) **C++ Pattern matching**. You are provided starter code in `match_pattern.hpp`, a data file `output.txt` which contains a continuous string of 268,435,456 characters (256 MB of data), and a driver routine.
   (a) (+10) Implement your C++ pattern matching function for the following api and submit file `match_pattern.hpp`.

```
void matchPattern(const string& text, const string& pattern);
```

   (b) (+10) Execute your function on the following strings: **CDEEJKL**, **ABDCK**, **FFGGHH**, **FINDME**, **HELLO**. For each string submit the number of times the pattern (string) was found **and** the start positions (first character in the input data file is in position 0).

4. (+20) **Hamilton paths and cycles**. A **Hamilton path** in a graph $G(V,E)$ is a path that visits each vertex exactly once. Formally, a Hamilton path is a path $P = (v_1, v_2, \ldots, v_n)$ such that each v_i is a distinct vertex of V and every pair $e(v_i, v_{i+1})$ is an edge in E. A **Hamilton cycle** (or Hamiltonian cycle) in a graph G is a cycle that visits each vertex exactly once and returns to the starting vertex. Formally, a Hamilton cycle is a cycle $C = (v_1, v_2, \ldots, v_n, v_1)$ such that each v_i is a distinct vertex of V (except v_1, which appears twice, at the beginning and the end) and every pair $e(v_i, v_{i+1})$ is an edge in E, with $e(v_n, v_1)$ also being an edge.
   (a) (+20) Submit your written work and solutions. Starting at vertex **a**, find a Hamilton cycle, if it exists, for each of the graphs (a-f) or multi-graphs in Figure[1]. If the graph has no Hamilton cycle, determine if it has a Hamilton path and report this if possible. If neither exist, just say so. Submit your work on a

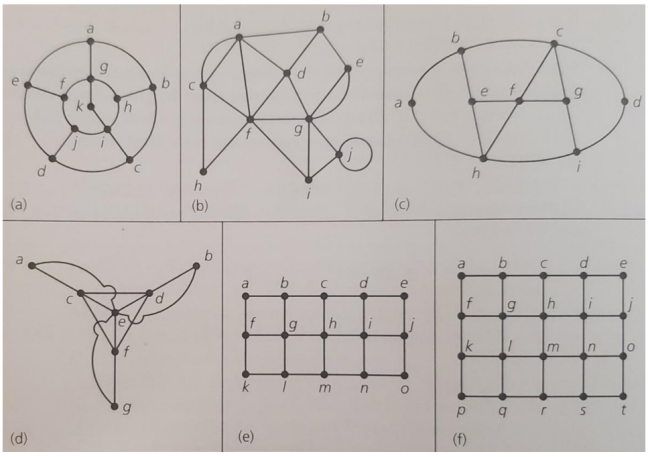

Figure 1: Hamilton path / cycle

Homework 6 (80 points, 0 EC points)

1. (+20) **Complex double linear system solver**. Plot both the log of the residual and the log of the normalized error ($\frac{\|b-Az\|_2}{\|A\|_\infty \|z\|_2 \epsilon_{\text{machine}}}$) versus the square matrix dimensions 16,32,64,...,8192 for the following LAPACK routine. It is supported in the OpenBLAS build on Hyak. Submit your plot, and label it accordingly.

```
lapack_int LAPACKE_zgesv(int matrix_order,
                         lapack_int n,
                         lapack_int nrhs,
                         lapack_complex_double* a,
                         lapack_int lda,
                         lapack_int* ipiv,
                         lapack_complex_double* b,
                         lapack_int ldb);
```

   Use the following snippet code to initialize your matrices and rhs vectors and note the headers I use:

```
#include <iostream>
#include <complex>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <vector>
#include <chrono>
#include <limits>
#include <cblas.h>
#include <lapacke.h>

...
int main() {
...
a =(std::complex<double>*)malloc(sizeof(std::complex<double>) * ma * na);
b = (std::complex<double>*)malloc(sizeof(std::complex<double>) * ma);
z = (std::complex<double>*)malloc(sizeof(std::complex<double>) * na);
...
srand(0);
int k =0;
for (int j = 0; j < na; j++) {
    for (int i = 0; i < ma ; i++) {
        a[k] = 0.5 - (double)rand() / (double)RAND_MAX
        + std::complex<double>(0, 1)
        * (0.5 - (double)rand() / (double)RAND_MAX);
        if (i==j) a[k]*=static_cast<double>(ma);
        k++;
    }
}
srand(1);
for (int i = 0; i < ma; i++) {
    b[i] = 0.5 - (double)rand() / (double)RAND_MAX
    + std::complex<double>(0, 1)
    * (0.5 - (double)rand() / (double)RAND_MAX);
}
...
```

2. (+20) **CPU-GPU data copy speed on HYAK**. Write a C++ code to measure the data copy performance between the host CPU and GPU (host to device), and between the GPU and the host CPU (device to host). Copy 8 bytes to 256MB increasing in multiples of 2. Plot the bandwidth for both directions: (bytes per second) on the y-axis and the buffer size in bytes on the x-axis. Submit your plot and test code.
3. (+20) **Compare FFTW to CUFFT on HYAK**. Measure and plot the performance of calculating the gradient of a 3D double complex plane wave defined on cubic lattices of dimension n^3 from 16^3 to $n = 256^3$, stride $n* = 2$ for both the FFTW and CUDA FFT (CUFFT) implementations on HYAK. Let each n be measured *ntrial* times and plot the average performance for each case versus n, $ntrial \geq 3$. Submit your performance plot which should have 'FLOPs' on the y-axis (or some appropriate unit of FLOPs) and the dimension of the cubic lattices (n) on the x-axis. You will need to estimate the operation count of computing the derivative using FFT on a lattice.
4. (+20) **Fourier transforms**. Evaluate the Fourier transform of the following functions by hand. Use the definitions I provided (includes $\frac{1}{\sqrt{2\pi}}$, this is common in physics but also now the default used in WolframAlpha - a powerful math AI tool) as well as the definition for Dirac delta I used in lecture if needed.

(a) $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$

(b) $f(t) = sin(\omega_0 t)$, ω_0 constant

(c) $f(x) = e^{-a|x|}$ and $a > 0$

(d) (distribution) $f(t) = \delta(t)$

Homework 5 (85 points, 10 EC points)

1. (+20) **John Conway's Game of Life**. Please modify the function

   ```
   void updateGrid(std::vector<std::vector<bool>& grid);
   ```

   from the starter code `conway-life.cpp` (with Python script `conway_display.py`) to implement the threaded function

   ```
   void updateGrid(std::vector<std::vector<bool>& grid, int gridSize, int numThreads);
   ```

   Put your solution in file `threaded_life.cpp`, and submit this file. Your implementation should handle the cases of `nthreads = 1, 4, 16`. You may find the codes `conway-life-grid-init.cpp` and `conway-life-compare-grids.cpp` useful for checking your answers as I will use some of the functions therein to grade your work. You will have to introduce boundary updates using ghost cells.

2. (+25) **Elevator scheduler**. I provide `hw5-elevator.cpp` and snippet `hw5-elevator.hpp` codes. Edit the header file to implement the two functions:

   ```
   void elevator(int id); void person(int id);
   ```

   Please don't change the global variables, but feel free to add what is needed for your functions. You will submit the file `hw5-elevator.hpp`. The exercise is to write a threaded elevator scheduler for a 50-story business building with 6 elevators. Up to 7500 people may visit the building in a day. Your functions should use the C++ threads library to simulate the behavior of the elevators and visitors of the building. The driver code takes as input the number of people to service (hint: makes it a bounded buffer). After servicing all person requests, have each elevator report the number of passengers it transported, and be certain that your code exits cleanly. Please print events to *cout* in a manner that is thread safe. For each person, report their ID and current floor and destination floor (i.e. *Person 0 wants to go from floor 7 to floor 9*), report the elevator number they get on (i.e. *Person 1 entered elevator 0*), report when they arrive at their target floor (i.e. *Person 98 arrived at floor 18*). For each elevator, report the direction of movement from current floor to next floor (i.e. *Elevator 2 moving from floor 12 to floor 19*), join all the elevator threads once all visitors have arrived at their floors, and report success (i.e. *Job completed!*).

   - randomly select the current floor and destination floor for each person
   - I let each elevator start from the ground floor if it matters
   - issue a person thread in the *detach* state (i.e. *thread(person, i).detach();*)
   - each elevator will be it's own joinable thread
   - keep track of the current floor and direction of each elevator
   - keep track of the destinations of the visitors who are waiting for elevators on each floor
   - determine which elevator should respond to a given request, based on its current location, direction, and the number of passengers it is already carrying
   - the maximum occupancy of the elevators is fixed, say 10
   - move the elevators up or down to their next destination, picking up and dropping off passengers as necessary
   - print events to screen as they occur
   - add some sleep functions to mimic the action of the elevators moving between floors

3. (+20) **OpenBLAS L1, L2, L3**. We will now upgrade the performance of our BLAS efforts. Measure and plot the performance of double precision L1, L2, and L3 BLAS using the OpenBLAS library and the functions stated here. Use appropriate vector and matrix of dimensions for problem sizes $n = 16$ to $n = 4096$, stride $n* = 2$. Let each n be measured *ntrial* times and plot the average performance for each instance versus n, $ntrial \geq 3$. Submit a single performance plot. Your plot will have 'flops' on the y-axis, or some variation of FLOPs such as MFLOPs, and the problem size on the x-axis.

   ```
   void cblas_daxpy(const int N, const double alpha, const double *X,
   const int incX, double *Y, const int incY);

   void cblas_dgemv(const enum CBLAS_ORDER order, const enum CBLAS_TRANSPOSE TransA,
   const int M, const int N, const double alpha, const double *A, const int lda,
   const double *X, const int incX, const double beta, double *Y, const int incY);

   void cblas_dgemm(const enum CBLAS_ORDER Order, const enum CBLAS_TRANSPOSE TransA,
   const enum CBLAS_TRANSPOSE TransB, const int M, const int N, const int K,
   const double alpha, const double *A, const int lda, const double *B,
   const int ldb, const double beta, double *C, const int ldc);
   ```

4. (+20) **Compare OpenBLAS to CUBLAS on HYAK**. Measure and plot the performance of double precision matrix multiply ($\alpha AB + \beta C \rightarrow C$) for square matrices of dimension $n = 16$ to $n = 8192$, stride $n* = 2$ for both the OpenBLAS and CUDA BLAS (CUBLAS) implementations on HYAK. Let each n be measured *ntrial* times and plot the average performance for each case versus n, $ntrial \geq 3$. Submit your performance plot and C++ test code. Your plot will have 'flops' on the y-axis, or some variation of FLOPs such as MFLOPs, and the dimension of the matrices on the x-axis.
5. (+10EC) **Matrix transpose**. Write C++ functions given the two APIs that compute A^T, the matrix transpose of a matrix stored in a single vector container in the column major index scheme. Test the correctness of both functions. Put the functions in file **transpose.hpp** which I will include in my test code for grading. Submit file **transpose.hpp**. Note the threaded function will create and join the threads internally.

```
void sequentialTranspose(std::vector<int> &matrix, int rows, int cols);
void threadedTranspose(std::vector<int> &matrix, int rows, int cols, int nthreads);
```

Conway's Game of Life

Updating the State (Conway's Rules)

The `updateGrid` function applies Conway's rules to update the state of the grid for each generation. Conway's rules are as follows:

1. **Underpopulation:** Any live cell with fewer than two live neighbors dies, as if by underpopulation.
2. **Survival:** Any live cell with two or three live neighbors survives to the next generation.
3. **Overcrowding:** Any live cell with more than three live neighbors dies, as if by overcrowding.
4. **Reproduction:** Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

The `updateGrid` function iterates through each cell in the grid and counts its live neighbors (including diagonals). Based on the count of live neighbors, it applies these rules to determine whether the cell should be alive or dead in the next generation. The new state of the grid is stored in a separate grid, and then it replaces the original grid at the end of the iteration.

In the provided code, the cell values in the grid are represented as boolean values, where `false` represents a dead cell (0) and `true` represents a live cell (1). This binary representation simplifies the implementation and makes it easier to apply Conway's rules, as each cell's state is effectively represented by a single bit.

Summary of Representation

- `false` (0): Represents a dead cell.
- `true` (1): Represents a live cell.

The rules for updating the grid, based on Conway's Game of Life, still apply using this binary representation. The rules determine whether a cell should transition from being alive to dead, or vice versa, based on the number of live neighbors it has.

How the Code Works

Let's go through each rule and how it's satisfied in the code:

1. **Underpopulation:** Any live cell with fewer than two live neighbors dies.

   This rule is implemented in the code by checking if a live cell (`grid[i][j] == true`) has fewer than two live neighbors (`aliveNeighbors < 2`). If so, the corresponding cell in the new grid (`newGrid[i][j]`) is set to `false` (dead).
2. **Survival:** Any live cell with two or three live neighbors survives to the next generation.

   This rule is implemented similarly by checking if a live cell has two or three live neighbors (`aliveNeighbors == 2 || aliveNeighbors == 3`). If so, the corresponding cell in the new grid remains alive.
3. **Overcrowding:** Any live cell with more than three live neighbors dies.

   This rule is implemented by checking if a live cell has more than three live neighbors (`aliveNeighbors > 3`). If so, the corresponding cell in the new grid is set to `false` (dead).
4. **Reproduction:** Any dead cell with exactly three live neighbors becomes a live cell.

   This rule is implemented by checking if a dead cell (`grid[i][j] == false`) has exactly three live neighbors (`aliveNeighbors == 3`). If so, the corresponding cell in the new grid is set to `true` (alive).

Homework 4 (170 points)

1. (+10) 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 ∞-norm (induced matrix norms). Submit your work and drawings.
2. (+20) **Row major Matrix class**. Reference the file `matrix_class.hpp` for the starter code for a Matrix class template for row major index referencing and `std::vector<T>` for the matrix storage scheme. Please put your function implementations in file `matrix_class.hpp`, and submit `matrix_class.hpp`.
   - (+5) Matrix transpose for $A \in \mathbb{R}^{m \times n}$ is defined $A^T_{i,j} = A_{j,i}$ and so $A^T \in \mathbb{R}^{n \times m}$. This method returns a matrix as defined by the class.
     - `Matrix<T> transpose() const{}`
   - (+5) Matrix infinity norm for $A \in \mathbb{R}^{m \times n}$ is defined $\|A\|_\infty = \max_{0 \le i \le m} \sum_{j=1}^{j=n} |A_{i,j}|$. This method returns a number.
     - `T infinityNorm() const{}`
   - (+5) Write the method to operator overload multiplication $*$ for the matrix class I provided. This method returns a matrix as defined by the class.
     - `Matrix<T> operator*(const Matrix<T> &other) const{}`
   - (+5) Write the method to operator overload addition + for the matrix class I provided. This method returns a matrix as defined by the class.
     - ```
       template <typename T>
       Matrix<T> Matrix<T>::operator+(const Matrix<T> &other) const{}
       ```
3. (+10) **Extremum**. Consider the surface defined by $xy + 2xz = 5\sqrt{5}$ and $(x, y, z) \in \mathbb{R}^3$. Find (a) the coordinate instance(s) affiliated with the minimum distance from a point on the surface to the origin, and (b) the value of the minimum distance. You will find (may safely assume) the domain $[-3, 3] \times [-3, 3] \times [-3, 3] \in \mathbb{R}^3$ holds the correct coordinate instance(s).
4. (+10) **IO bandwidth**. Write a C++ function that `write`s type `double` square matrices in column major order to file in binary. Measure the time required to complete the `write` for matrices of dimension 32, 64, 128, ... 16384 (2GB). Write a C++ function that `read`s binary matrices from file to type `double` matrices in memory. Measure the time required to complete each `read` for the same dimensions. (a) Make a single plot of the *read* and *write* measurements with the bandwidth (bytes per second) on the y-axis, and the problem dimension on the x-axis. Submit your plot.
5. (+20) **File access time**. (a) Write C++ functions for the given function declarations that perform *row* and *column* swap operations on a type `double` matrix stored in a file in column major index order. Test the swapping capabilities for correctness. Put the functions you write in file **file_swaps.hpp**. (b) Conduct a performance test for square matrix dimensions 16, 32, 64, 128, ... 8192, measuring the time required to conduct file-based *row* and *column* swaps separately. Let each operation be measured *ntrial* times, $ntrial \ge 3$. Make a single plot of the *row* and *column* swap average times on the y-axis ($log_{10}(time)$) and the problem dimension on the x-axis. Submit your header file **file_swaps.hpp** and plot. You may find the code snippet helpful.

```
void swapRowsInFile(std::fstream &file, int nRows, int nCols, int i, int j);
void swapColsInFile(std::fstream &file, int nRows, int nCols, int i, int j);

// snippet
#include <iostream>
#include <fstream>
#include <vector>
#include <utility>
#include <algorithm>
#include <cstdlib>
#include <ctime>
#include <cstdio>
#include <chrono>
#include "file_swaps.hpp"

int main(int argc, char *argv[])
{
// Generate the matrix
```
```

```
std::vector<double> matrix(numRows * numCols);
// init matrix elements in column major order
// write the matrix to a file
std::fstream file(filename, std::ios::out | std::ios::binary);
file.write(reinterpret_cast<char *>(&matrix[0]), numRows * numCols * sizeof(double));
file.close();
// Open the file in read-write mode for swapping
std::fstream fileToSwap(filename, std::ios::in | std::ios::out | std::ios::binary);
// Get random indices i and j for row swapping
// Measure the time required for row swapping using file I/O
auto startTime = std::chrono::high_resolution_clock::now();
// Swap rows i and j in the file version of the matrix
swapRowsInFile(fileToSwap, numRows, numCols, i, j);
auto endTime = std::chrono::high_resolution_clock::now();
std::chrono::duration<double> duration = endTime - startTime;
// Close the file after swapping
fileToSwap.close();
//...
// after each problem size delete the test file
std::remove(filename.c_str());
// ...
}
```

6. (+20) **Length of a graph** of $f(x)$ on $[a,b]$ is $L = \int_a^b \sqrt{1+(f'(x))^2}dx$. Given $f(x) = ln(x) - \frac{1}{8}x^2$ on $[1,6]$. (a) Find the length of f on the interval analytically. You will need this to calculate the error in your codes. (b) Write a C++ code parallelized using the **C++ threads** library that numerically evaluates the length of this function on the interval using Riemann sum. Your code should accept the number of points used to partition the interval and the number of threads to spawn. (c) Plot the strong scaling efficiency on 1,2,4,8,16 threads of your code for $n = 1.e8$ partition points, i.e. time versus thread count. (d) Plot the *log* of the numerical error for $10, 100, ...1.e6$ partition points (increasing by factors of $10x$ each time). Submit your code and plots.

7. (+20) **Length of a graph** of $f(x)$ on $[a,b]$ is $L = \int_a^b \sqrt{1+(f'(x))^2}dx$. Given $f(x) = ln(x) - \frac{1}{8}x^2$ on $[1,6]$. (a) Write a C++ code parallelized using **MPI** that numerically evaluates the length of this function on the interval using Riemann sum. Your code should accept the number of points used to partition the interval. (b) Plot the strong scaling efficiency of your code on $1, 2, 4, 8, 16, 32$ processes for $n = 1.e8$ partition points using Hyak. (c) Plot the *log* of the numerical error for $10, 100, ...1.e6$ partition points (increasing by factors of $10x$ each time). Submit your code and plots.

8. (+20) **Linear algebra freebies**.

   (a) (+10) **Basis**. Does the given set $S = \{1, 1-x, (1-x)^2\}$ form a basis for the space of polynomials up to degree 2, $x \in \mathbb{R}$? Show your work. (Hint: must be both linearly independent and span the space)

   (b) (+10) **Gram-Schmidt**. Given set $S = \{x_1(t), x_2(t), x_3(t)\} \ni x_1(t) = t^2, \quad x_2(t) = t, \quad x_3(t) = 1$ and $t \in [-1,1] \in \mathbb{R}$, construct by hand an orthonormal basis for S using the inner product $(x,y) = \int_{-1}^{1} x(t)y(t)\,dt$.

9. (+20) **Communication Bandwidth**. (a) Use the following template declaration to write a **MPI** C++ function using point-to-point communication that implements broadcast. Submit the function in file `my_broadcast.hpp`. (b) Measure and plot the performance of your function and the `int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)` function on messages of size $8B, 16B, ..., 256MB$ bytes (doubling the message size each instance). Do this exercise using Hyak on 4 and 32 MPI processes, and make a plot of your measurements with bandwidth (bytes per second) on the y-axis, and the message size in bytes on the x-axis. Submit your plot.

   - ```
     template <typename T>
     void my_broadcast(T* data, int count, int root, MPI_Comm comm);
     ```

10. (+20) **Strassen**. Use notes from the class lecture to implement the C++ template for the (recursive) Strassen matrix multiplication algorithm for matrices with even dimension, `C = strassenMultiply(A, B);`. Use the provided starter code `strassen.cpp` and implement in it the declared method `strassenMultiply`. Plot the average performance (FLOPs) multiplying `double` precision square matrices of even dimension from $n = 2$ to $n = 512$ as measured *ntrial* times, $ntrial \geq 3$. Submit `strassen.cpp` and performance plot.
     ```

Exam 1 (90 points, 0 extra credit (EC) points)

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;
    }
```

1. (+30) Matrix Multiplication Loop Permutations. Implement templated gemm, $C \leftarrow \alpha AB + \beta C$ ($A \in \mathbb{T}^{m \times p}$, $B \in \mathbb{T}^{p \times n}$, $C \in \mathbb{T}^{m \times n}$, $\alpha, \beta \in \mathbb{T}$), for each $\{i, j, k\}$ for-loop permutation using the specifications provided here:

    - ```
      template<typename T>
      void mm_ijk(T a, const std::vector<T>& A, const std::vector<T>& B, T b,
      std::vector<T>& C, int m, int p, int n);
      ```

    - ```
      template<typename T>
      void mm_jki(T a, const std::vector<T>& A, const std::vector<T>& B, T b,
      std::vector<T>& C, int m, int p, int n);
      ```

    - ```
      template<typename T>
      void mm_kij(T a, const std::vector<T>& A, const std::vector<T>& B, T b,
      std::vector<T>& C, int m, int p, int n);
      ```

    - ```
      template<typename T>
      void mm_jik(T a, const std::vector<T>& A, const std::vector<T>& B, T b,
      std::vector<T>& C, int m, int p, int n);
      ```

    - ```
      template<typename T>
      void mm_ikj(T a, const std::vector<T>& A, const std::vector<T>& B, T b,
      std::vector<T>& C, int m, int p, int n);
      ```

    - ```
      template<typename T>
      void mm_kji(T a, const std::vector<T>& A, const std::vector<T>& B, T b,
      std::vector<T>& C, int m, int p, int n);
      ```

Pay special attention that your matrices will now be represented within a single vector in this exercise. Please utilize column major ordering as defined in lecture when assigning and accessing matrix elements in this format. Reject faulty cases with a quick return by checking the dimensions in each function. Turn in the following C++ source codes and header files:

- exam1_problem1.hpp
- mm-ijk.cpp, mm-ijk.hpp
- mm-kij.cpp, mm-kij.hpp
- mm-jki.cpp, mm-jki.hpp
- mm-jik.cpp, mm-jik.hpp
- mm-kji.cpp, mm-kji.hpp
- mm-ikj.cpp, mm-ikj.hpp

Your submission will be compiled and tested for correctness on simple cases as shown here:

```
g++ -c mm-ijk.cpp
g++ -c mm-jki.cpp
g++ -c mm-kij.cpp
g++ -c mm-jik.cpp
g++ -c mm-ikj.cpp
g++ -c mm-kji.cpp
g++ -std=c++14 -o xtst-ijk-perms tst-ijk-permutations.cpp mm-ijk.o \
                                 mm-jki.o mm-kij.o mm-jik.o mm-ikj.o mm-kji.o
```

Hints:
File exam1_problem1.hpp:

```
#ifndef EXAM1_PROBLEM1_HEADER_HPP
#define EXAM1_PROBLEM1_HEADER_HPP
#include "mm-ijk.hpp"
#include "mm-kij.hpp"
#include "mm-jki.hpp"
#include "mm-jik.hpp"
#include "mm-kji.hpp"
#include "mm-ikj.hpp"
#endif
```

File mm-ijk.hpp:

```
#ifndef MM_IJK_HPP
#define MM_IJK_HPP
#include <vector>
template<typename T>
void mm_ijk(T a, const std :: vector<T>& A,
                 const std :: vector<T>& B,
            T b, std::vector<T>&C,
            int m, int p, int n);
#endif
```

File mm-ijk.cpp:

```
#include <iostream>
#include <vector>
// ijk
template <typename T>
void mm_ijk(T a, const std::vector<T> &A, const std::vector<T> &B,
            T b, std::vector<T> &C, int m, int p, int n)
{
    // Check if the dimensions are valid for multiplication
    // Compute aAB+bC
    // column major - ijk
}
template void mm_ijk<double>(double a, const std::vector<double> &A,
const std::vector<double> &B, double b, std::vector<double> &C,
int m, int p, int n);
template void mm_ijk<float>(float a, const std::vector<float> &A,
```

```
const std::vector<float> &B, float b, std::vector<float> &C,
int m, int p, int n);
template void mm_ijk<int>(int a, const std::vector<int> &A,
const std::vector<int> &B, int b, std::vector<int> &C,
int m, int p, int n);
```

2. (+30) Memory access time. Write C++ functions that perform **row and column swap** operations on a type double matrix stored in column major index order using a single vector container for the data. The specifications are provided here. Put the functions in file **matrix_swaps.hpp** (no need for header guards -just the code for your functions). Conduct a performance test for square matrix dimensions 16, 32, 64, ... 4096, measuring the time required to conduct row swaps and column swaps separately. Let each operation be measured *ntrial* times, $ntrial \geq 3$. Make a single plot of your *row* and *column* swap timing measurements with time on the y-axis (could be $log_{10}(time)$) and the problem dimension on the x-axis. Submit files **mem_swaps.hpp** and the plot.

   ```
   void swapRows(std::vector<double> &matrix, int nRows, int nCols, int i, int j);
   void swapCols(std::vector<double> &matrix, int nRows, int nCols, int i, int j);
   ```

   You may test the swap capabilities on randomly selected index pairs using a function like this:

   ```
   #include <utility>   // For std::pair
   std::pair<int, int> getRandomIndices(int n)
   {
       int i = std::rand() % n;
       int j = std::rand() % (n - 1);
       if (j >= i)
       {
           j++;
       }
       return std::make_pair(i, j);
   }
   // ... from inside main()
   //std::pair<int, int> rowIndices = getRandomIndices(M);
   //int i = rowIndices.first;
   //int j = rowIndices.second;
   //std::pair<int, int> colIndices = getRandomIndices(N);
   // ...
   ```

3. (+20) Ring. $\forall x, y \in \mathbb{Z}$ define binary operator $\oplus$ as $x \oplus y = x + y - 1$, and binary operator $\odot$ as $x \odot y = x + y - xy$. $(\mathbb{Z}, \oplus, \odot)$ is a ring.

   (a) Additive Identity. Find $z \in \mathbb{Z} \ni a \oplus z = z \oplus a = a, \forall a \in \mathbb{Z}$.

   (b) Additive Inverse. $\forall a \in \mathbb{Z}, \exists b \in \mathbb{Z} \ni a \oplus b = b \oplus a = z$. Find b.

   (c) Show that $\oplus$ commutes.

   (d) Multiplicative Identity. A ring with unity is a ring with member $u \in \mathbb{Z}, u \neq z \ni a \odot u = u \odot a = a, \forall a \in \mathbb{Z}$. Find u.

4. (+5) Write the 6-bit representation of $+\infty$ and $-\infty$.

5. (+5) Define the vector 1-norm, 2-norm, and ∞-norm, and the matrix ∞-norm, for $x \in \mathbb{R}^n$.

(a) Find $a, b \in \mathbb{R} \ni (1 + i\sqrt{3})^{11} = a + ib$.

(b) Find values of $(1 + i\sqrt{3})^{\frac{1}{5}}$.

(c) Solve for $w \in \mathbb{C}$ given $w^{\frac{4}{3}} + 2i = 0$.

1. (+15) Write the following as the ratio of integers. Show work.

   (a) $1 + 10^{-2} + 10^{-4} + 10^{-6} + \cdots$

   (b) $376.376376\ldots$

   (c) $.999\overline{9}$

2. (+10) Estimate the following as the ratio of integers using the secant approximation. Show work.

   (a) $(1.1)^{\frac{1}{3}}$

   (b) $\sqrt{8.5}$

3. (+10) Given $(x + y + z)^7$, find the expansion coefficients of the following terms. Show work.

   (a) $x^2y^2z^3$

   (b) x^3z^4

4. (+5) Given $(x + 2y - 3z + 2w + 5)^{16}$, find the expansion coefficient of the following term. Show work.

   (a) $x^2y^3z^2w^5$

5. (+10) Two numbers $a, b \in \mathbb{Z}$ are relatively prime when $gcd(a, b) = 1$, or $\exists x, y \in \mathbb{Z}$ with $ax + by = 1$. Recall, $c = gcd(a, b)$ when $c|a$ and $c|b$, and for any other divisor d of a, b then $d|c$.

   (a) For any $n \in \mathbb{Z}^+$, prove $8n + 3$ and $5n + 2$ are relatively prime. Show work. Hint: try Euclid's algorithm

   (b) Find the $gcd(250, 111)$ and show result as linear combination of these integers. Show work.

6. (+5) Write the prime factorization of 980220.

Homework 1 (70 points, 0 extra credit (EC) points)

1. (+15) Show work.
   (a) Find $a, b \in \mathbb{R} \ni (1 + i\sqrt{3})^{11} = a + ib$.
   (b) Find values of $(1 + i\sqrt{3})^{\frac{1}{5}}$.
   (c) Solve for $w \in \mathbb{C}$ given $w^{\frac{4}{3}} + 2i = 0$.
2. (+15) Write the following as the ratio of integers. Show work.
   (a) $1 + 10^{-2} + 10^{-4} + 10^{-6} + \cdots$
   (b) $376.376376\ldots$
   (c) $.999\overline{9}$
3. (+10) Estimate the following as the ratio of integers using the secant approximation. Show work.
   (a) $(1.1)^{\frac{1}{3}}$
   (b) $\sqrt{8.5}$
4. (+10) Given $(x + y + z)^7$, find the expansion coefficients of the following terms. Show work.
   (a) $x^2y^2z^3$
   (b) x^3z^4
5. (+5) Given $(x + 2y - 3z + 2w + 5)^{16}$, find the expansion coefficient of the following term. Show work.
   (a) $x^2y^3z^2w^5$
6. (+10) Two numbers $a, b \in \mathbb{Z}$ are relatively prime when $gcd(a, b) = 1$, or $\exists x, y \in \mathbb{Z}$ with $ax + by = 1$. Recall, $c = gcd(a, b)$ when $c|a$ and $c|b$, and for any other divisor d of a, b then $d|c$.
   (a) For any $n \in \mathbb{Z}^+$, prove $8n + 3$ and $5n + 2$ are relatively prime. Show work. Hint: try Euclid's algorithm
   (b) Find the $gcd(250, 111)$ and show result as linear combination of these integers. Show work.
7. (+5) Write the prime factorization of 980220.

Take Home Final (150 points, 20 extra credit points (EC))

1. (+20) **Fourier transforms**. Evaluate the Fourier transform of the following functions by hand. Use the definitions I provided (includes $\frac{1}{\sqrt{2\pi}}$, this is common in physics but also now the default used in WolframAlpha - a powerful math AI tool) as well as the definition for Dirac delta I used if needed.
   (a) $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$
   (b) $f(t) = sin(\omega_0 t)$, ω_0 constant
   (c) $f(x) = e^{-a|x|}$ and $a > 0$
   (d) (distribution) $f(t) = \delta(t)$

2. (+10) **Correlation**. By definition, *correlation* is $p \odot q = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} p^*(\tau) q(t+\tau) d\tau$, and measures how similar one signal or data function is to another. Let $p(\tau) = \langle p \rangle + \delta_p(\tau)$ and $q(\tau) = \langle q \rangle + \delta_q(\tau)$, where $\langle\rangle$ and $\delta()$ denote the mean values and fluctuation functions (deviations about the mean). Two functions are defined to be *uncorrelated* when $p \odot q = \langle p \rangle \langle q \rangle$. Evaluate $p \odot q$ of the following functions:

$$p(t) = \begin{cases} 0 & t < 0 \\ 1 & 0 < t < 1 \\ 0 & t > 1 \end{cases} \quad , \; q(t) = \begin{cases} 0 & t < 0 \\ 1 - t & 0 < t < 1 \\ 0 & t > 1 \end{cases}$$

3. (+5EC) **Autocorrelation**. Aside, periodic functions exhibit pronounced *autocorrelation*s as shifting such functions by their period puts the function directly on itself. Alternatively, random functions or noise is characterized as being uncorrelated. Evaluate the autocorrelation $p \odot p$ of the following function:

$$p(t) = \begin{cases} 0 & t < 0 \\ 1 & 0 < t < 1 \\ 0 & t > 1 \end{cases}$$

4. (+20) **Fourier transform diffusion equation solve**. Consider the diffusion equation $\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2}$ where $T(x,t)$ describes the temperature profile of a long metal rod.
   (a) Assume you know $T(x,0)$ and define the Fourier transform of $T(x,t)$ to be $\tau(k,t)$. Transform the original equation and initial conditions into k-space. Solve the resulting equation. Inverse transform the result to obtain the solution in terms of the original variables.
   (b) Find the temperature in the rod given initial conditions $\kappa = 10^3 \frac{m^2}{s}$ and

$$T(x,0) = \begin{cases} 0 & |x| > 1m \\ 100^o \text{ C} & |x| \leq 1m \end{cases} .$$

5. (+20) **Compare OpenBLAS to CUBLAS on HYAK**. Measure and plot the performance of double precision matrix multiply ($\alpha AB + \beta C \rightarrow C$) for square matrices of dimension $n = 16$ to $n = 8192$, stride $n* = 2$ for both the OpenBLAS and CUDA BLAS (CUBLAS) implementations on HYAK. Let each n be measured *ntrial* times and plot the average performance for each case versus n, $ntrial \geq 3$. Submit your performance plot and C++ test code. Your plot will have 'flops' on the y-axis, or some variation of FLOPs such as MFLOPs, and the dimension of the matrices on the x-axis.

6. (+10EC) **Matrix transpose**. Write C++ functions given the two APIs that compute A^T, the matrix transpose of a matrix stored in a single vector container in the column major index scheme. Test the correctness of both functions. Put the functions in file **transpose.hpp** which I will include in my test code for grading. Submit file **transpose.hpp**. Note the threaded function will create and join the threads internally.

```
void sequentialTranspose(std::vector<int> &matrix, int rows, int cols);
void threadedTranspose(std::vector<int> &matrix, int rows, int cols, int nthreads);
```

7. (+20) **Memory access time**. On a computer of your choice, write C++ functions for the given APIs that perform row and column swap operations in *memory* on a type double matrix stored in column major index order using a single vector container for the data. Test the swap capabilities on randomly selected index pairs using the function I provide here. Put your functions (not 'getRandomIndices') in file **mem_swaps.hpp**, no need for header guards -just the code for your functions. Conduct a performance test for square matrix dimensions 16, 32, 64, 128, ... 16384, measuring the time required to conduct row and column swaps separately. Let each operation be measured *ntrial* times and plot the average time versus matrix dimension, $ntrial \geq 3$. Make a single plot of your *row* and *column* swap timing measurements with time on the y-axis ($log_{10}(time)$) and the problem dimension on the x-axis. Submit file **mem_swaps.hpp** and plot.

```
void swapRows(std::vector<double> &matrix, int nRows, int nCols, int i, int j);
void swapCols(std::vector<double> &matrix, int nRows, int nCols, int i, int j);

#include <utility>   // For std::pair
std::pair<int, int> getRandomIndices(int n)
{
    int i = std::rand() % n;
    int j = std::rand() % (n - 1);
    if (j >= i)
    {
        j++;
    }
    return std::make_pair(i, j);
}


// ... from inside main()
//std::pair<int, int> rowIndices = getRandomIndices(M);
//int i = rowIndices.first;
//int j = rowIndices.second;
//std::pair<int, int> colIndices = getRandomIndices(N);
// ...
```

8. (+20) **File access time**. On the same computer as problem 1, write C++ functions for the given APIs that perform row and column swap operations on a type `double` matrix stored in column major index order in a *FILE*. Use a randomly selected pair of indices to test the swapping capabilities. Put the functions you write in file **file_swaps.hpp**. Conduct a performance test for square matrix dimensions 16, 32, 64, 128, ... 16384, measuring the time required to conduct *file-based* row and column swaps separately. Let each operation be measured *ntrial* times and plot the average time versus matrix dimension, $ntrial \geq 3$. Make a single plot of your *file-based row* and *column* swap timing measurements with time on the y-axis ($log_{10}(time)$) and the problem dimension on the x-axis. Submit your header file **file_swaps.hpp** and plot.

```
void swapRowsInFile(std::fstream &file, int nRows, int nCols, int i, int j);
void swapColsInFile(std::fstream &file, int nRows, int nCols, int i, int j);

// snippet
#include <iostream>
#include <fstream>
#include <vector>
#include <utility>
#include <algorithm>
#include <cstdlib>
#include <ctime>
#include <cstdio>
#include <chrono>
#include "file_swaps.hpp"

int main(int argc, char *argv[])
{
// Generate the matrix
std::vector<double> matrix(numRows * numCols);
// init matrix elements in column major order
// write the matrix to a file
std::fstream file(filename, std::ios::out | std::ios::binary);
file.write(reinterpret_cast<char *>(&matrix[0]), numRows * numCols * sizeof(double));
file.close();
// Open the file in read-write mode for swapping
std::fstream fileToSwap(filename, std::ios::in | std::ios::out | std::ios::binary);
// Get random indices i and j for row swapping
// Measure the time required for row swapping using file I/O
auto startTime = std::chrono::high_resolution_clock::now();
// Swap rows i and j in the file version of the matrix
swapRowsInFile(fileToSwap, numRows, numCols, i, j);
auto endTime = std::chrono::high_resolution_clock::now();
std::chrono::duration<double> duration = endTime - startTime;
// Close the file after swapping
```

```
fileToSwap.close();
//...
// after each problem size delete the test file
std::remove(filename.c_str());
// ...
}
```

9. (+20) **CPU-GPU data copy speed on HYAK**. Write a C++ code to measure the data copy performance between the host CPU and GPU, and between the GPU and the host CPU. Copy 8 bytes to 256MB increasing in multiples of 2. You will plot the bandwidth for both directions: (bytes per second) on the y-axis, and the buffer size in bytes on the x-axis. Submit your plot and test code.
10. (+20) **Compare FFTW to CUFFT on HYAK**. Measure and plot the performance of calculating the gradient of a 3D double complex plane wave defined on cubic lattices of dimension n^3 from 16^3 to $n = 256^3$, stride $n* = 2$ for both the FFTW and CUDA FFT (CUFFT) implementations on HYAK. Let each n be measured *ntrial* times and plot the average performance for each case versus n, $ntrial \geq 3$. Submit your performance plot and C++ test code. Your plot will have 'flops' on the y-axis (or some appropriate unit of FLOPs) and the dimension of the cubic lattices (n) on the x-axis. You will need to estimate the operation count of computing the derivative using FFT on a lattice.
11. (+5EC) **Root finding**. Write a C++ function that implements a Newton or bisection iteration to estimate the *real* roots to a polynomial equation. Your code should accept the degree of the polynomial, the coefficients, and the domain from the command line (as below). Submit your test code. I will run it against a couple test polynomials of degree 3 and 4 that have simple real roots.

```
Enter the degree of the polynomial: 3
Enter coefficient 3: *
Enter coefficient 2: *
Enter coefficient 1: *
Enter coefficient 0: *
Enter the start of the domain: 0
Enter the end of the domain: 10
Roots found:
-0.*****
1.**
3.*****
```