AMATH 483 / 583 - Final

Due 5pm Friday June 9

June 9, 2023

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

$$
\begin{aligned}
\mathcal{F}\{f(x)\}(k) = F(k) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} e^{ikx} dx \\
&= \frac{1}{2\pi\sigma} \int_{-\infty}^{\infty} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2 + ikx\right\} dx \\
&= \frac{1}{2\pi\sigma} \exp\left\{-\frac{\sigma^2 k^2}{2} + ik\mu\right\} \int_{-\infty}^{\infty} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu-i\sigma^2 k)^2\right\} dx \\
&= \frac{1}{2\pi\sigma} \exp\left\{-\frac{\sigma^2 k^2}{2} + ik\mu\right\} \sqrt{2\pi}\sigma \\
&= \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{\sigma^2 k^2}{2} + ik\mu\right\}
\end{aligned}
$$

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

$$
\begin{aligned}
\mathcal{F}\{f(t)\}(\omega) = F(\omega) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \sin(\omega_0 t) e^{i\omega t} dt \\
&= \frac{i}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \sin(\omega_0 t)\sin(\omega t) dt \\
&= \frac{i}{\sqrt{2\pi}} \left[\frac{1}{2}\delta(\omega-\omega_0) - \frac{1}{2}\delta(\omega+\omega_0)\right].
\end{aligned}
$$

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

$$
\begin{aligned}
\mathcal{F}\{f(x)\}(k) = F(k) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-a|x|} e^{ikx} dx \\
&= \frac{1}{\sqrt{2\pi}} \left[\int_{-\infty}^{0} e^{ax+ikx} dx + \int_{0}^{\infty} e^{-ax+ikx} dx\right] \\
&= \frac{1}{\sqrt{2\pi}} \int_{0}^{\infty} e^{-ax} \left[e^{-ikx} + e^{ikx}\right] dx \\
&= \frac{1}{\sqrt{2\pi}} \left[\frac{1}{a+ik} + \frac{1}{a-ik}\right] \\
&= \sqrt{\frac{2}{\pi}} \frac{a}{a^2+k^2}
\end{aligned}
$$

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

Solution.

$$\begin{aligned}\mathcal{F}\{f(t)\}(\omega) = F(\omega) &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\delta(t)e^{i\omega t}dt \\ &= \frac{1}{\sqrt{2\pi}}\end{aligned}$$

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:

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

Solution. For $t > 1$ we have $p \odot q = 0$, while for $t \in [0, 1]$ we have

$$\begin{aligned} p \odot q &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} p^*(\tau)q(t+\tau)d\tau \\ &= \frac{1}{\sqrt{2\pi}}\int_{0}^{1-t} 1\cdot(1-t-\tau)d\tau \\ &= \frac{1}{\sqrt{2\pi}}\left[\tau - t\tau - \frac{1}{2}\tau^2\right]\Bigg|_0^{1-t} \\ &= \frac{1}{\sqrt{2\pi}}\left(1-t-t(1-t)-\frac{1}{2}(1-t)^2\right) \\ &= \frac{1}{2\sqrt{2\pi}} \end{aligned}$$

3. (+5EC) **Autocorrelation**. Aside, periodic functions exhibit pronounced *autocorrelations* 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:

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

Solution. We have

$$p \odot p = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} p^*(\tau)p(t+\tau)d\tau$$

Plugging in our provided function and looking at where $p(t) \neq 0$, we see that can write this as

$$\begin{aligned} p \odot p &= \frac{1}{\sqrt{2\pi}}\int_{0}^{1-t} d\tau \\ &= \frac{1-t}{\sqrt{2\pi}} \end{aligned}$$

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.

Solution. We begin by transforming our equation into k-space. We define

$$\tau(k,t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ikx} T(x,t) dx$$

Then we have $\mathcal{F}\left\{\frac{\partial T}{\partial t}\right\} = \frac{\partial \tau}{\partial t}$ and $\mathcal{F}\left\{\frac{\partial^2 T}{\partial x^2}\right\} = -k^2 \tau(x,t)$. Hence our transformed equation becomes

$$\frac{\partial \tau}{\partial t} = -\kappa k^2 \tau$$

This is a first order ODE whose solution is given by

$$\tau(k,t) = A e^{-\kappa k^2 t}$$

Let $T_0(x) = T(x,0)$ be our initial condition, and let $\mathcal{F}\{T_0(x)\} = \hat{T}_0(k)$ be the Fourier transform of our initial condition. Using this, we can solve for our solution coefficient A and write

$$\tau(k,t) = \hat{T}_0(k) e^{-\kappa k^2 t}$$

To find our solution in position space in terms of the original variables we take the inverse Fourier transform. Our solution is then given by

$$T(x,t) = \mathcal{F}^{-1}\{\tau(k,t)\} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{T}_0(k) e^{-\kappa k^2 t - ikx} dk$$

b Find the temperature in the rod given initial conditions $\kappa = 10^3 \frac{m^2}{s}$ and

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

Solution. We are given $T_0(x) = T(x,0)$ as defined above. We begin by Fourier transforming this initial condition to find $\hat{T}_0(k)$.

$$\begin{aligned} \hat{T}_0(k) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} T_0(x) e^{ikx} dx \\ &= \frac{100}{\sqrt{2\pi}} \int_{-1}^{1} e^{ikx} dx = \frac{100}{\sqrt{2\pi}} \frac{e^{ikx}}{ik}\bigg|_{-1}^{1} \\ &= 100\sqrt{\frac{2}{\pi}} \frac{\sin(k)}{k} \end{aligned}$$

Having found $\hat{T}_0(k)$, we can plug this in to our inverse Fourier transform expression derived in part (a) to recover our solution $T(x,t)$.

$$T(x,t) = \frac{100}{\pi} \int_{-\infty}^{\infty} \frac{\sin(k)}{k} e^{-\kappa k^2 t - ikx} dk$$

5. (+20) **Compare OpenBLAS to CUBLAS on HYAK**. Measure and plot the performance of double precision matrix multiply ($\alpha AB + \beta C \to C$) 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* ≥ 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* ≥ 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* ≥ 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.*****
```