**ECE 4822: Engineering Computation IV**

**Homework No. 1: Linear Algebra and DSP in C++**

**Goal:** Write code that does some useful engineering calculations in pure C++ to serve as a baseline for future homework assignments.

**Description:**

As mentioned in the syllabus, create a directory called *ece\_4822/homework* in your home directory. In this directory, create subdirectories *hw\_01/p01* and *hw\_01/p02* that contain your solutions to the tasks below. Use make files and generate executables named *p01.exe* and *p02.exe*. Do all computations using 32-bit floating-point numbers.

Note that for this assignment, the C++ you are writing is really C code. You do not have to create classes, etc. Your main program should do the looping for *niter* described below, but your actual calculation should be done using a function call to a function you write (do not use library functions yet).

1. Write a C++ program, *p01.cc*, that has the following interface:

*p01.exe nrows ncols niter*

where:

*nrows*: the number of rows in the first matrix (and the number of columns in the second matrix)

*ncols*: the number of columns in the first matrix (and the number of rows in the second matrix)

*niter*: the number of iterations

This program should loop for *niter* times and do the following:

* generate a first matrix of random values of dimension *nrows x ncols*,
* generate a second matrix of random values of dimension *ncols x nrows*,
* multiply the matrices and print the result in a user-friendly format.

Matrix multiplication should be done in a function defined as:

*bool mmult(float\* mat3, float\* mat1, float\* mat2, long nrows, long ncols)*

where $mat3 = mat1 x mat2$. You can allocate memory in the main program before you start the iterations. We do not want to mess with memory allocation just yet.

Compile this program using gcc with “-O2” and time it for $10,000$ iterations using the Unix time command (or alternately valgrind). Evaluate the amount of CPU time used as a function of $N=nrows = ncols=[1:10,000]$ (essentially plot CPU time as a function of N for the case of square matrices).

Your implementation of matrix multiplication should be basic – nested for loops and direct array indexing (no pointers). Do not get fancy – just keep it simple and slow ☺

1. An autocorrelation function is defined as:

$$R\left[k\right]= \sum\_{n=0}^{N-1}x\left[n\right]x\left[n+k\right],    k=0, 1, …, K.$$

Write a program, *p02.cc*, that has the following interface:

*p02.exe N K niter*

where:

*N*: the number of data points in the signal $x[n]$

*K*: the number of samples of the autocorrelation function

*niter*: the number of iterations

This program should loop for *niter* times and do the following:

* generate *N* random values for x[n] in the range [-1,1],
* compute the autocorrelation function,
* display the values of the autocorrelation function for each iteration.

The autocorrelation computation should be done in a function defined as:

*bool autocor(float\* R, float\* x, long N, long K)*

where $R$ contains the output autocorrelation function. You can allocate memory in the main program before you start the iterations. Again, we do not want to mess with memory allocation just yet.

Debug this by generating a sinewave and making sure the autocorrelation function peaks at the proper lag value (*k*).

Compile this program using gcc with “-O2” and time it for $10,000$ iterations. Plot the amount of CPU time used as a function of $N= [10\*K:1,000\*K]$ and $K = 1000$.

For both problems, plot the results and save the plot to a jpg file named *p01.jpg* and *p02.jpg* respectively in a directory:

*$HOME/ece\_4822/homework/hw\_01/plots*

Plot the results on a log-log scale where the axes are labeled in original values. See this example for more details on this:

*https://www.isip.piconepress.com/courses/temple/ece\_4822/resources/templates/20211016\_logplots.xlsx*

Analyze the computational complexity of the algorithm with respect to the key parameters such as the number of data points and demonstrate that your plots are consistent with this analysis. This is the most important part of this assignment.

Send your results for p01 for one iteration of a $1000x1000$ matrix multiplication (compute your average time based on $10,000$ iterations) by email to the course instructor for inclusion in the class web page. We will track these benchmarks throughout the semester. We will focus on improving these numbers over time.

Finally, be sure to search the class listserv archive for extensive discussions on this topic.