**ROB 550: Fall 2019: C and Python Programming Review Assignment**

*This assignment requires that you write and submit C and Python code to Canvas. You are asked to complete this assignment on your own with the help of course instructors but not other students per the course collaboration policy. We will grade your C code with the “gcc” compiler on Linux with so please be sure to use “pure C” code per the review handout and examples – do not rely on C++ functions or classes.*

*Submit your solutions to Canvas – make sure you “submit” your assignment on Canvas once, after you upload your archive of source code files in the directory structure required. If you accidentally submit too soon send the course instructor an email – we can offer you the ability to resubmit. If you upload files but do not submit it will appear to us that you never submitted anything so be sure to “submit” on Canvas as well as uploading.*

***Linux Terminal:*** A good review of the common Linux terminal commands is [here](http://linuxcommand.org/lc3_learning_the_shell.php).

***Review of C / C++:*** The website [www.cplusplus.com](http://www.cplusplus.com) among others provides a step-by-step review. Also please see the posted ROB550 programming review notes. This document has notes on compiling with gcc in a Linux environment. Additional programming examples are also available in a tar file in this folder.

***Review of Python:*** We will be using Python 2.7 extensively in the class. There are many tutorials online to help learn Python. Code Academy has a step-by-step tutorial [here.](http://www.codecademy.com/tracks/python) This tutorial is meant for beginners, which means it might be a bit slow for experienced programmers, but, at the same time, this also means that it is easy to follow along without getting lost. For more in-depth details relating to Python 2.7, there is a lot of information on Python.org’s [documentation index](https://docs.python.org/2/index.html), including a link to a tutorial.

Numerical Python (NumPy) is a package that allows you to do scientific computing with Python. Many of you may be familiar with using MATLAB from ENGR101 or other classes.  [Here](http://wiki.scipy.org/NumPy_for_Matlab_Users) is an in-depth NumPy vs. MATLAB comparison overview.  [This is a second source](http://mathesaurus.sourceforge.net/matlab-python-xref.pdf) specifically narrowing the comparison down to commands. To get MATLAB-like plotting functionality, we may use Matplotlib, a plotting library for Python that works well with numpy.  [This website](http://www.loria.fr/~rougier/teaching/matplotlib/) has many examples, as well as links to tutorials and other information regarding matplotlib.​ There is also [official documentation](http://matplotlib.org) available.

For specific labs, we will use other tools from [SciPy](http://www.scipy.org/), which already includes both numpy and matplotlib - but you won’t need to know them unless we mention them.

1. **Test Case Generator:** Write and submit to Canvas a Python program testGen.py which generates a random matrix in CSV file format ([official specification](https://www.ietf.org/rfc/rfc4180.txt)). Entries should be normally distributed (hint: look at the numpy.random module). The program should get matrix dimensions from the command-line. These are provided as two integers <rows> and <columns>. The resulting matrix should be emitted to standard output. Be sure to produce an error message to standard error, and return an error status, if the command-line is invalid. You should use [redirection pipes](http://linuxcommand.org/lc3_lts0070.php) (‘>’, ‘<’ and ‘|’) to then save the terminal output to a file.

Example output:

|  |
| --- |
| $ python testGen.py 3 3 0.743185580615, -0.661983636154, -0.162464115428 -2.88839229747, 1.16262315379, -1.06499003415 1.54714645403, 0.275264735597, -0.846846986479 |

|  |
| --- |
| $ python testGen.py 3 fff ERROR: expected two integer arguments |

|  |
| --- |
| $ python testGen.py 3 3 > A.csv // (over)writes to file A.csv |

1. **Matrix Multiplication:** This problem asks you to write three versions of code to perform matrix multiplication **A \* B**: (1) One standalone program in C (submit to Canvas as matmult.c), (2) One standalone program in “pure” Python (no libraries outside the standard Python libraries allowed[[1]](#footnote-0)) (submit to Canvas as matmult\_pure.py), and (3) One Python implementation using numpy (submit to Canvas as matmult\_npy.py).[[2]](#footnote-1) Undoubtedly, your Python code will be simpler, highlighting an important way that Python adds value. However, you will find that a fast pure Python implementation is difficult to produce; most likely your pure Python implementation will take substantial time to multiply two 1000x1000 matrices.

Each version of your code should be a standalone program (C) / script (Python) with the names specified above. Each code should take as input arguments the number of rows in **A**, the number of columns in **A**, the number of rows in **B**, and the number of columns in **B**. The C code must be invoked with:

|  |
| --- |
| $ ./matmult 4 3 3 5 |

to multiply a 4x3 **A** matrix by a 3x5 **B** matrix which generates a 4x5 matrix result. The Python code would be equivalently called with

|  |
| --- |
| $ python matmult\_pure.py 4 3 3 5 $ python matmult\_npy.py 4 3 3 5 |

The filenames containing the input matrices to multiply are A.csv and B.csv, respectively and must be hard coded. Note that Linux is case-sensitive. Note that your code should work for any compatible positive sizes including 1 and should check for inappropriate size values (e.g., negative) or non-matched matrix dimensions. Sensible error messages should be printed to stderr, and programs should exit with an error status. We will use the test case generator from the previous problem to produce input matrix files.

Compute the product of **A** and **B** and write the resulting matrix to the file C.csv in CSV format.

Use timing functions around the actual matrix multiplication to determine the speed of the operation and print this information to the screen in each code. For C, you can use function gettimeofday(). For Python you can use time.time(). See below for example timestamp functions.

Test your code on small matrices / vectors to ensure it works. Feel free to use Matlab or any other application you’d like to make sure that your results are correct.

Now we ask that you benchmark your code using time functions on matrices of several different sizes:

1. **A** and **B** are 10x10 matrices;
2. **A** and **B** are 100x100 matrices;
3. **A** and **B** are 1000x1000 matrices;

Comment on your results in the README.txt file (2-3 sentences are sufficient).

# Useful functions

A function to read in CSV files to a stacked array in C:

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  ...  /\*\*  \* Allocates a matrix which must be free'd by the calling free()  \*/ double\* readMatrixFromFile(char\* fileName, int height, int width) {  FILE\* fp = fopen(fileName, "r");  if (fp == NULL) {  fprintf(stderr, "Can't open %s.\n", fileName);  return NULL;  }  double val;  double\* M = (double\*) malloc(height \* width \* sizeof(double));  for(int i = 0; i < height; i++) {  for(int j = 0; j < width; j++) {  if (fscanf(fp, " %lf", &val) != 1) {  fprintf(stderr, "Couldn't read value.\n");  return NULL;  }  // Discard the comma without checking.  fgetc(fp);  M[i \* width + j] = val;  }  }  fclose(fp);  return M; } |

A function to write a stacked array to a CSV file in C:

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  ...  int writeMatrixToFile(char\* fileName, double\* matrix, int height, int width) {  FILE\* fp = fopen(fileName, "w");  if (fp == NULL) {  return 1;  }    for (int i = 0; i < height; i++) {  for (int j = 0; j < width; j++) {  if (j > 0) {  fputc(',', fp);  }  fprintf(fp, "%lf", matrix[i\*width +j]);  }  fputs("\r\n", fp);  }  fclose(fp);  return 0; } |

A microsecond timestamp function in C:

|  |
| --- |
| #include <sys/time.h> ...  int64\_t utime\_now (void){  struct timeval tv;  gettimeofday (&tv, NULL);  return (int64\_t) tv.tv\_sec \* 1000000 + tv.tv\_usec; } |

A microsecond timestamp function in Python:

|  |
| --- |
| import time ...  def utime\_now():  return int(time.time()\*1E6) |

**Submission Guidelines**

Archive (as a .zip file) and submit your assignment to Canvas with the following structure:

|  |
| --- |
| <shortname>-intro\_assignment.zip  ↳<shortname>  ↳ testGen.py - matrix generator  ↳ matmult.c - c implementation  ↳ matmult\_pure.py - pure python implementation  ↳ matmult\_npy.py - numpy implementation  ↳ README.txt - benchmarks & commentary |
|  |

1. Specifically, use only the libraries found here: <https://docs.python.org/2/library/index.html> The idea is to do the math by yourself in the code - not with help of pre-existing library functions. [↑](#footnote-ref-0)
2. You may want to use a stacked array format in C to improve code efficiency relative to a two-dimensional matrix representation. For pure Python, you can use a list of lists to represent the matrices. [↑](#footnote-ref-1)