Due on Friday, October 9th, 2020.

− − − − − − − − − − − − − − − ⋆ − − − − − − − − − − − − − −

1. python, machine representation of numbers, etc

   (a) Explain the difference(s) between a python list and a numpy array.

   (b) The python math library and the numpy library both contain various common functions.

   What's the difference between math.sin() and numpy.sin()?

   (What arguments can the latter take that the former cannot?)

   (c) Look up array 'slicing' if you don't know it well already. What is the output of the code

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
```

   and why?

   (d) (For yourself; no need to submit this) Make sure you can work out what the results would be if the last line of the previous code is replaced by one of the following:

```
print(arr[:, -2])
print(arr[0, 3:])
print(arr[0, 0:-2])
print(arr[1, ::2])
print(arr[0, 2:0:-1])
```

   Slicing is an essential technique, so if you don't know this well, you should figure this out now.

   (e) Computer languages nowadays often use 64 bits to store floating-point numbers (i.e., they use double precision). Explain how these 64 bits are used. (exponent, mantissa, sign)

   From your description, work out the value of the largest floating point number that can be represented in this format.

2. Matrix decompositions

   (a) Explain the difference between LU decompositions and Cholesky decompositions.

   (b) What is the QR decompostion?

   (c) The real square matrix A has the QR decomposition

$$A = Q_1 R_1$$

   Show that the matrix $R_1 Q_1$ has the same eigenvalues as the matrix A.

3. Memory (RAM) limitations:

   (a) Write python code to create a $N \times N$ square matrix filled with random floating-point numbers uniformly distributed between 0 and 1, and save the matrix in a variable.

   Run your code for $N = 10, 20, 50, 100, 200, 500, 1000, ...$, or another sequence of sizes if you think appropriate.

   In each case, find out how much memory (RAM) your program is using. (On a unix system, this could be done using the top command. On a non-unix system, there should be equivalent tools to monitor memory usage.)

   Make sure to stop well before you use 100% of your machine's RAM, because your computer will probably freeze of crash at that point.

   Plot the memory usage versus matrix size.

   (b) Based on your data: what maximum matrix sizes (maximum N) can be comfortably loaded on memory, on a typical modern-day desktop/laptop machine?

   (c) Try calculating the memory taken by a matrix of maximum size, using the fact that each floating point number uses 64 bits of memory.