Matrix Inversion

You can invert a matrix A by creating the $n \times 2n$ matrix $[A \quad I]$. This means you stack A right next to the $n \times n$ identity matrix. Then perform Gaussian Elimination on A just like you were solving a system of equations. You will end up with the matrix $[I \quad A^{-1}]$. Sounds easy. But matrix inversion is highly unstable numerically. You will investigate in this lab.

1. The Hilbert matrix $a_{ij} = 1/(i+j+1)$ is notoriously unstable. (Read about it online; note that our definition uses "+1" because python and math are different sometimes.)
2. Implement matrix inversion and check it on some small ~~Hilbert matrices~~ **random real matrices**. (Check by verifying $AA^{-1} = I$. This will *not* be exact so account for rounding errors).
3. Make a plot of matrix size vs. error. Error is defined as the Frobenius norm of the matrix $(AA^{-1} - I)$ so $err = ||(AA^{-1} - I)||_F$. (It's trivial to do in numpy so look it up!)
4. Implement GEPP (Gaussian elimination with partial pivoting) and make a similar plot to part 2. You should see the errors decrease with GEPP.
5. Finally, plot the error in the inherent `np.linalg.inv` function and compare it to yours.

You can seen an example of GEPP here. Note they do NOT make the diagonal all 1's like we do – but you can keep doing it our way. The main idea is the pivoting.

I wrote this after a short investigation of Hilbert matrices which are known to be poorly conditioned (the condition number grows quite predictably, if you want to look it up). I assumed that GEPP would behave poorly in direct correlation to the condition number but it does *not*. Random matrices provide a much cleaner pattern.