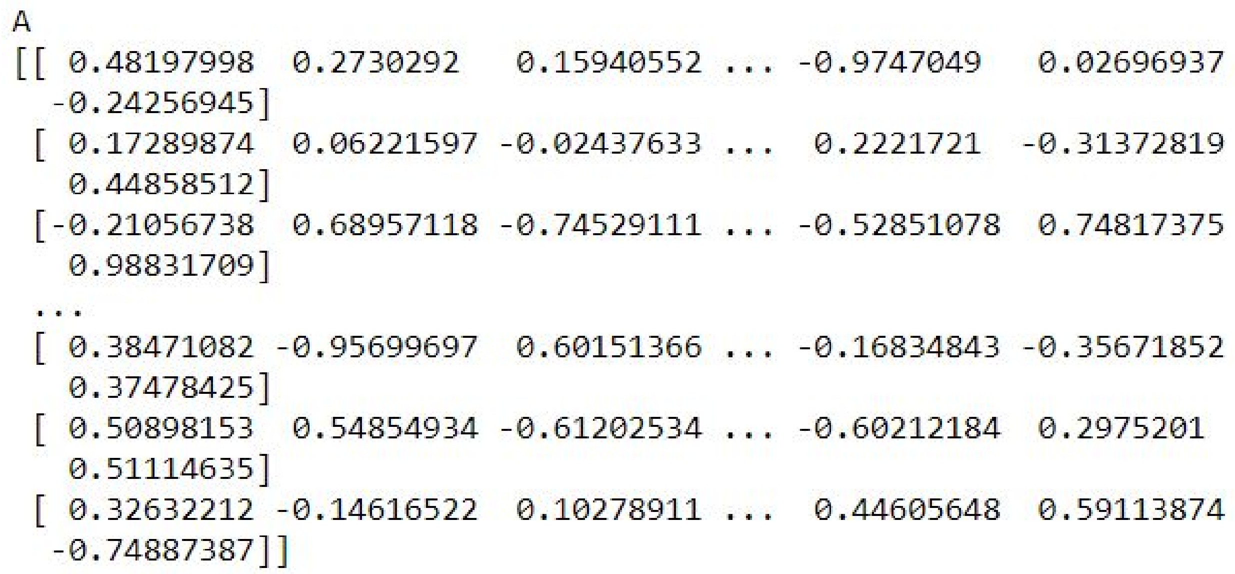
AMS326 Final Report

Megan Tan, 110598264

For this final, problems 1, 2, and 4 were done in separate iPython notebook.

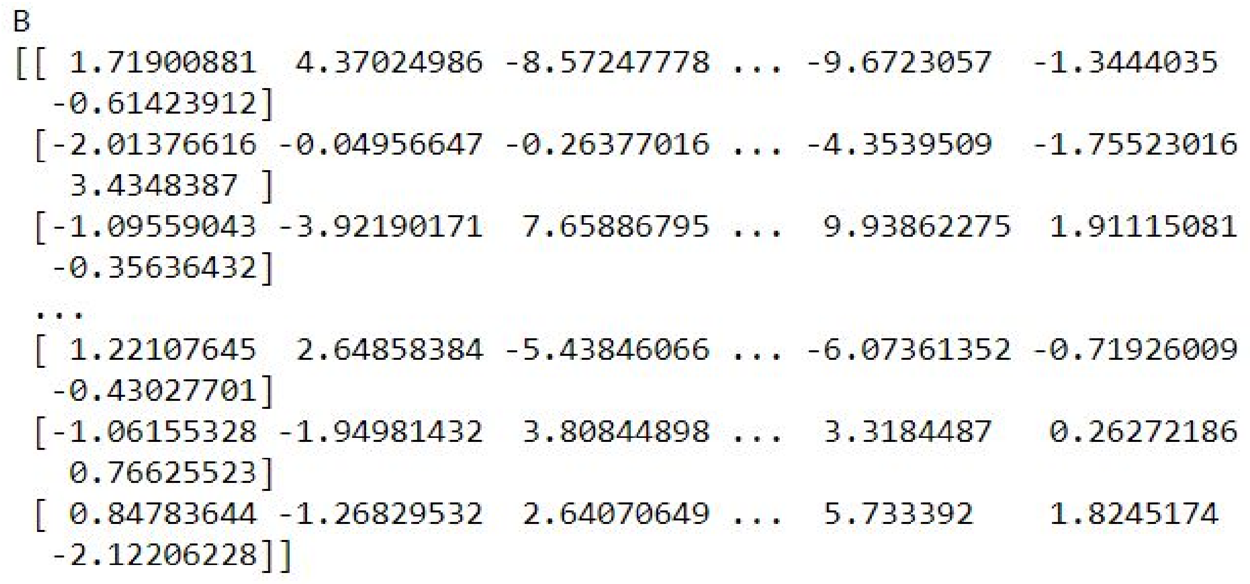
Problem 1 Report

For this problem, an empty matrix called A of size 32x32 was generated. A random number between -1 and 1 was then generated 32x32 times, and each value in the matrix was updated with that random number. The inverse of A, called B, was then calculated using numpy's inbuilt inverse function. The dot product of A and B was then calculated using numpy's dot product function. The partial matrices of A, B and A*B are shown below.



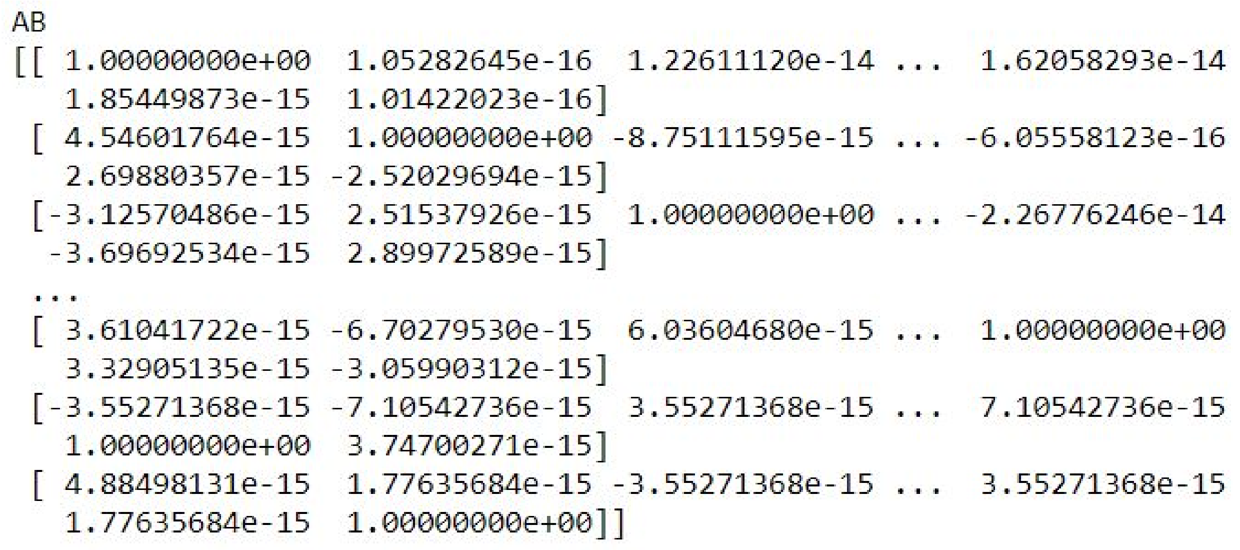
```
A
[[ 0.48197998  0.2730292   0.15940552 ... -0.9747049   0.02696937
  -0.24256945]
 [ 0.17289874  0.06221597 -0.02437633 ...  0.2221721  -0.31372819
   0.44858512]
 [-0.21056738  0.68957118 -0.74529111 ... -0.52851078  0.74817375
   0.98831709]
 ...
 [ 0.38471082 -0.95699697  0.60151366 ... -0.16834843 -0.35671852
   0.37478425]
 [ 0.50898153  0.54854934 -0.61202534 ... -0.60212184  0.2975201
   0.51114635]
 [ 0.32632212 -0.14616522  0.10278911 ...  0.44605648  0.59113874
  -0.74887387]]
```

Figure 1: Matrix A



```
B
[[ 1.71900881  4.37024986 -8.57247778 ... -9.6723057  -1.3444035
  -0.61423912]
 [-2.01376616 -0.04956647 -0.26377016 ... -4.3539509  -1.75523016
   3.4348387 ]
 [-1.09559043 -3.92190171  7.65886795 ...  9.93862275  1.91115081
  -0.35636432]
 ...
 [ 1.22107645  2.64858384 -5.43846066 ... -6.07361352 -0.71926009
  -0.43027701]
 [-1.06155328 -1.94981432  3.80844898 ...  3.3184487   0.26272186
   0.76625523]
 [ 0.84783644 -1.26829532  2.64070649 ...  5.733392    1.8245174
  -2.12206228]]
```

Figure 2: Matrix B, the inverse of A



```
AB
[[ 1.00000000e+00  1.05282645e-16  1.22611120e-14 ...  1.62058293e-14
   1.85449873e-15  1.01422023e-16]
 [ 4.54601764e-15  1.00000000e+00 -8.75111595e-15 ... -6.05558123e-16
   2.69880357e-15 -2.52029694e-15]
 [-3.12570486e-15  2.51537926e-15  1.00000000e+00 ... -2.26776246e-14
  -3.69692534e-15  2.89972589e-15]
 ...
 [ 3.61041722e-15 -6.70279530e-15  6.03604680e-15 ...  1.00000000e+00
   3.32905135e-15 -3.05990312e-15]
 [-3.55271368e-15 -7.10542736e-15  3.55271368e-15 ...  7.10542736e-15
   1.00000000e+00  3.74700271e-15]
 [ 4.88498131e-15  1.77635684e-15 -3.55271368e-15 ...  3.55271368e-15
   1.77635684e-15  1.00000000e+00]]
```

*Figure 3: Matrix A*B*

Problem 2 Report

For this problem, the X coordinate is the first number in each given tuple in the question, and the Y coordinate is the second number in each given tuple in the question. The Z coordinate, which is the height of the trees is then generated with a random normal distribution with mean 8.888 and standard deviation 1.949. A fourth list with the x,y and z coordinate of each tree is created so that the trees can be numbered from 0 to 15. Each tree corresponds to the index in that list. The x,y, and z coordinates, as well as the list of trees are shown in the screenshot below.

Figure 4: The x, y, and z coordinates of the trees, and the list of each tree with its corresponding x,y, and z coordinate.

A greedy algorithm is then used to compute the shortest distance needed to travel to each tree. The greedy algorithm works by starting at the first tree as the current tree, and then traversing through the list of trees by choosing the tree that is closest to the current tree and going to that tree. This is done until all trees are visited. The list of tree coordinates that give the shortest distance is shown below.

Figure 5: List of tree coordinates for the shortest path starting at the first tree

The trees should be visited, starting from tree 0, in the order shown below to get the shortest distance.

Figure 6: Indices of trees to visit to get the shortest distance

The length of wire used, which is the total distance needed to visit each tree in the order shown above is 32.665355104763904. It is also shown in the screenshot below.

Figure 7: Optimal wire length

Because the height of the trees are randomly generated, each run of the program may yield results different from the ones shown above.

Problem 4 Report
For this problem, uniformly distributed random numbers x,y, and z in the range of [-3,3],[-4,4],[-5,5] respectively were generated for each given N value. For each N value, this was done M=60 times.

For each time in N, the point generated was checked if it was in the sphere with the equation $(x/3)^2 + (y/4)^4 + (z/5)^2 = 1$. The total number of points that was in the sphere, N_{in}, was recorded. To get the estimated volume V of the sphere, the following equation was used. $V = N_{in}/N * 6 * 8 * 10$.

The standard deviation of each M experiments was then computed using the formula $\sigma(N) = (1/M * \sum_{i=1}^{M}(V(N)_i - \overline{V})^2)^{.5}$, where $\overline{V}$ is the volume of the ellipsoid found analytically with the formula $\overline{V} = 4\pi/3 * 3 * 4 * 5$.

The standard deviations, rounded to 4 decimals for neatness, are shown in the table below.

N	10^4	10^5	10^6	10^7
$\sigma(N)$	2.6650	0.8205	0.2902	0.0689

It is also shown in the printed output below.

The plot of the standard deviations vs the N is shown below.

Figure 8: Graph of standard deviation (error) vs number of iterations (N)

Note that because this problem uses randomly generated numbers, each run of the program will produce different results. However, the trend of decreasing standard deviations should be the same.

<u>Appendix: Python Libraries Used</u>

1. Numpy
2. Numpy.linalg
3. Matplotlib.pyplot
4. Math