***Week 1***

General constant convention:

* : *dimensionality of full problem space*
* : *dimensionality of problem vectors*
* : *number of vectors in problem space*

Problem definition: searching for intersections/nearest points of approaches of vectors in n dimensional space.

* Further generalisation to increase scope – “vectors” (at least in my mind) tends to suggest 2D lines: i.e, – can be expanded to m dimensional vectors up to n dimensions
  + Where is some initial starting point, are directions, are scalars.
* Further generalisation to include more than 2 vectors
  + Points where all vectors intersect? Points where more than/less than vectors intersect? Points where there exists an intersection? Etc.

Results in all methods having three primary performance metrics to be compared: How well it scales with:

1. Dimensionality of problem space
2. Dimensionality of problem vectors
3. Number of vectors

In addition, further properties of each method may include:

1. Does it identify the existence/lack of intersections?
2. Will it find the closest point of approach should an intersection not exist?
3. Do all vectors need to have the same dimensionality?
4. Will it identify all possible intersections?

Example method: rudimentary GCSE method for finding intersections of 2 2D vectors in 3D space.

* Identifies the existence/lack of intersections.
* Will not find the closest point of approach.

1. Formulate the two vector equations as a set of simultaneous equations, with 3 equations (one per dimension ) and 2 unknowns (one per dimension )
2. Arbitrarily choosing 2 of the 3 equations, formulate as a 2x2 matrix and solve the inversion, resulting in some values for .
3. Confirm values are consistent with the last equation.

Method easily extends to higher dimensional problem spaces. The intuitive process taken is that an intersection is found from a projection of one of the dimensional axis (as selected “arbitrarily” when picking your two out of equations), solving the simple intersection between two lines in 2D space, and confirming that the calculated scalar coefficients are consistent along every other dimension. Each new dimension therefore simply adds a single “confirmation” calculation, meaning it scales in with dimensionality. This is also, intuitively, the mathematical limit for how well an algorithm can scale with dimensionality (?).

Scaling in is a bit more complex. A simple first-thought solution would be just to recalculate for every 2-combination possible between all vectors. This results in a scaling of (from the equation for combinations, with samples fixed at 2). There intuitively exists a better way to do this, as it likely results in many redundant/repeated calculations.

Scaling seems to be most complex in . Still yet to figure this out, but it involves calculating the null space of a matrix along with an initial particular solution. The former is intuitively more computationally intensive to calculate, some sources suggest a time complexity of to depending on method and matrix properties. More research needs to be done, but one can likely conclude that this is the primary bottleneck of this method.

Also, the listed complexities are worst case computation costs, which occur at an intersection. A failure to match any of the further dimensions provides a sufficient condition for non-intersection, and can therefore move to the next iteration immediately.

This method therefore performs best in the following conditions:

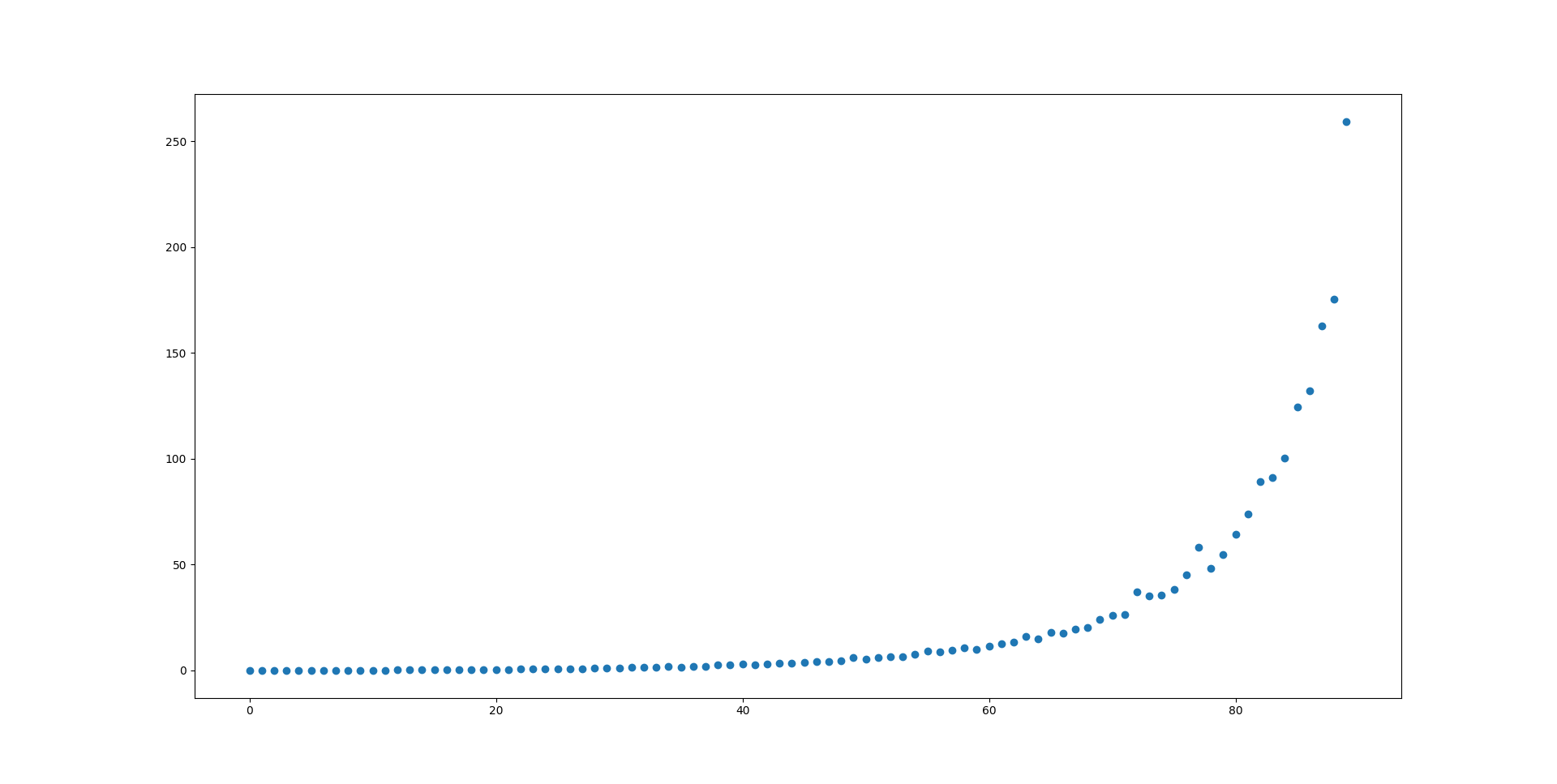
1. Only the intersection point is required. Should the lines not intersect, no nearest-point alternative is necessary.

**TODO:** Code up an implementation to demonstrate this.

Some way of sifting different problems into different categories, and applying different methods to optimally tackle different problems.

Ways of simplifying dimensionality – probably shows up one way or another in the matrix properties.

***Week 6***

Came up with general reasonings as to the performance of the orthogonality-based algorithm on a series of randomly generated high-dimensional vectors. Produced some graphs on performance with dimensionality.

Timeit graph, shows significant scaling with dimensionality (1000 dim space), each x axis point represents 5 dimensions.

Performs best when dimensionality of total space is much larger than dimensionality of vectors (iterative), since random vectors under this config are far more likely to be orthogonal.

Otherwise, scales n^3. Iterations scale similarly.

Could try prove iterative convergence similar to how the Bellman optimality equations are shown to be convergent?