Getting to the Point! The Exact Fitting Problem

Austin Smith

University of Idaho
Department of Mathematics and Statistical Science

Abstract

Suppose you were traveling through a downtown urban sprawl of a large city, aiming to see the most of the city as possible. If you listed out the top twenty storefronts you wanted to see, what straight line would take you to the most number of attractions? Could you always find such a straight line?

Similarly, consider the way a graphing calculator creates a line of best fit, defining "best" by the "least squares approach" (i.e. the linear regression problem). What if instead we fit a line using something called the "exact fitting" method? In this method, the "best" line is the one that intersects the most number of points. What is the most efficient method to calculate this line?

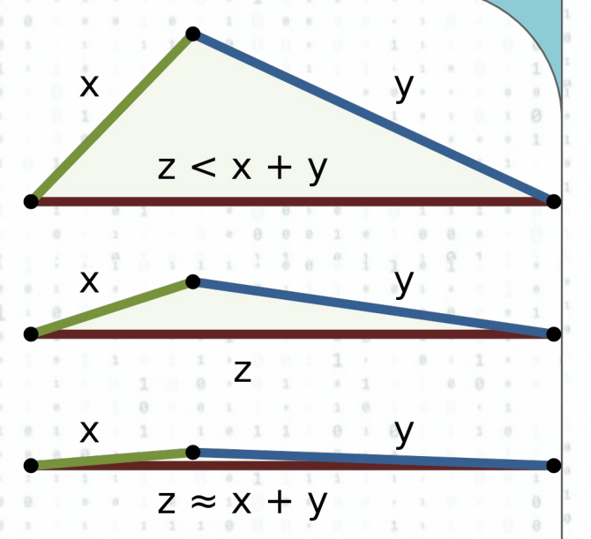
Finding the most efficient algorithm to compute this line is an open question in computational geometry, known as the exact fitting problem. Guibas et al. have previously shown an O(min{N^2log(N), N^2}) time solution in two dimensions.

Here, we examine an alternative approach using properties of determinants to create a new algorithm implemented in the C++ coding language. An estimate for the time efficiency of the algorithm can be conjectured to be O(N^3). Because this problem has widespread implications, finding an efficient algorithm would have positive impacts on numerous fields including coding theory.

Background Information

- Determinants: A way of representing area.
- By the triangle inequality: we can see that three points are collinear if the area between them is zero.

$$Area = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_1 & 1 \end{vmatrix}$$



- Big Oh Notation: It is a way of describing computational time. In general, an algorithm with a smaller Big Oh is more time efficient. Some common examples of Big Oh and their complexities are: $O(1) < O(log[N]) < O(N) < O(N^2) < O(N^3) < O(N^4) < \ldots < O(N!)$ Here, O(N!) is the slowest algorithm.
- Any line in 2 dimensions can be described uniquely by its slope and intercept, so tracking these two characteristics is sufficient to see whether two lines are identical.
- Any two points can be fit by a line, so the minimum answer of the exact fitting problem is always 2. This is known as the trivial solution.
- N Choose 3: Also known as combinations or C(N,3), it is the total number of ways to pick three objects from a set of N, assuming order does not matter. The formula for N Choose 3 can be given by: $\frac{N!}{3! * (N-3)!}$

The Algorithm

```
Input: A set of N points, named S
Output: The maximum number of points on a line
1) Go through all N Choose 3 combinations of points in S:
    1a) If the three points have a determinant of zero (all fit on one line), add them to the list of possibilities
    1b) If the three points do not fit on the same line, disregard this combination
2) For all items in the list, evaluate the slope and intercept of the line
3) For all items in the list, combine those with the same slope and intercept
4) Evaluate the number of points intersected by each line
5) The largest number from step 4 is our answer
```

Snippet of the Code

```cpp
long long unsigned int temp = 0;
for (int i =0; i < N; i++){
    for (int j = i + 1; j < N; j++){
        for (int k = j + 1; k < N; k++){
            if (det == 0){
                combo[temp].xcoords.push_back(xpoints[i]);
                combo[temp].ycoords.push_back(ypoints[i]);

                combo[temp].xcoords.push_back(xpoints[j]);
                combo[temp].ycoords.push_back(ypoints[j]);

                combo[temp].xcoords.push_back(xpoints[k]);
                combo[temp].ycoords.push_back(ypoints[k]);
            }
            else{
                combo.erase(combo.begin() + temp);
                temp--;
            }
            //printvector(combo[temp].xcoords, combo[temp].ycoords);
            temp++;
            if (temp == size){
                //cout<<"Generated combinations"<<endl;
                return;
            }
        }
    }
}
```

This code is step 1 of the algorithm. It will check all combinations of three points to see whether they have a zero determinant. If they do, it adds them to the list of possible points. If they have a zero determinant, it erases the combination.

To view the code in its entirety, scan the QR code below!

Connections to Other Fields

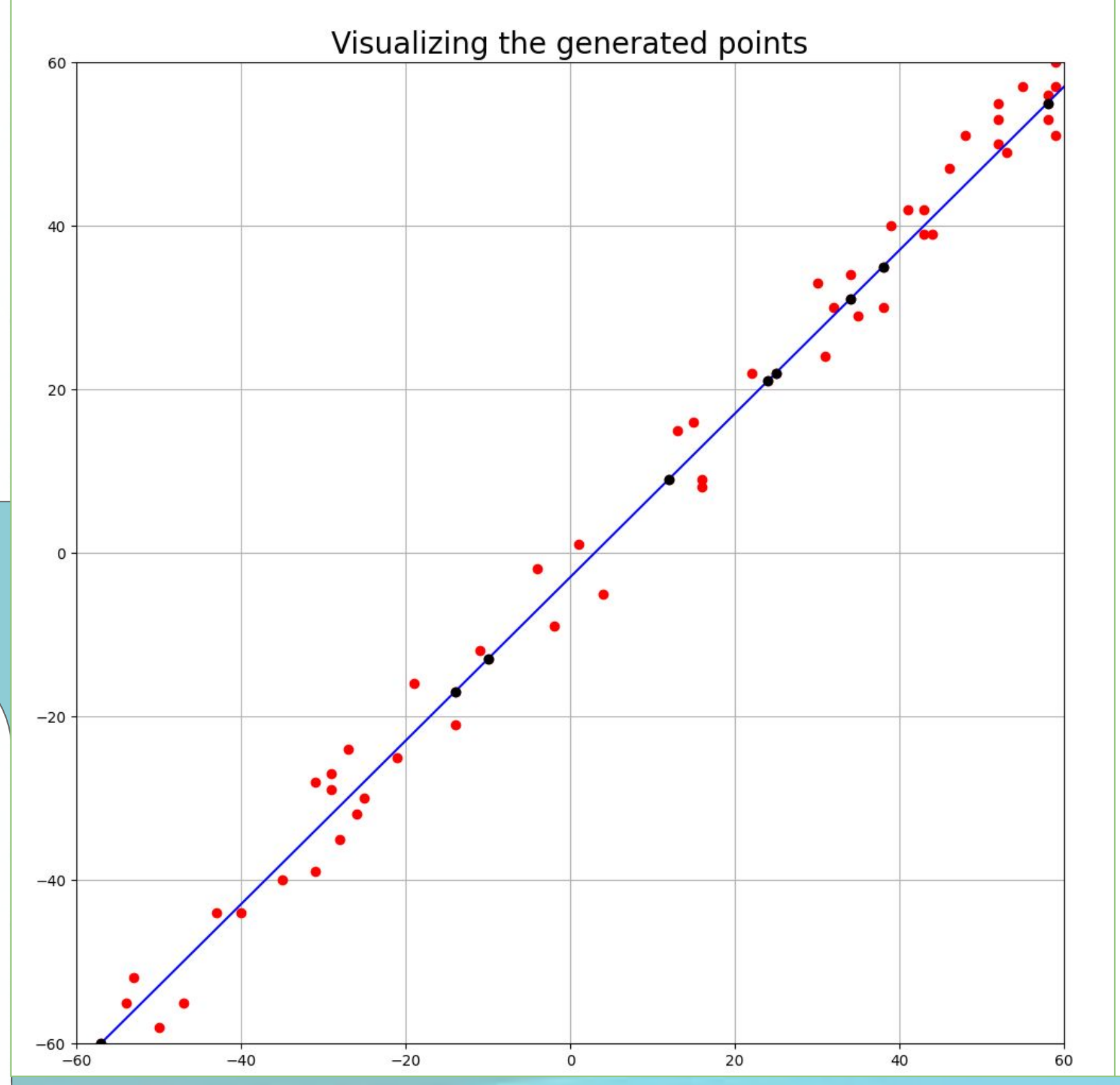
When the Curiosity Rover landed on Mars, it immediately began transmitting its revolutionary findings back to Earth. Along the way, some of the data may have been corrupted from cosmic radiation, fluctuations in power, or even human error. Given the 2.5 billion USD price of the information this rover collected, being able to detect and correct these errors was vital. In other forms of more widely-used data communication, from sending emails to storing family photos on hard drives, corruption detection and correction continues to be of the utmost importance.

The minimum distance of an error correction code C is defined to be the minimum Hamming distance between two distinct codewords of C. Finding the minimum distance of a code is key to error correction efforts in coding theory, as a code with minimum distance M can detect M-1 errors and even correct (M-1)/2 of those errors using a process called nearest neighbor decoding.

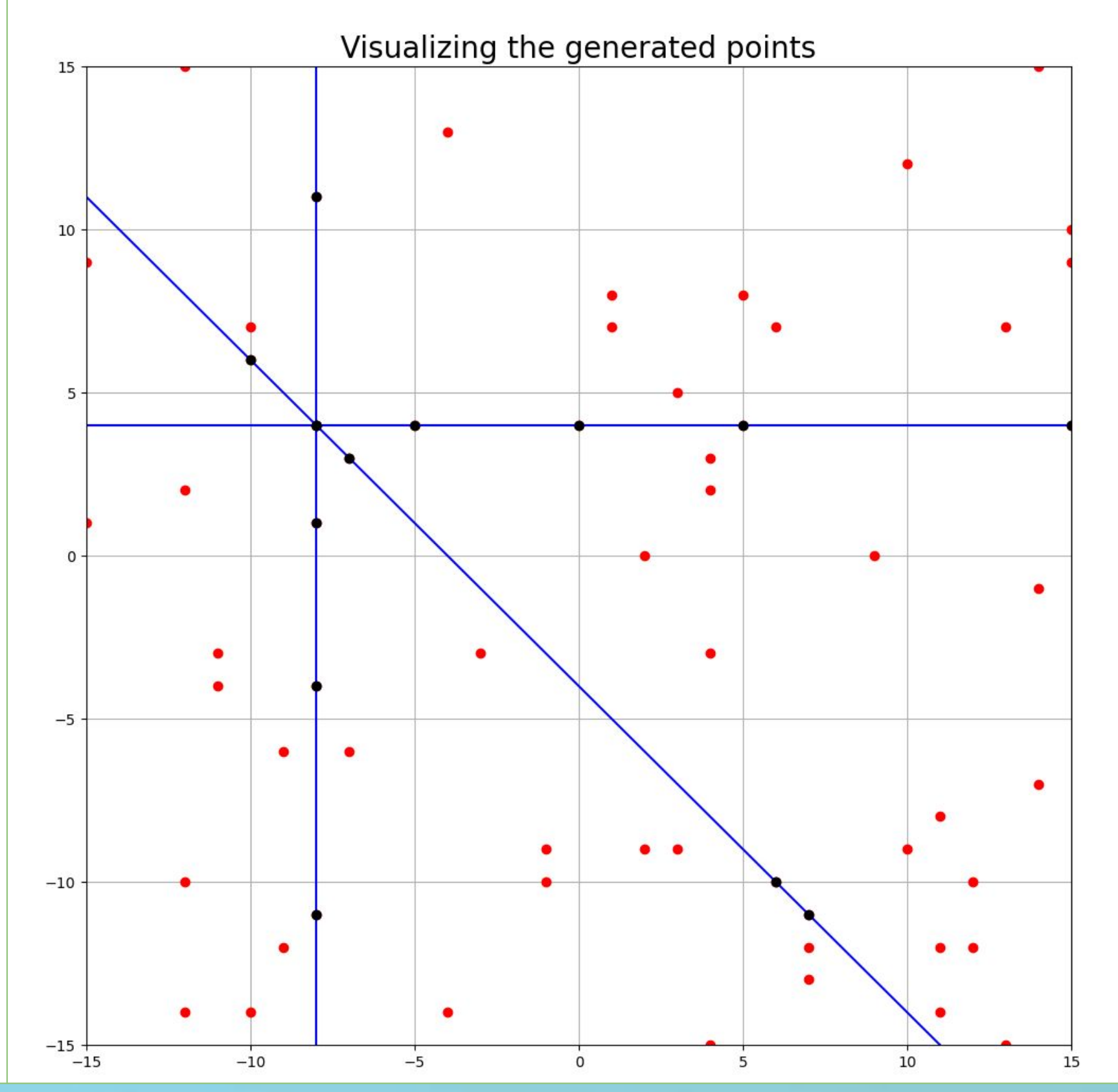
Although seemingly unrelated, finding the maximum number of points that are intersected by a hyperplane is equivalent to finding the minimum distance of a linear code with a generator matrix which has the coordinate of the points as columns.

Examples

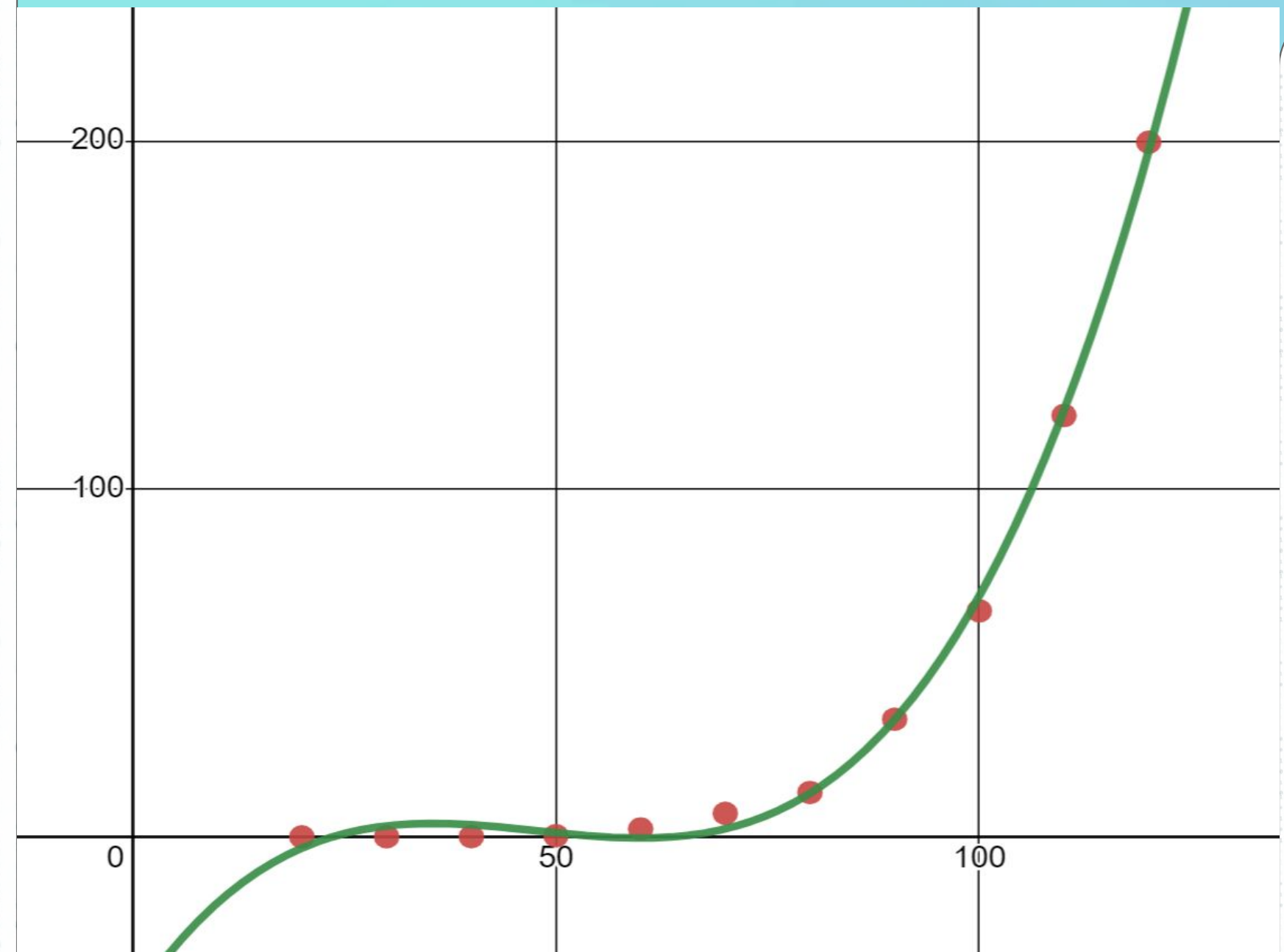
The largest number of collinear points is 9



The largest number of collinear points is 5



Time Estimates



A regression analysis of various sizes of N shows that a cubic polynomial fits the data with a 0.998 R Squared value. This highly indicates that this algorithm likely runs at O(N^3) time.

Future Work

The first obvious extension to this problem is to expand the problem into higher dimensions. Dr. Stefan Tohaneanu and I are currently working on using properties of higher dimensional simplexes to generalize this algorithm into fitting planes in three dimensions. A unique challenge of higher dimensional versions of this problem is that in three dimensions, points may be collinear as well as coplanar, making characterizing a plane by a normal vector and a point especially challenging.

The exact fitting problem also has a surprising amount of connections to other fields besides just geometry and coding theory. Another approach to this problem in two dimensions involves points into a simplicial complex (graph) where vertices are points and edges connect collinear points. Then, instead of finding the largest number of collinear points, the problem shifts to finding the largest simplex in the graph. This is also known as the clique problem or the complete subgraph problem.

There's also interesting potential study into the expected value of collinear points, or even the largest number of points in a grid where there is only the trivial solution. This line of inquiry has connections to the aptly named no-three-in-line problem from chess, which is an open problem with fascinating ties to probability theory and algebra.

Acknowledgements

Thank you to Joseph Miller and Caden Sampsel for their help and tips for navigating C++.

Thank you to Katelyn Taylor for improving the readability of this poster and contributing to the title.

Thank you to Professor Tohaneanu for guiding my research.

Best wishes to all of professors, instructors, teachers, and mentors who have helped me get to where I am now.

References

Alexander Vardy, Algorithmic complexity in coding theory and the minimum distance problem, in: Proc. of the 29th ACM Symposium on Theory of Computing (1997) 92-96.

David Eppstein, Random no-three-in-line sets, in: 11011110 Blog (2018), https://11011110.github.io/blog/.

Karen Brown, Linear codes and error-correction, in: University of Northern Iowa Presidential Scholars Theses (1990 – 2006) 44.

L.J. Guibas, M. Overmars and J.-M. Robert, The exact fitting problem for points, in: Proc. of the 3rd Can. Conf. on Comp. Geom. (1991) 171-174.