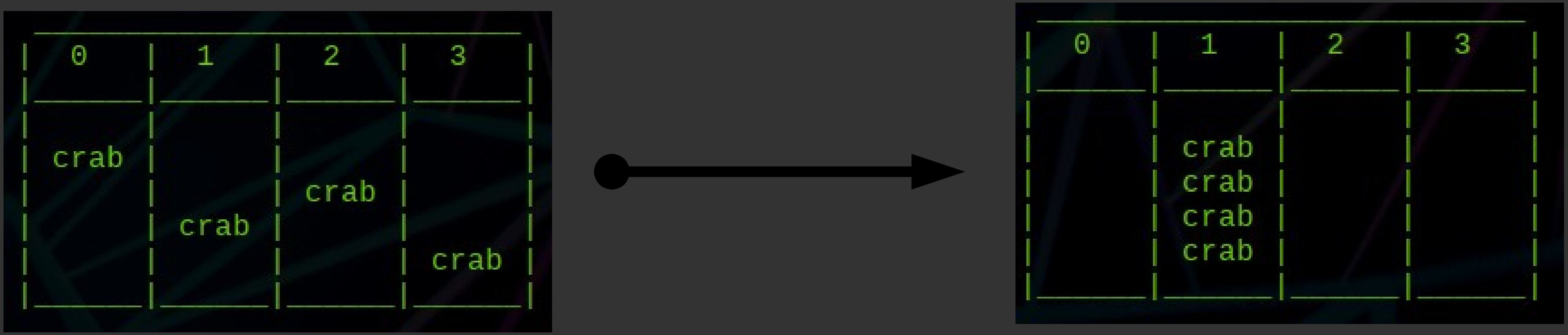
Advent of Code 2021

Day 7 – Part 1

- Problem description: https://adventofcode.com/2021/day/7
- A “swarm” of crabs arranged with various non-negative integer horizontal positions (ex: 0, 7, 3)
- The crabs crabs can move and they burn fuel proportional to the distance traveled (1 unit of fuel to move 1 unit of distance)
- Task: move all crabs to the same horizontal position while minimizing the total amount of fuel spent. Submit the sum of the fuel spent by all crabs.



Day 7 – Part 1

- Proposed consolidation point: $\hat{y}$
- Crab horizontal position: x
- Fuel function for single crab: $f(\hat{y}) = |\hat{y} - x|$
- Fuel function for all crabs: $f(\hat{y}) = \sum_i |\hat{y} - x_i|$

Day 7 – Part 1: Brute Force Technique

- For every horizontal position y between the minimum and maximum crab positions
  - For every crab x
    - Add abs(y – x) to the sum
  - Return the minimum fuel cost.
- Is brute force feasible?
  - 1841 iterations of the outer loop
  - 1000 iterations of the inner loop
- Yes brute force is trivial

```
>>> data.shape
(1000,)
>>> data.min()
0
>>> data.max()
1840
```

Day 7 – Part 1

The End.

Day 7 – Part 1

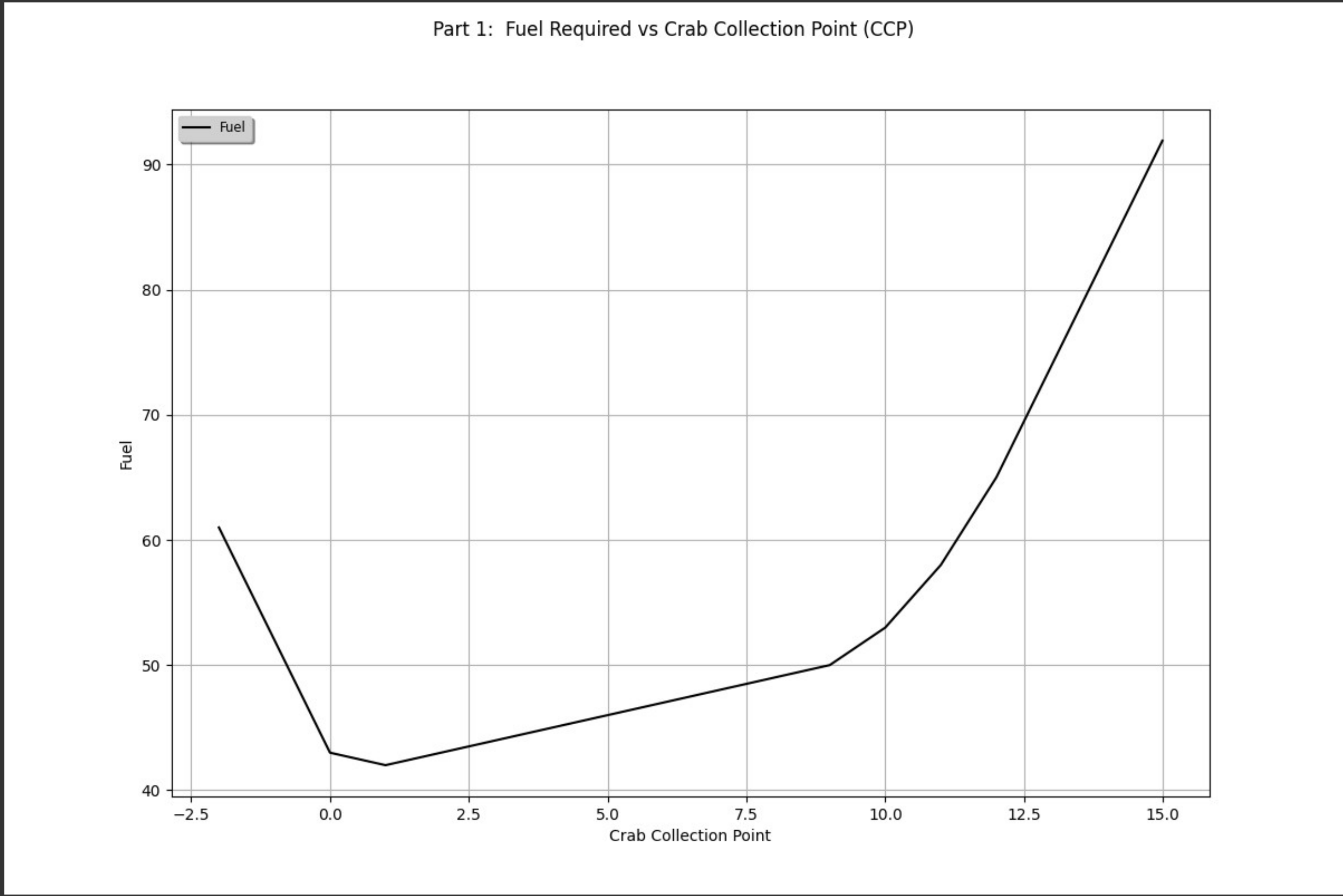
JK

Day 7 – Part 1

Crab positions:

[0, 0, 0, 0, 1, 9, 10, 11, 12]

It’s a convex function, which means it’s easy to minimize



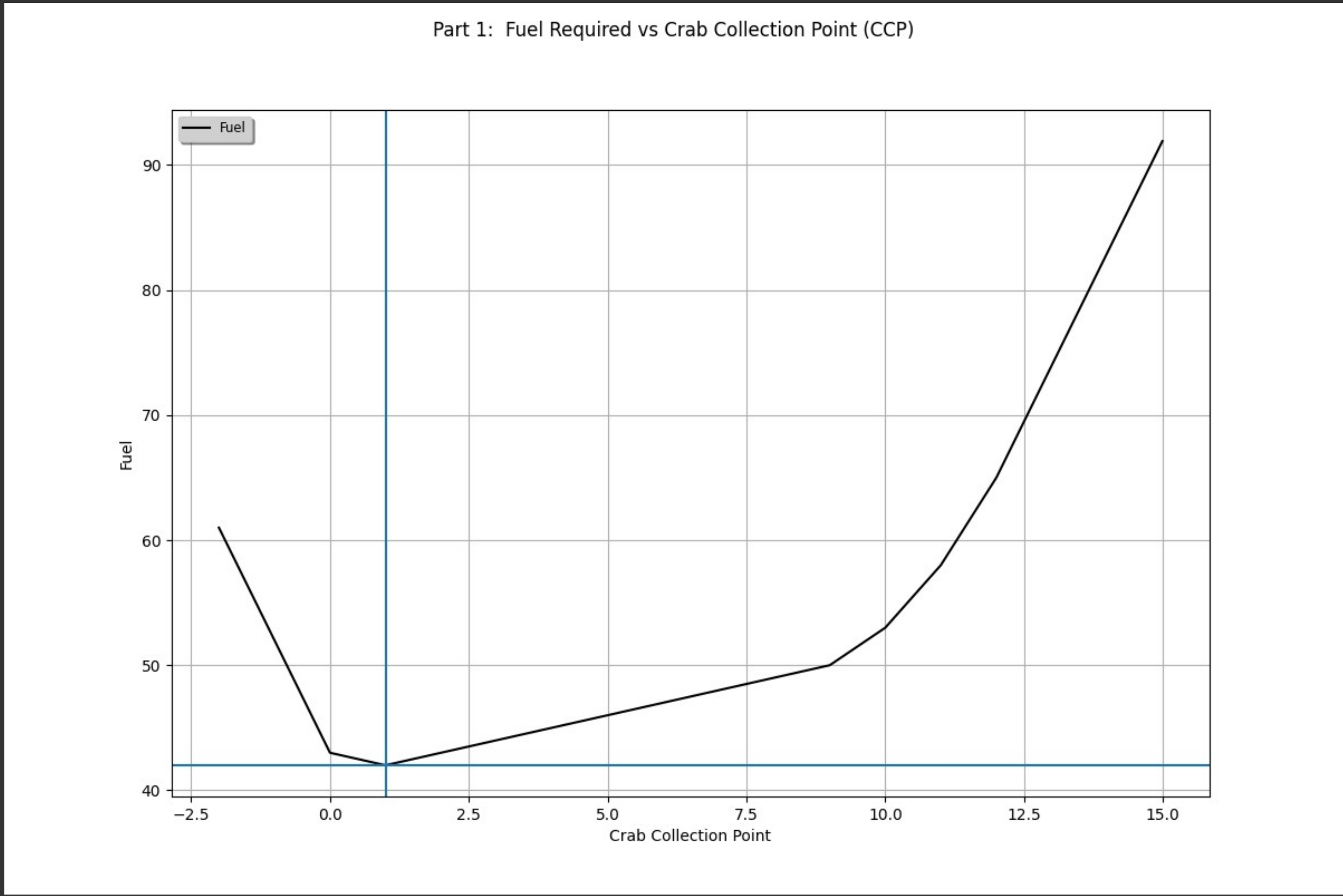
Day 7 – Part 1

Crab positions:

[0, 0, 0, 0, 1, 9, 10, 11, 12]

Minimum fuel cost occurs when the slope of the fuel function is zero*.

*for this function, the slope is technically not defined at this point



Day 7 – Part 1: Math

$X = \{x_0, x_1, ...\}$, initial crab positions

$f(X, \hat{y}) = \sum_i |\hat{y} - x_i|$, fuel cost to move all crabs to position $\hat{y}$

Goal: minimize $f(\hat{y})$ by finding where derivative is 0

Trick question, $f(\hat{y})$ is only piece-wise differentiable.

Equivalent fuel function:

$$f(X, \hat{y}) = \begin{cases} \sum_i \hat{y} - x_i \text{ , where } \hat{y} \geq x_i \\ \sum_i x_i - \hat{y} \text{ , where } x_i > \hat{y} \end{cases}$$

Equivalent fuel derivative:

$$f'(X, \hat{y}) = \begin{cases} \sum_i 1 \text{ , where } \hat{y} \geq x_i \\ \sum_i 1 \text{ , where } x_i > \hat{y} \end{cases}$$

Another way to view the fuel function:

$$f'(X, \hat{y}) = \sum \hat{x}_i \geq \hat{y} - \sum_i \hat{y} < x$$

- In the basic form, the function is only piece-wise differentiable and undefined at 0
- By rearranging, the fuel function derivative minimized when then number of crabs to the right of y is equal to the number of crabs to the left of y
- This central location is the median.

Day 7 – Part 1: Implementation

```python
#!/usr/bin/python3

import numpy as np

with open("input", "r") as in_file:
    data = np.asarray(in_file.read().strip().split(','), np.int64)
    print(np.absolute(data - np.round(np.median(data))).sum())
```

Day 7 – Part 2

- The same crab layout and objective as before, but with a different fuel function.

> As it turns out, crab submarine engines don't burn fuel at a constant rate. Instead, each change of 1 step in horizontal position costs 1 more unit of fuel than the last: the first step costs `1`, the second step costs `2`, the third step costs `3`, and so on.

Crab Fuel Function	
Distance	Total Fuel
1	1
2	3
3	6
4	10
5	15
6	21

Recursive Definition:

$$f(1) = 1$$

$$f(x) = x + f(x-1)$$

Day 7 – Part 2

- The individual crab fuel function is the sum of every number between 1 and the distance traveled.
- Can be calculated directly:

$$f(d) = \frac{d \cdot (d + 1)}{2}$$

- Overall crab fuel function:

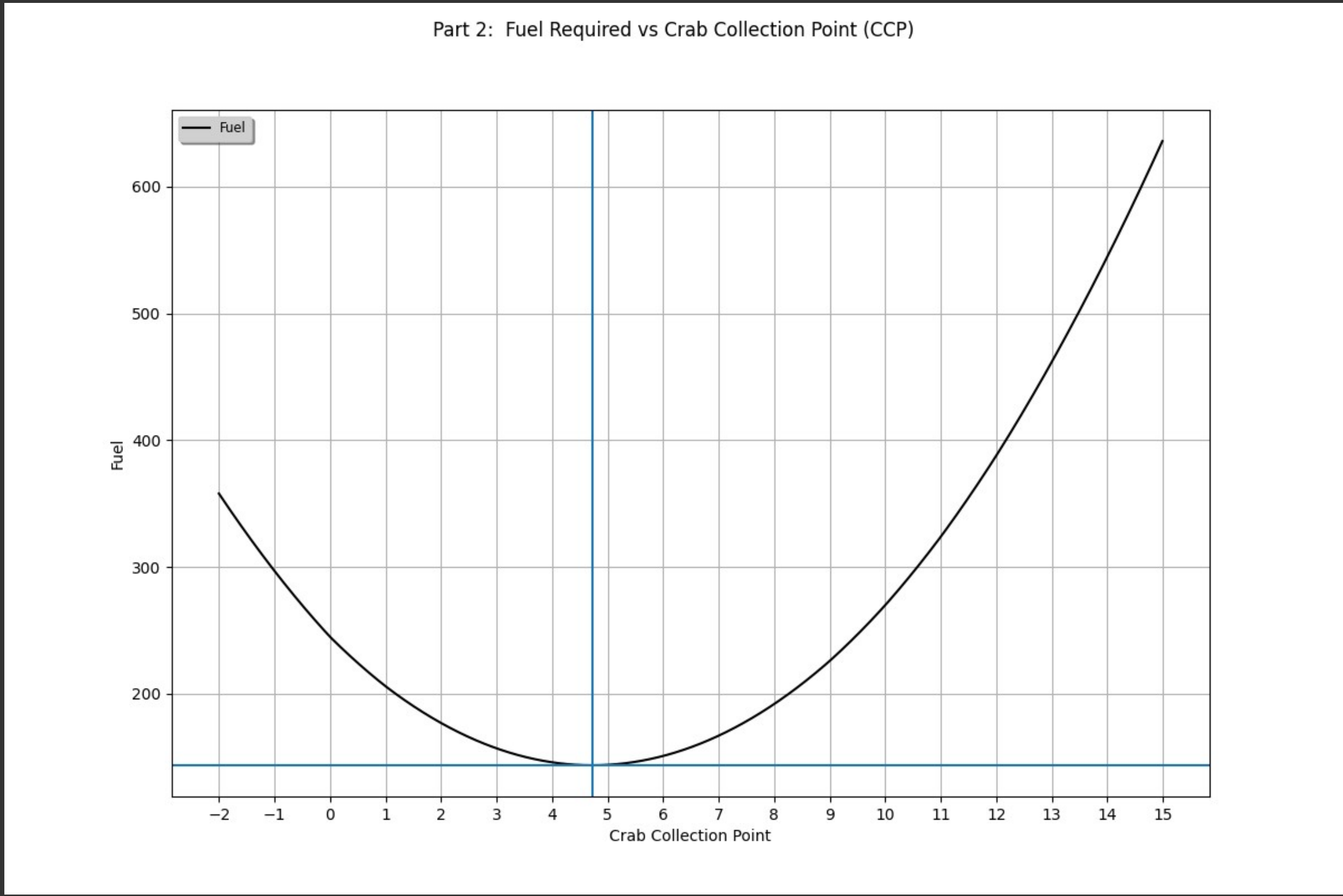
$$f(\hat{y}) = \sum_i \frac{d_i \cdot (d_i + 1)}{2} \text{ , where } d_i = |\hat{y} - x_i|$$

Day 7 – Part 2

Crab positions:

[0, 0, 0, 0, 1, 9, 10, 11, 12]

Minimum fuel cost occurs when the slope of the fuel function is zero.

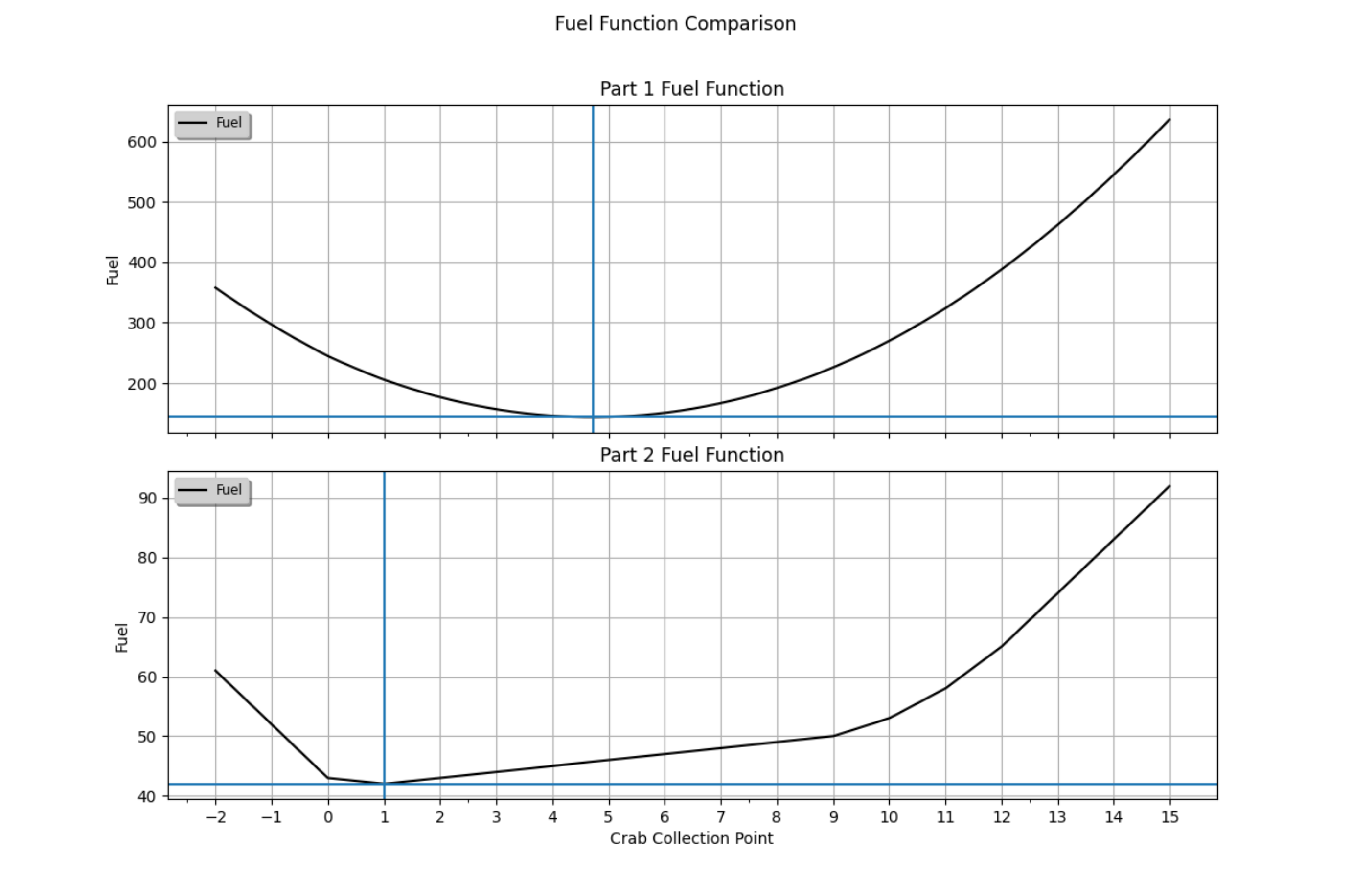


Day 7 – Part 2

Crab positions:

[0, 0, 0, 0, 1, 9, 10, 11, 12]

The crab collection point is dependent on the fuel function.



Day 7 – Part 2

$X = \{x_0, x_1, ...\}$, initial crab positions

$d_i = |\hat{y} - x_i|$

$f(X, \hat{y}) = \sum_i \frac{d_i(d_i + 1)}{2} = \frac{1}{2} \sum_i \left[d_i^2 + d_i\right]$, fuel cost to move all crabs to position $\hat{y}$

$$\frac{df}{d\hat{y}} = \frac{df}{dd_i} \cdot \frac{dd_i}{d\hat{y}}$$

$$\frac{df}{d\hat{y}} = \frac{1}{2} \sum_i 2 d_i + \frac{1}{2} \left[\sum_i \hat{y} \geq x_i - \sum_i x_i > \hat{y}\right]$$

$$\frac{df}{d\hat{y}} = \sum_i d_i + \frac{1}{2} \left[\sum_i \hat{y} \geq x_i - \sum_i x_i > \hat{y}\right]$$

$$\frac{df}{d\hat{y}} = \sum_i (\hat{y} - x_i) + \frac{1}{2} \left[\sum_i \hat{y} \geq x_i - \sum_i x_i > \hat{y}\right]$$

Day 7 – Part 2

$$\frac{df}{d\,\hat{y}} = \sum_i \left[\hat{y} - x_i\right] + \frac{1}{2}\left[\sum_i \hat{y} \geq x_i - \sum_i x_i > \hat{y}\right]$$

$$\frac{df}{d\,\hat{y}} = \sum_i \hat{y} - \sum_i x_i + \frac{1}{2}\left[\sum_i \hat{y} \geq x_i - \sum_i x_i > \hat{y}\right]$$

$$\frac{df}{d\,\hat{y}} = N \cdot \hat{y} - \sum_i x_i + \frac{1}{2}\left[\sum_i \hat{y} \geq x_i - \sum_i x_i > \hat{y}\right]$$

Day 7 – Part 2

$$0 = N \cdot \hat{y} - \sum_i x_i + \frac{1}{2}\left[\sum_i \hat{y} \geq x_i - \sum_i x_i > \hat{y}\right]$$

$$N \cdot \hat{y} = \sum_i x_i - \frac{1}{2}\left[\sum_i \hat{y} \geq x_i - \sum_i x_i > \hat{y}\right]$$

$$\hat{y} = \frac{1}{N}\sum_i x_i - \frac{1}{2N}\left[\sum_i \hat{y} \geq x_i - \sum_i x_i > \hat{y}\right]$$

Day 7 – Part 2

$$\hat{y} = \boxed{\frac{1}{N}\sum_i x_i} - \frac{1}{2N}\boxed{\left[\sum_i \hat{y} \geq x_i - \sum_i x_i > \hat{y}\right]}$$

What is this?

What is the min/max of this expression?

Day 7 – Part 2

$$\hat{y} = \underbrace{\frac{1}{N}\sum_i x_i}_{\substack{\text{Average of all}\\ \text{crab positions}}} - \frac{1}{2N}\underbrace{\left[\sum_i \hat{y} \geq x_i - \sum_i x_i > \hat{y}\right]}_{\substack{\text{Min: } -N\\ \text{Max: } N}}$$

Day 7 – Part 2

$$\hat{y} = \underbrace{\boxed{\frac{1}{N}\sum_i x_i}}_{\text{Average of all crab positions}} - \frac{1}{2} \cdot \frac{\pm N}{N}$$

Day 7 – Part 2

$$\hat{y} = \boxed{\frac{1}{N}\sum_i x_i} \pm \frac{1}{2}$$

Average of all crab positions

Day 7 – Part 2: Implementation

```python
#!/usr/bin/python3

import numpy as np

from code import interact

def get_fuel(dist):
    return (dist * (dist + 1)) / 2

with open("input", "r") as in_file:
    data = np.asarray(in_file.read().strip().split(','), np.int64)
    ccp_0 = np.round(data.mean() - 0.5) #crab collection point
    ccp_1 = np.round(data.mean() + 0.5) #crab collection point

    abs_diff = np.absolute(data - ccp_0)
    min_fuel = np.vectorize(get_fuel)(abs_diff).sum()
    print("%d %d" % (ccp_0, min_fuel))

    abs_diff = np.absolute(data - ccp_1)
    min_fuel = np.vectorize(get_fuel)(abs_diff).sum()
    print("%d %d" % (ccp_1, min_fuel))
```

Day 19

- Problem description: https://adventofcode.com/2021/day/19
- ~35 sensors
- ~25 beacons per sensor
- Beacon positions relative to the sensor are known, but the orientation of the beacons may be reflected and rotated about any axis any number of times
- Adjacent sensors share 12 beacons
- Tasks:
  - task 1: find the number of unique beacons
  - task 2: find the maximum manhattan distance between sensor centers
  - Determining the correct orientation of each sensor will essentially solve both tasks

Day 19

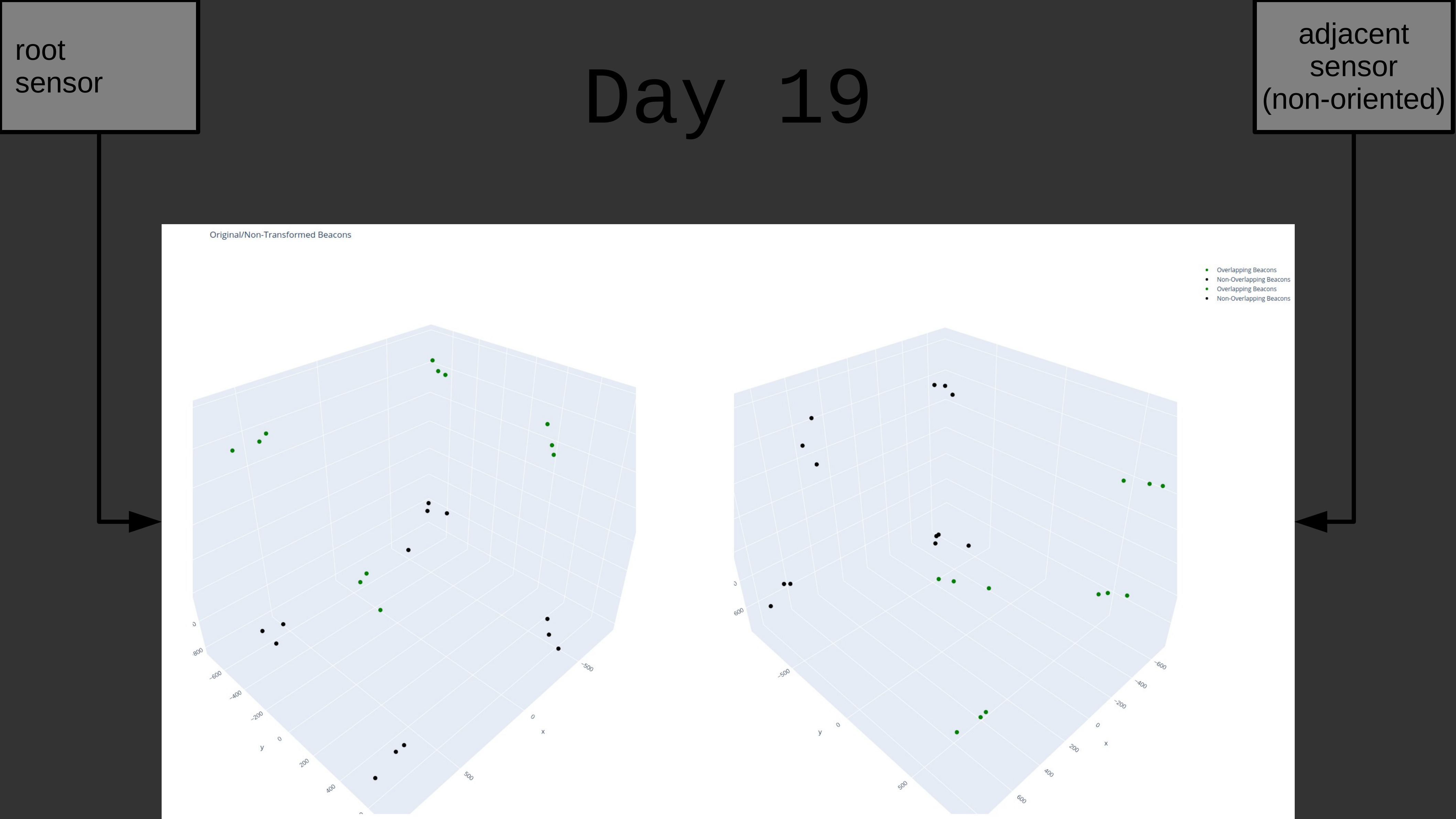
original orientation

```
>>> sensors[1].original_beacons
array([[ 670,  487, -847],
       [-432,  537, -395],
       [-521,  374,  359],
       [-666,  441,  313],
       [ 740,  701,  863],
       [-432,  701, -383],
       [ 721, -698, -623],
       [ 818,  605,  871],
       [-282, -564, -496],
       [ 766,  577,  737],
       [-375, -596, -489],
       [-433, -537, -427],
       [ 745, -562, -653],
       [ 650, -523,  785],
       [  24,   36,    8],
       [ 608, -417,  733],
       [-386, -616,  490],
       [ 453, -460,  776],
       [ 745, -802, -648],
       [-454,  633, -472],
       [-403, -500,  596],
       [ 693,  557, -710],
       [ 669,  547, -874],
       [-462, -478,  508],
       [-529,  539,  346]], dtype=int32)
```

true orientation

```
>>> sensors[1].original_beacons @ sensors[1].transform
array([[ 847, -670,  487],
       [ 395,  432,  537],
       [-359,  521,  374],
       [-313,  666,  441],
       [-863, -740,  701],
       [ 383,  432,  701],
       [ 623, -721, -698],
       [-871, -818,  605],
       [ 496,  282, -564],
       [-737, -766,  577],
       [ 489,  375, -596],
       [ 427,  433, -537],
       [ 653, -745, -562],
       [-785, -650, -523],
       [  -8,  -24,   36],
       [-733, -608, -417],
       [-490,  386, -616],
       [-776, -453, -460],
       [ 648, -745, -802],
       [ 472,  454,  633],
       [-596,  403, -500],
       [ 710, -693,  557],
       [ 874, -669,  547],
       [-508,  462, -478],
       [-346,  529,  539]])
```

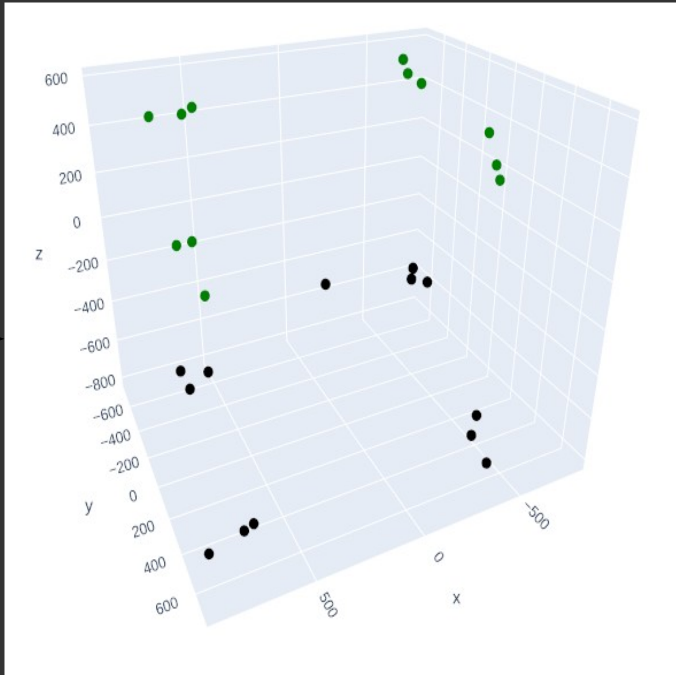
Day 19

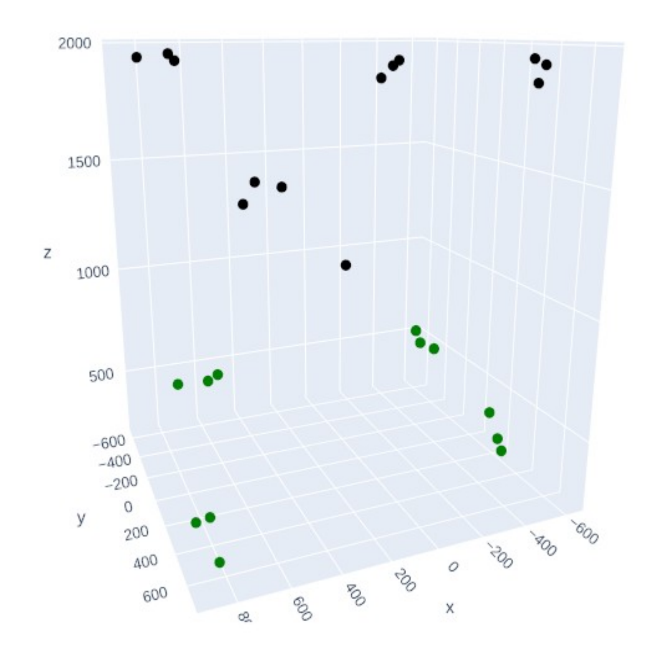

root sensor
adjacent sensor (non-oriented)
Original/Non-Transformed Beacons
Overlapping Beacons
Non-Overlapping Beacons
Overlapping Beacons
Non-Overlapping Beacons
x
y

Day 19

root sensor

adjacent sensor (oriented)





Day 19

General approach:

1. The root node is oriented correctly
2. Calculate inter-beacon pairwise distances for each sensor
3. From an oriented sensor s1, consider an unoriented sensor s2
4. If 12 matching inter-beacon distances:
   - while s2 not oriented:
     - Rotate/reflect s2 beacons using one of 24 projection matrices
     - for each pair of overlapping beacons b1, b2
       - s2 center = b1 – b2
       - if s2 center is equal for all overlapping pairs of beacons, s2 is oriented
   - if not, try another projection matrix
5. If unoriented sensors remain, goto 3

Day 19

Generate inter-beacon pairwise distances for each sensor:

```python
        #get pairwise beacon-to-beacon distances
        self.dist = np.zeros(shape=(self.beacons.shape[0], self.beacons.shape[0]), dtype=np.float64)

        for idx in range(self.beacons.shape[0]):
            for jdx in range(self.beacons.shape[0]):

                if idx == jdx:
                    continue

                self.dist[idx, jdx] = (
                    (self.beacons[idx][0] - self.beacons[jdx][0])**2 +
                    (self.beacons[idx][1] - self.beacons[jdx][1])**2 +
                    (self.beacons[idx][2] - self.beacons[jdx][2])**2) ** 0.5
```

```
>>> sensors[0].dist
array([[   0. , 1512. , 1906. ,   66.4, 1350.6, 1531.9, 1996.8, 1207.1, 1431.8, 1515.5, 2137.4, 1494.5, 1743.5, 1493.4,  187.8, 1272.9, 1249.2, 1779.3, 1112.1, 1980.7, 2375.4, 1608.5, 1492.9, 2258.5, 2306.2],
       [1512. ,    0. , 1831.5, 1520.1, 1550.3,  990. , 1005.5, 1542. ,   95.5,  151.8, 1157.4, 1605.8, 1752.7,  939.4, 1333.4,  859.5, 1603.5, 1755.2, 1554.4,  963.4, 1502.9, 1099.6, 1521.3, 1337.3, 1450.6],
       [1906. , 1831.5,    0. , 1941.8, 1327.3, 1504.6, 1571.7, 1266.6, 1815.8, 1963.1, 1607.5, 1249. ,  204.4, 1529.1, 1872.9, 1005.1, 1261.1,  148.8, 1374.4, 1559.2, 1058.5, 1461.1, 1284.4, 1085.7, 1016.4],
       [  66.4, 1520.1, 1941.8,    0. , 1338.1, 1578.6, 1994.8, 1269.6, 1441.6, 1518.2, 2134.9, 1485.6, 1783.2, 1538.7,  207. , 1298.7, 1312.2, 1817.5, 1176.1, 1979.9, 2404.8, 1657.6, 1482.5, 2284.6, 2336. ],
       [1350.6, 1550.3, 1327.3, 1338.1,    0. , 1876.3, 1201.6, 1718.9, 1539.2, 1594. , 1264.3,  164.8, 1309. , 1853. , 1337.2, 1020.5, 1751.9, 1298.1, 1734.2, 1218.1, 1766.3, 1927.3,  182.1, 1645.3, 1719.2],
       [1531.9,  990. , 1504.6, 1578.6, 1876.3,    0. , 1640.1,  817.1,  934.2, 1108.4, 1769.6, 1914.7, 1366.2,   62.8, 1390.7,  885.7,  863.7, 1392.6,  866.2, 1593.3, 1394. ,  126.9, 1878. , 1326. , 1330.1],
       [1996.8, 1005.5, 1571.7, 1994.8, 1201.6, 1640.1,    0. , 2001.9, 1070. , 1065.4,  159.3, 1160.8, 1605.3, 1616.7, 1871.5,  991.1, 2048.4, 1574.9, 2055.8,   52.4, 1145.7, 1697.8, 1063.4,  967.7, 1122.8],
       [1207.1, 1542. , 1266.6, 1269.6, 1718.9,  817.1, 2001.9,    0. , 1468.7, 1644.6, 2124.3, 1768.9, 1067.5,  832.6, 1159.1, 1016.2,   65. , 1120.2,  135.8, 1968.2, 1730.3,  791.5, 1778.5, 1694.3, 1661.6],
       [1431.8,   95.5, 1815.8, 1441.6, 1539.2,  934.2, 1070. , 1468.7,    0. ,  183.2, 1224.3, 1601.2, 1727.3,  881. , 1251.7,  832.6, 1531. , 1732.9, 1476. , 1028.5, 1535. , 1048. , 1520.6, 1373.5, 1479.8],
       [1515.5,  151.8, 1963.1, 1518.2, 1594. , 1108.4, 1065.4, 1644.6,  183.2,    0. , 1216.7, 1658.9, 1882.8, 1054.7, 1334.9,  978.8, 1707.5, 1885.8, 1647.2, 1027.1, 1643.9, 1222.9, 1571.2, 1475. , 1592.9],
       [2137.4, 1157.4, 1607.5, 2134.9, 1264.3, 1769.6,  159.3, 2124.3, 1224.3, 1216.7,    0. , 1206.3, 1660. , 1749.3, 2017. , 1115.5, 2168.6, 1624.9, 2183.3,  197.2, 1146.1, 1821.1, 1112.3,  977.9, 1132.4],
       [1494.5, 1605.8, 1249. , 1485.6,  164.8, 1914.7, 1160.8, 1768.9, 1601.2, 1658.9, 1206.3,    0. , 1254.6, 1895.3, 1477.9, 1040.7, 1799.1, 1237.1, 1795.8, 1180.4, 1683.3, 1958.2,  110.9, 1569.8, 1640.4],
       [1743.5, 1752.7,  204.4, 1783.2, 1309. , 1366.2, 1605.3, 1067.5, 1727.3, 1882.8, 1660. , 1254.6,    0. , 1390.3, 1712.5,  913.8, 1061.1,   59.8, 1173.5, 1588.9, 1136.9, 1323.4, 1289.6, 1149.5, 1085.7],
       [1493.4,  939.4, 1529.1, 1538.7, 1853. ,   62.8, 1616.7,  832.6,  881. , 1054.7, 1749.3, 1895.3, 1390.3,    0. , 1348.2,  868.9,  882.4, 1416.3,  874.1, 1570. , 1417. ,  189.5, 1856.6, 1341.4, 1352.7],
       [ 187.8, 1333.4, 1872.9,  207. , 1337.2, 1390.7, 1871.5, 1159.1, 1251.7, 1334.9, 2017. , 1477.9, 1712.5, 1348.2,    0. , 1158.6, 1208.2, 1746.2, 1069.9, 1852.4, 2263.5, 1475.6, 1463.8, 2140.6, 2194.3],
       [1272.9,  859.5, 1005.1, 1298.7, 1020.5,  885.7,  991.1, 1016.2,  832.6,  978.8, 1115.5, 1040.7,  913.8,  868.9, 1158.6,    0. , 1062.8,  918.3, 1069.6,  961. , 1110.5,  931.7,  999.9,  985.9, 1042.8],
       [1249.2, 1603.5, 1261.1, 1312.2, 1751.9,  863.7, 2048.4,   65. , 1531. , 1707.5, 2168.6, 1799.1, 1061.1,  882.4, 1208.2, 1062.8,    0. , 1115.1,  166.4, 2015.2, 1749.9,  830.9, 1811.8, 1720.2, 1682. ],
       [1779.3, 1755.2,  148.8, 1817.5, 1298.1, 1392.6, 1574.9, 1120.2, 1732.9, 1885.8, 1624.9, 1237.1,   59.8, 1416.3, 1746.2,  918.3, 1115.1,    0. , 1226.5, 1559.4, 1099.8, 1350.6, 1271.1, 1114.1, 1050.4],
       [1112.1, 1554.4, 1374.4, 1176.1, 1734.2,  866.2, 2055.8,  135.8, 1476. , 1647.2, 2183.3, 1795.8, 1173.5,  874.1, 1069.9, 1069.6,  166.4, 1226.5,    0. , 2022.7, 1844.1,  855.3, 1805.2, 1800. , 1774.7],
       [1980.7,  963.4, 1559.2, 1979.9, 1218.1, 1593.3,   52.4, 1968.2, 1028.5, 1027.1,  197.2, 1180.4, 1588.9, 1570. , 1852.4,  961. , 2015.2, 1559.4, 2022.7,    0. , 1117.8, 1651.2, 1083.7,  937.6, 1093.5],
       [2375.4, 1502.9, 1058.5, 2404.8, 1766.3, 1394. , 1145.7, 1730.3, 1535. , 1643.9, 1146.1, 1683.3, 1136.9, 1417. , 2263.5, 1110.5, 1749.9, 1099.8, 1844.1, 1117.8,    0. , 1360.9, 1647. ,  187. ,   71.6],
       [1608.5, 1099.6, 1461.1, 1657.6, 1927.3,  126.9, 1697.8,  791.5, 1048. , 1222.9, 1821.1, 1958.2, 1323.4,  189.5, 1475.6,  931.7,  830.9, 1350.6,  855.3, 1651.2, 1360.9,    0. , 1926.2, 1309.1, 1298. ],
       [1492.9, 1521.3, 1284.4, 1482.5,  182.1, 1878. , 1063.4, 1778.5, 1520.6, 1571.2, 1112.3,  110.9, 1289.6, 1856.6, 1463.8,  999.9, 1811.8, 1271.1, 1805.2, 1083.7, 1647. , 1926.2,    0. , 1524.3, 1604.3],
       [2258.5, 1337.3, 1085.7, 2284.6, 1645.3, 1326. ,  967.7, 1694.3, 1373.5, 1475. ,  977.9, 1569.8, 1149.5, 1341.4, 2140.6,  985.9, 1720.2, 1114.1, 1800. ,  937.6,  187. , 1309.1, 1524.3,    0. ,  163.2],
       [2306.2, 1450.6, 1016.4, 2336. , 1719.2, 1330.1, 1122.8, 1661.6, 1479.8, 1592.9, 1132.4, 1640.4, 1085.7, 1352.7, 2194.3, 1042.8, 1682. , 1050.4, 1774.7, 1093.5,   71.6, 1298. , 1604.3,  163.2,    0. ]])
```

Day 19

Generate projection matrices:

```python
    def get_transforms(self):
        """
        Generate the 24 unique transformation matricies
        """

        #define the basic identity and rotation matrices
        ident = np.asarray([[ 1,  0,  0],
                            [ 0,  1,  0],
                            [ 0,  0,  1]])

        rot_y = np.asarray([[ 0,  0,  1],
                            [ 0,  1,  0],
                            [-1,  0,  0]])

        rot_x = np.asarray([[ 1,  0,  0],
                            [ 0,  0, -1],
                            [ 0,  1,  0]])

        rot_z = np.asarray([[ 0, -1,  0],
                            [ 1,  0,  0],
                            [ 0,  0,  1]])

        #generate the 24 unique transformation matrices
        transforms = list()

        #generate every possible combination of 0-3 rotations about each axis
        for x in range(4):
            for y in range(4):
                for z in range(4):

                    transform = ident.copy()

                    for _ in range(x):
                        transform = rot_x @ transform

                    for _ in range(y):
                        transform = rot_y @ transform

                    for _ in range(z):
                        transform = rot_z @ transform

                    transforms.append(transform.copy())

        #convert to numpy array and retain only the 24 unique transfomation matrices
        transforms = np.asarray(transforms)
        transforms = np.unique(transforms, axis=0)
        return transforms
```

Day 19

Projection matrix examples:

```
>>> Sensor.transforms[:12]
array([[[-1,  0,  0],
        [ 0, -1,  0],
        [ 0,  0,  1]],

       [[-1,  0,  0],
        [ 0,  0, -1],
        [ 0, -1,  0]],

       [[-1,  0,  0],
        [ 0,  0,  1],
        [ 0,  1,  0]],

       [[-1,  0,  0],
        [ 0,  1,  0],
        [ 0,  0, -1]],

       [[ 0, -1,  0],
        [-1,  0,  0],
        [ 0,  0, -1]],

       [[ 0, -1,  0],
        [ 0,  0, -1],
        [ 1,  0,  0]],

       [[ 0, -1,  0],
        [ 0,  0,  1],
        [-1,  0,  0]],

       [[ 0, -1,  0],
        [ 1,  0,  0],
        [ 0,  0,  1]],

       [[ 0,  0, -1],
        [-1,  0,  0],
        [ 0,  1,  0]],

       [[ 0,  0, -1],
        [ 0, -1,  0],
        [-1,  0,  0]],

       [[ 0,  0, -1],
        [ 0,  1,  0],
        [ 1,  0,  0]],

       [[ 0,  0, -1],
        [ 1,  0,  0],
        [ 0, -1,  0]]])
```

```
>>> Sensor.transforms[12:]
array([[[ 0,  0,  1],
        [-1,  0,  0],
        [ 0, -1,  0]],

       [[ 0,  0,  1],
        [ 0, -1,  0],
        [ 1,  0,  0]],

       [[ 0,  0,  1],
        [ 0,  1,  0],
        [-1,  0,  0]],

       [[ 0,  0,  1],
        [ 1,  0,  0],
        [ 0,  1,  0]],

       [[ 0,  1,  0],
        [-1,  0,  0],
        [ 0,  0,  1]],

       [[ 0,  1,  0],
        [ 0,  0, -1],
        [-1,  0,  0]],

       [[ 0,  1,  0],
        [ 0,  0,  1],
        [ 1,  0,  0]],

       [[ 0,  1,  0],
        [ 1,  0,  0],
        [ 0,  0, -1]],

       [[ 1,  0,  0],
        [ 0, -1,  0],
        [ 0,  0, -1]],

       [[ 1,  0,  0],
        [ 0,  0, -1],
        [ 0,  1,  0]],

       [[ 1,  0,  0],
        [ 0,  0,  1],
        [ 0, -1,  0]],

       [[ 1,  0,  0],
        [ 0,  1,  0],
        [ 0,  0,  1]]])
```

```
>>> Sensor.transforms @ (1, 2, 3)
array([[-1, -2,  3],
       [-1, -3, -2],
       [-1,  3,  2],
       [-1,  2, -3],
       [-2, -1, -3],
       [-2, -3,  1],
       [-2,  3, -1],
       [-2,  1,  3],
       [-3, -1,  2],
       [-3, -2, -1],
       [-3,  2,  1],
       [-3,  1, -2],
       [ 3, -1, -2],
       [ 3, -2,  1],
       [ 3,  2, -1],
       [ 3,  1,  2],
       [ 2, -1,  3],
       [ 2, -3, -1],
       [ 2,  3,  1],
       [ 2,  1, -3],
       [ 1, -2, -3],
       [ 1, -3,  2],
       [ 1,  3, -2],
       [ 1,  2,  3]])
```

Day 19

Find overlapping beacons based on pairwise inter-beacon distances:

```
    def get_pairs(self, dst):
        #determine the mapping from source-sensor-beacon-index to destination-sensor-beacon-index
        pairs = list()

        #intersect1d returns the values shared between the 3D arrays
        intersection_set = np.intersect1d(self.dist, dst.dist)

        #a shape of 67 implies the self sensor and the destination sensor have 12 overlapping beacons
        #(12 * 11) / 2 unique pairwise distances
        #this check is optional, but speeds up execution quite a bit because it skips the pairwise comparisons below on
        #non-overlapping sensors
        if intersection_set.shape[0] == 67:
            for idx in range(self.dist.shape[0]):
                for jdx in range(dst.dist.shape[0]):
                    #check for 12 identical pairwise-distances using src-beacon-idx and dest-beacon-jdx as the base of
                    #measurement
                    #this indicates src.beacons[idx] and dst.beacons[jdx] are the same beacon
                    tmp = np.intersect1d(self.dist[idx], dst.dist[jdx])
                    if tmp.shape[0] == 12:
                        pairs.append((idx, jdx))

        return pairs
```

Day 19

Core logic of brute forcing the transforms and traversing the graph.

```python
    def orient_peers(self):
        if self.visited:
            return

        self.visited = True

        for dst in Sensor.sensors:
            if dst.visited :
                continue

            pairs = self.get_pairs(dst)

            if len(pairs) != 12:
                continue

            for transform in Sensor.transforms:
                #apply a transform to dst
                tmp = dst.beacons @ transform

                #calculate dst center relative to src based on each beacon
                dst_centers = set()

                for idx, jdx in pairs:
                    dst_centers.add(tuple(self.beacons[idx] - tmp[jdx]))

                #if the number of dst_centers is 1, this indicates the transform was correct
                if len(dst_centers) == 1:
                    #add the scanner_center to the list of scanner centers
                    Sensor.sensor_centers.append(dst_centers.pop())

                    #set the beacons to the transformed orientation and apply the linear offset of the scanner center
                    dst.beacons = tmp + Sensor.sensor_centers[-1]
                    dst.orient_peers()
                    break
```

Day 19

Final logic to determine the number of unique beacons (part 1), and the maximal manhattan distance between sensor centers.

```python
with open(file_name, "r") as in_file:
    sensors = list()
    sensor_datas = in_file.read().split("\n\n")

    for sensor_data in sensor_datas:
        beacons = [coord.split(',') for coord in sensor_data.strip().split('\n')[1:]]
        sensors.append(Sensor(beacons))

    sensors[0].orient_peers()
    concatenated = np.concatenate([sensors[idx].beacons for idx in range(len(Sensor.sensors))])
    concat_uniq = np.unique(concatenated, axis=0)
    print(concat_uniq.shape)

    max_dist = 0
    for idx in range(len(sensors) - 1):
        for jdx in range(idx + 1, len(sensors)):
            dist = abs(Sensor.sensor_centers[idx][0] - Sensor.sensor_centers[jdx][0]) + \
                   abs(Sensor.sensor_centers[idx][1] - Sensor.sensor_centers[jdx][1]) + \
                   abs(Sensor.sensor_centers[idx][2] - Sensor.sensor_centers[jdx][2])

            if dist > max_dist:
                max_dist = dist

    print(max_dist)
```

Day 19

Visualization of oriented and arranged sensors/beacons.