Advent of Code 2023; Day 24 - part 2

Vincent BONINO

Contents

---

1 Context

Find *here* the statement of the challenge.

My goal was to find the solution using mathematics only.

Here is how it went:

2 Modelisation

2.1 Representing the problem

Many hailstones are flying through the air in perfectly linear trajectories. And we are aiming to throw one ourselves to **hit every hailstone in a single throw**!

So, once we have assigned a number to all N hailstones, we can represent each hailstone with two numbers only:

$$(p_{0_i}, v_i) \iff p_i(t) = p_{0_i} + t * v_i$$

With

- p_{0_i} being the position of the i-th stone at time 0 ($p_i(0)$),
- v_i being the linear speed of the i-th stone,
- $p_i(t)$ being the position of the i-th stone at time t.

And since the problem lives in a 3 dimension space:

$$\forall i \in 1..N, \forall t, p_i(t) = (p_{i_x}, p_{i_y}, p_{i_z}) = (x_i, y_i, z_i) \tag{1}$$

And finally, what we are looking for is

$$(p_{0_S}, v_S), \forall n \in 1..N, \exists t_n, p_S(t_n) = p_n(t_n) \tag{2}$$

2.2 First hypothesis

At first, finding a solution to (2) is very, very, very unlikely. Particularly when the input to this problem contains **300 hailstones**.

In the challenge's statement, we can read:

> *While it seems extremely unlikely, if you throw it just right, you should be able to hit every hailstone in a single throw!*

So there *is* a solution, we know it. We can focus on finding it rather than asking ourselves it it exists.

And I will build all my computation on the following hypothesis:

A solution to a sub-problem (with less hailstone) should be the solution to the global problem.

Note: it will also allow me to lose the equivalence during my reasoning. I am only looking for a working example.

2.3 Finding a sub-problem

My first approach was to attempt to solve it as a linear equations system problem.
As we are in 3D (1) and we modeled the problem as (2), we are looking for 6 values. (3 for the starting position, and 3 for the speed).

How many hailstones do I have to consider ?

It is trivial to find a solution for a 2-hailstone sub-problem:

1. Take $A = p_1(t_1)$, hailstone 1's position at t_1,
2. Take $B = p_2(t_2)$, hailstone 2's position at t_2,
3. Name $\overrightarrow{v} = \overrightarrow{AB}$, the distance between both points
4. Name $p = A - \overrightarrow{v}$

And $(p, \overrightarrow{v})$ is solution to this problem.

At least to me, it seems way harder to find a solution for a 3-hailstone sub-problem so... let's start with 3.

3 Sub-problem attempts

3.1 3-hailstone sub-problem

$$\begin{cases} p_1(t_1) = p_S(t_1) \\ p_2(t_2) = p_S(t_2) \\ p_3(t_3) = p_S(t_3) \end{cases} \iff \begin{matrix} p_{0_1} + t_1 * v_1 = p_{0_S} + t_1 * v_S \\ p_{0_2} + t_2 * v_2 = p_{0_S} + t_2 * v_S \\ p_{0_3} + t_3 * v_3 = p_{0_S} + t_3 * v_S \end{matrix}$$

with each one of this line being actually in 3 dimensions (example with 1):

$$\begin{cases} x_{0_1} + t_1 * v_{x_1} = x_{0_S} + t_1 * v_{x_S} \\ y_{0_1} + t_1 * v_{y_1} = y_{0_S} + t_1 * v_{y_S} \\ z_{0_1} + t_1 * v_{z_1} = z_{0_S} + t_1 * v_{z_S} \end{cases}$$

Both t_1 and v_{x_S} are unknown values, so I write t_1 as a relation of other values to have a linear system.

Finally, we get:

$$\begin{cases} t_1 = \dfrac{x_{0_S} - x_{0_1}}{v_{x_1} - v_{x_S}}; t_2 = \dfrac{y_{0_S} - y_{0_2}}{v_{y_2} - v_{y_S}}; t_3 = \dfrac{z_{0_S} - z_{0_3}}{v_{z_3} - v_{z_S}} \\ [1] v_{x_1} * y_{0_1} - v_{x_S} * y_{0_1} + x_{0_S} * v_{y_1} - x_{0_1} * v_{y_1} = v_{x_1} * y_{0_S} - \boxed{v_{x_S} * y_{0_S}} + \boxed{x_{0_S} * v_{y_S}} - x_{0_1} * v_{y_S} \\ [2] v_{x_1} * z_{0_1} - v_{x_S} * z_{0_1} + x_{0_S} * v_{z_1} - x_{0_1} * v_{z_1} = v_{x_1} * z_{0_S} - \boxed{v_{x_S} * z_{0_S}} + \boxed{x_{0_S} * v_{z_S}} - x_{0_1} * v_{z_S} \\ [3] v_{y_2} * x_{0_2} - v_{y_S} * x_{0_2} + y_{0_S} * v_{x_2} - y_{0_2} * v_{x_2} = v_{y_2} * x_{0_S} - \boxed{v_{y_S} * x_{0_S}} + \boxed{y_{0_S} * v_{x_S}} - y_{0_2} * v_{x_S} \\ [4] v_{y_2} * z_{0_2} - v_{y_S} * z_{0_2} + y_{0_S} * v_{z_2} - y_{0_2} * v_{z_2} = v_{y_2} * z_{0_S} - \boxed{v_{y_S} * z_{0_S}} + \boxed{y_{0_S} * v_{z_S}} - y_{0_2} * v_{z_S} \\ [5] v_{z_3} * x_{0_3} - v_{z_S} * x_{0_3} + z_{0_S} * v_{x_3} - z_{0_3} * v_{x_3} = v_{z_3} * x_{0_S} - \boxed{v_{z_S} * x_{0_S}} + \boxed{z_{0_S} * v_{x_5}} - z_{0_3} * v_{x_S} \\ [6] v_{z_3} * y_{0_3} - v_{z_S} * y_{0_3} + z_{0_S} * v_{y_3} - z_{0_3} * v_{y_3} = v_{z_3} * y_{0_S} - \boxed{v_{z_S} * y_{0_S}} + \boxed{z_{0_S} * v_{y_5}} - z_{0_3} * v_{y_S} \end{cases} \quad (3)$$

We end up with this 9 equations - 9 unknown values sub-problem.
And if we ignore times we wrote as a function of other values, a 6 equations - 6 unknown values sub-sub-problem.

Done ? Not exactly... the squared terms of the right side of the multiplication are multiplication of two unknown values.
This is a non-linear system and I won't be able to solve it easily.

Unless... unless this terms cancel out each other. And they actually do!
Line [1] terms cancel out with line [4]'s, line [2]'s with line [5]'s and line [3]'s with line [6]'s.

I can apply the following line modifications, as allowed to solve linear equations systems:

- $L'_1 \leftarrow L_1 + L_3$
- $L'_2 \leftarrow L_2 + L_5$
- $L'_3 \leftarrow L_3 + L_1$
- $L'_4 \leftarrow L_4 + L_6$
- $L'_5 \leftarrow L_5 + L_2$
- $L'_6 \leftarrow L_6 + L_4$

Note: using old lines after they were updated makes me loose equivalence, but as previously stated, this is not an issue for this exercise.
The real issue is that L_1 and L_3 are now the same, and similarly for L_2-L_5 and L_4-L_6.

In the end, all I get is a 6 equations - 9 unknown values problem.

Let's get bigger. If we lose half our equations (not counting time ones), what happens if we double our input ?

3.2 6-hailstone sub-problem

We might have had doubts with 3, but now with 6 it would be VERY unlikely to find a solution that is not also the global one.

I'm not re-detailing all the computations step-by-step. We basically duplicate the previous subsection, and merge them together:

1. Take 6 random hailstones of the input
2. Write equations as in (3)
3. Simplify as explained above
4. Re-write the problem as a matrix equation

We are now attempting to solve $Ax = B$, with

$$A = \begin{pmatrix} v_{y_1} - v_{y_2} & v_{x_2} - v_{x_1} & 0 & y0_2 - y0_1 & x0_1 - x0_2 & 0 \\ v_{z_1} - v_{z_3} & 0 & v_{x_3} - v_{x_1} & z0_3 - z0_1 & 0 & x0_1 - x0_3 \\ 0 & v_{z_2} - v_{z_3} & v_{y_3} - v_{y_2} & 0 & z0_3 - z0_2 & y0_2 - y0_3 \\ v_{y_4} - v_{y_5} & v_{x_5} - v_{x_4} & 0 & y0_5 - y0_4 & x0_4 - x0_5 & 0 \\ v_{z_4} - v_{z_6} & 0 & v_{x_6} - v_{x_4} & z0_6 - z0_4 & 0 & x0_4 - x0_6 \\ 0 & v_{z_5} - v_{z_6} & v_{y_6} - v_{y_5} & 0 & z0_6 - z0_5 & y0_5 - y0_6 \end{pmatrix}$$

$$X = \begin{pmatrix} x0_S \\ y0_S \\ z0_S \\ v_{x_S} \\ v_{y_S} \\ v_{z_S} \end{pmatrix}$$

$$B = \begin{pmatrix} x0_1 * v_{z_1} - v_{x_1} * z0_1 + z0_3 * v_{x_3} - v_{z_3} * x0_3 \\ x0_1 * v_{y_1} - v_{x_1} * y0_1 + y0_2 * v_{x_2} - v_{y_2} * x0_2 \\ y0_2 * v_{z_2} - v_{y_2} * z0_2 + z0_3 * v_{y_3} - v_{z_3} * y0_3 \\ x0_4 * v_{z_4} - v_{x_4} * z0_4 + z0_6 * v_{x_6} - v_{z_6} * x0_6 \\ x0_4 * v_{y_4} - v_{x_4} * y0_4 + y0_5 * v_{x_5} - v_{y_5} * x0_5 \\ y0_5 * v_{z_5} - v_{y_5} * z0_5 + z0_6 * v_{y_6} - v_{z_6} * y0_6 \end{pmatrix}$$

We have enough equations, the system is linear. Now, there is *just* to solve it.

4 Solving the equations system

I could have transformed this system matrix in its echelon form by hand, and then have solved the problem.
I *could*, but it would have been long, very tedious but not particularly difficult.
To ease my pain, I made it in python (see `day24.py`).

However, there was a few things to pay attention to, mostly the fact that the puzzle input's value are rather large number.
So I had to be extra cautious when computing matrix determinants, to avoid float precision errors. For example, using build-in or external libraries didn't work fork for me.

- I went for computing the echelon form of the A matrix.
  I found out about the Bareiss algorithm, a Gaussian elimination variant allowing division/fraction free computation. The article tells about getting the echelon form but I could not find it, however I got the determinant of the matrix.

- I completed with Cramer's rule, allowing to solve a system only using determinants.

See the code for a bit more details/implementation.

And with that, I got all I need !