Yann PIERRE

Maëllys MASCART

Maxent BRUNEL

ODE : LAND A PLANE

This document will expose our models, plans and solutions for the glider project.

Introduction :

ODEs are used in a wide range of models to study and predict different systems. Some famous models include :

- ➢ The Lotka-Voltera predator-prey model,
- ➢ The Verhulst equation,
- ➢ The SIR model
- ➢ Glider model

```
function Glider_EE(T,h,teta,vini,mud,mul)
      % Glider model
      % Explicit-Euler
      % % % % % % % % % % % % % % %
      N=round(T/h); % stepsize
      t=linspace(0,T,N+1); % grid
      y=zeros(4,N+1); % numerical approximation
      % EE:
      y(:,1)=[0 0 teta vini]';
      for j=1:N
      y(:,j+1)=y(:,j)+h*Glyder(mud,mul,y(:,j));
   end
   % visualization:
      plot(y(1,:),y(2,:));
      xlabel('x')
   ylabel('y')
end
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function f = Glyder(mud,mul,y)
      % SIR model
      f = zeros(4,1);
      f(1) = y(4)*cos(y(3));
      f(2) = y(4)*sin(y(3));
      f(3) = -9.8*cos(y(3))/y(4)+mul*y(4);
   f(4) = -9.8*sin(y(3))-mud*(y(4))^2;
end
```

Objectives:

1) The plane must reach/touch the ground horizontally, it must not crash.
2) The plane must land at a specific point.
   More especially, the goal is the plane must arrive at a specific place AND touch the floor horizontally.

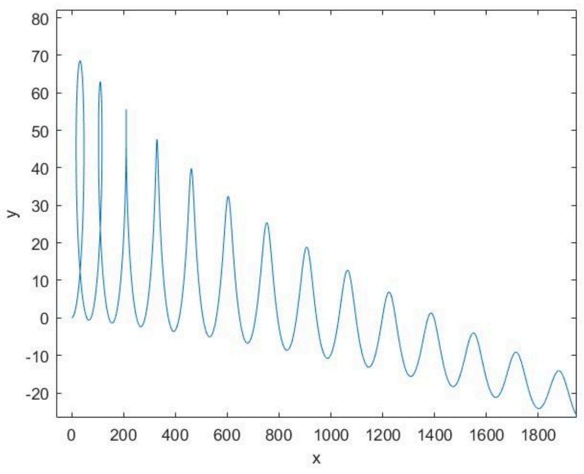
If we have time:

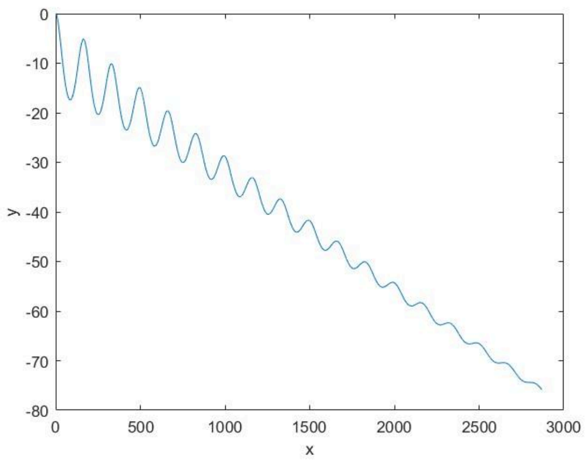
3) The plane must go as far as possible.

Methods / Solutions:

1) Landing :

   With the above code, we succeed to plot the trajectory of a glider, and we can modify the starting speed, angle and position, which will probably need to be randomized at every episode.





Now, we want to simulate a landing. The first thing would be to set up a threshold at y=0 which indicates a “crash”.

Second step would be to set the “action” for the glider to adjust its trajectory, which means adjust the angle teta. I’m seeing from the examples we were given, that the “action” is changing the state of the wing (which translates in real life to moving the wing’s flaps, which affects the lift and drag coefficients).

But landing also means reaching this y=0 slowly and steady. So a first idea would be to put a second line, close to the ground (y=5,10 ? we’ll test), and reward the agent for staying in this interval. And maybe a bigger reward if he reaches a speed of 0 in it.

We’ll need to check if we need to put a threshold on the angle teta. It could be funny to see it diving straight down, and adjust the trajectory at the last second.

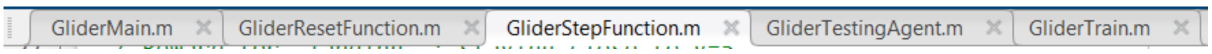
2) Specific landing

We'll need to randomize a "landing spot", and adjust the threshold at every episode (If you're above the landing interval and beyond the landing point you failed etc..)

First results:

All our code is available on this github link : https://github.com/MaxBasse/Matlab
There are several versions (so different "branches" in github) and we will explain in the following how we end up here.

1) Environment



We took your glider project, and modified it to keep only what is interesting for us. "GliderMain" creates a new environment and a new agent, who will be trained by "GliderTrain", using the step and reset function. "GliderTestingAgent" is used to compare a trained agent and a normal glider without any restriction.

Because we also use RK4 for our solutions, the script which will be the most changed is the step function, and especially the reward and penalty.

2) First approach

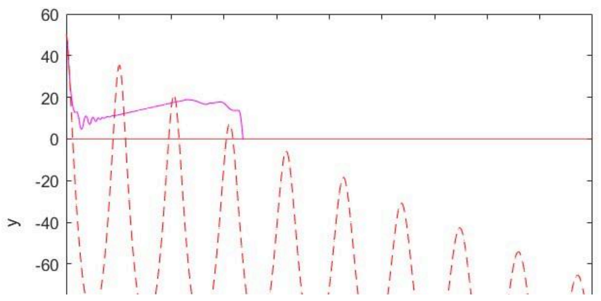
For now, we consider the test to be failed only if y<0 (we don't look at the angle).
The initial conditions are :

```
% X
X0 = 0;
% Y rand
Y0 = 100 + randi([-3,3])*10;
% V rand
V0 = 20 + (rand-1/2)*10; %randi([-1,1])*10
% theta rand
theta0 = randi([-9,9])*pi/36;
% wing
wing0 = 2;
```

The Reward for the agent is set to RewardForLanding, which is defined like this :

RewardForLanding=State(1) / (State(2) + 0.001)

Our idea was that, the reward becomes greater for x increasing, and for y decreasing (the +0.001 was to prevent dividing by 0).
These are the result after an 100 episodes training :



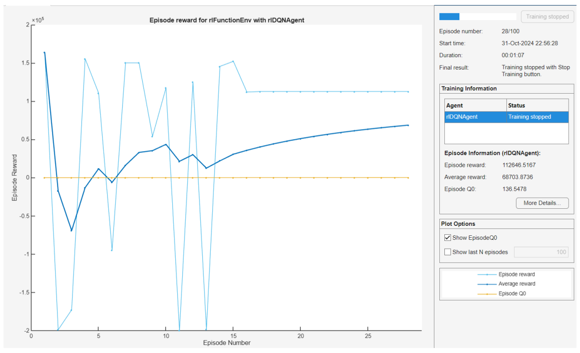
The obvious is that the agent learned to stay above the y=0 line, and crashed because he didn't have speed anymore.

The agent in this simulation is Glider1. The problem is that we have trouble creating a similar agent because of the initial conditions. Sometimes, he's just oscillating normally, because he won't have any penalty.

So I change theta to an angle between -30° and 0° for more consistency, and reduce y0 :

```
theta0 = randi([-30,0])*pi/180;
Y0 = 40 + rand*20
```

But now he learned to climb high and oscillating until the episode is finished :



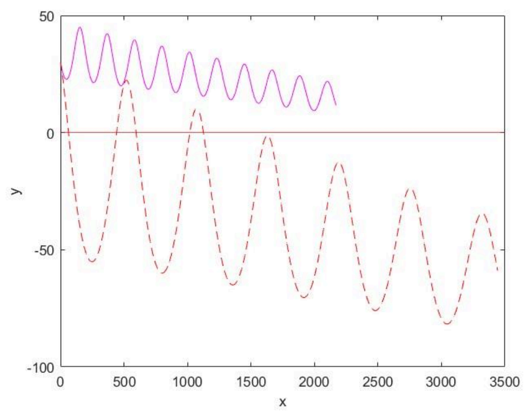
Episode number: 28/100
Start time: 31-Oct-2024 22:56:28
Duration: 00:01:07
Final result: Training stopped with Stop Training button.

Training Information

Agent	Status
rlDQNAgent	Training stopped

Episode Information (rlDQNAgent):
Episode reward: 112646.5167
Average reward: 68703.8736
Episode Q0: 136.5478

Plot Options
[✓] Show EpisodeQ0
☐ Show last N episodes 100



So our goal for the new version is to adjust the reward, so that being close to y=0 is more valuable than not crashing.

Secondly, we realized that the rng was fixed in the reset function. So Glider1 gives good results only around its training angle of -5°. Now we will train with real random angles.

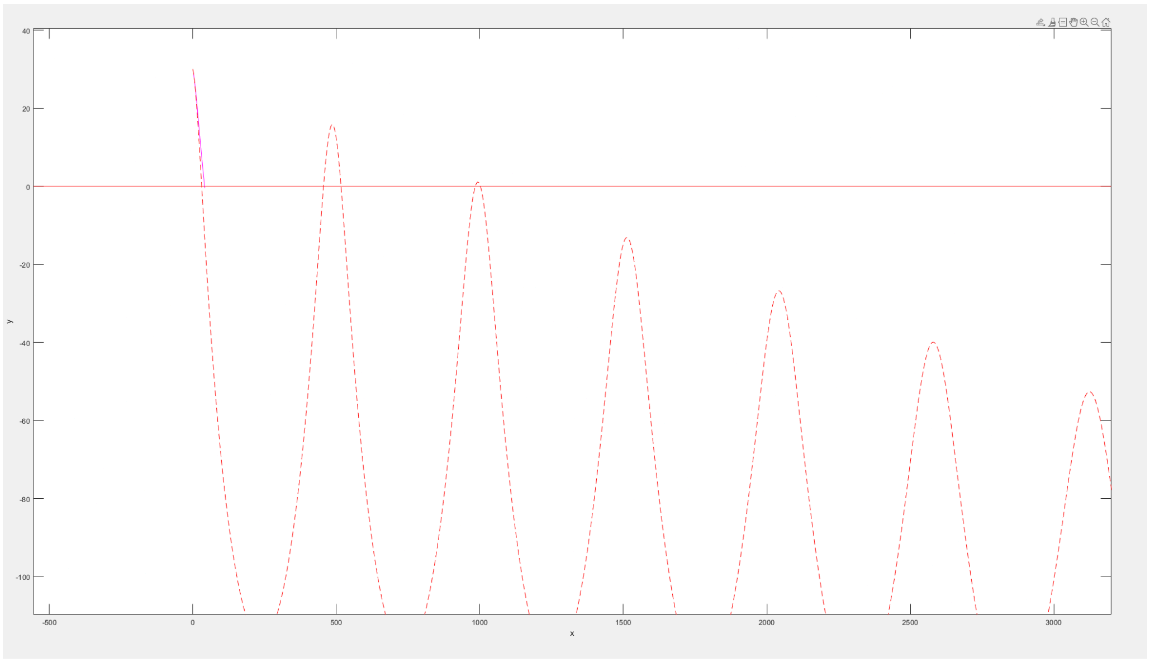
3) Adjusting the rewards

At this point began our journey to find potential solutions. We spent several hours trying differents reward possibilities :

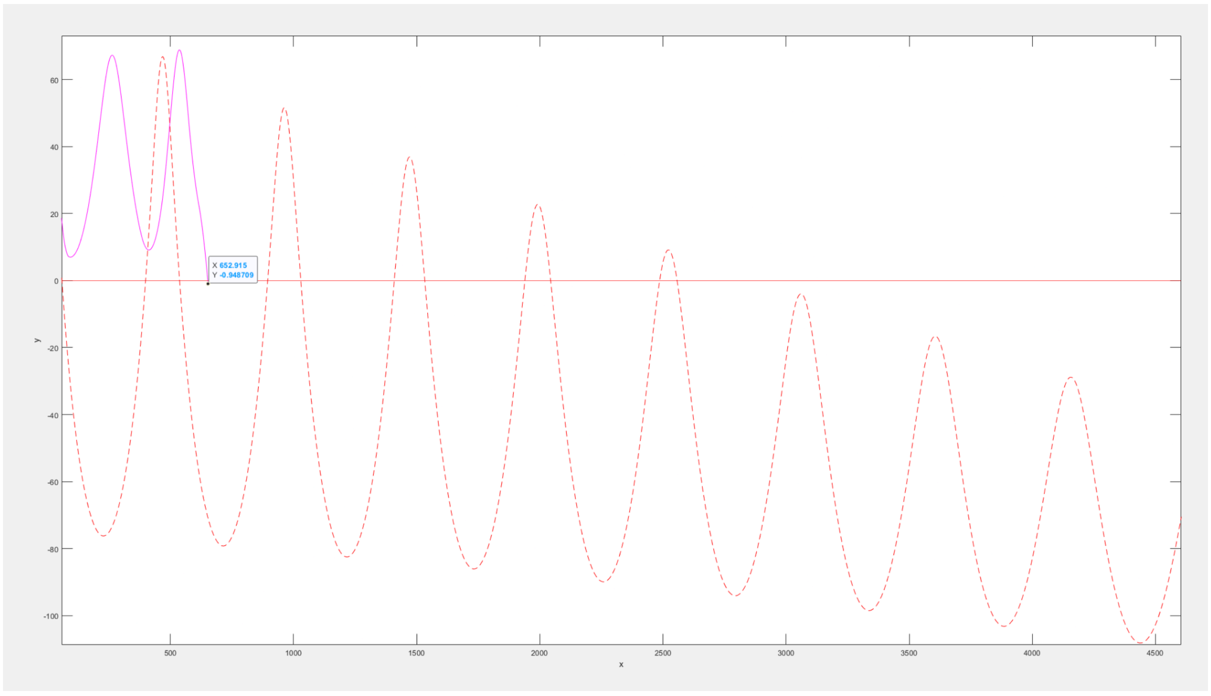
- Using a counter to value the time spent below a particular altitude
- A reward increasing exponentially when you get closer to y=0
- A reward increasing with the angle being close to zero

But each time, the glider won't stay at a fixed altitude.

The two following graphs are the results given by an agent trained with other rewards/ adjusted rewards.
The code is in the main branch in github : https://github.com/MaxBasse/Matlab/tree/master
The first one used a reward system that emphasized the "landing bonus" with an angle condition for the landing (i.e. the landing would be considered a crash if the angle was too steep) the neural network understood that and was trying to crash right away pointing the nose up at the last second.



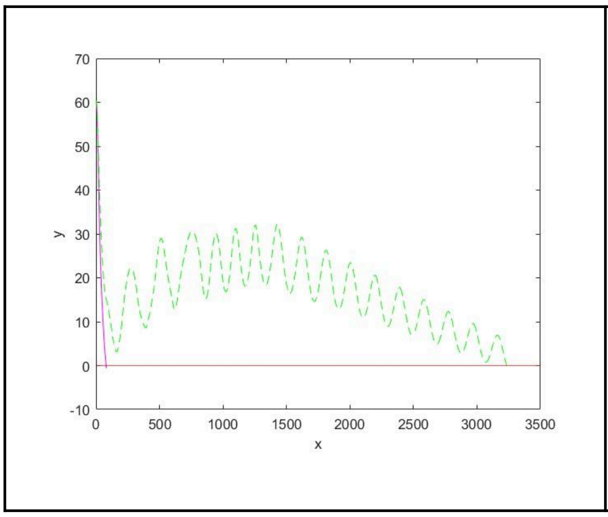
The second graphs shows a similar reward system but emphasizing the “precision” of the landing, a target was set and the reward was given proportionally to how close the landing was from the target, this seems to work relatively well but needs to be refined a bit since the precision still isn’t that great.

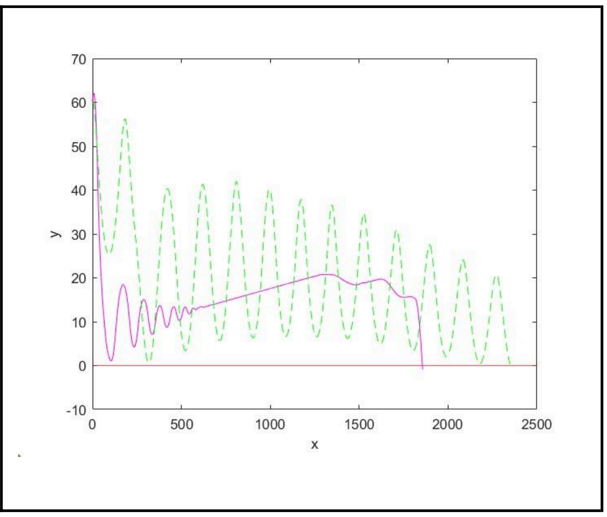


4) Using the angle

Like the example before, I tried to compute the reward regarding the distance traveled and the angle of the flight.

It doesn’t work every time, but the results are better than Glider1. Here is Glider1 in pink and Glider2 in green (both available in the “yann” branch : https://github.com/MaxBasse/Matlab/tree/yann)





New approach:

We will now try to force a straight trajectory, by using the angle as a terminal condition.