ML-Opt-Problem-Set

Calvin Ang, Jichuan Wu, Phoenix Tamaoki, Yuyang Gong

November 2023

In this problem set you will develop your own PINNs solution to 2-dimensional projectile motion with drag. The differential equation for drag is as follows

$$\frac{d^2\vec{s}}{dt^2} = -\mu \left\| \frac{d\vec{s}}{dt} \right\|_2 \frac{d\vec{s}}{dt} - \vec{g}$$

making this into a linear system we have that if the displacement in the x and y direction are s_x and s_y and the velocities in the x and y direction are v_x and v_y that the differential equation becomes

$$s_x'(t) = v_x(t) \tag{1}$$
$$s_y'(t) = v_y(t) \tag{2}$$
$$v_x'(t) = -\mu\sqrt{v_x(t)^2 + v_y(t)^2}v_x \tag{3}$$
$$v_y'(t) = -\mu\sqrt{v_x(t)^2 + v_y(t)^2}v_y - g. \tag{4}$$

where μ being the coefficient of drag and g being the acceleration of gravity, namely, $g = 9.8$.

1 Construct your data

1.1 (a) Get started

- Implement the equations of motion for a projectile based on the given equation. Hint:(`def proj_motion(y,t)`)
- Use the `odeint` function from the `scipy.integrate` module to solve the equations of motion for a projectile launched with an initial velocity of 10 m/s at an angle of 45 degrees. Using $g = 9.8$:

```
# Initial condition
y0 = [0,0,10,10]
# Time points
t = np.linspace(0, 0.85,500)

# Solve the ODE
solution = odeint(proj_motion, y0, t) # y0 solve range, t the linspace
```

1.2 (b) Sample your data with noise

Next, we will select our data. Take only a third of the data from the left-hand side of the data that you collected above and make that into your training data. You will also want to add in some Gaussian noise with values from 0.1 to -0.1.

2 Construct your solver

- Convert the training data (time,position,velocity) into PyTorch tensors.
- Implement a PINN model using PyTorch. The model should have an input layer with one neuron (representing time), five hidden layers each with 256 neurons, and an output layer with four neurons (representing the x and y positions and velocities). Use the `ReLU` activation function for the hidden layers.

3 Train your solver

- Set up the Adam optimizer with a learning rate of 0.0005. The optimizer will be used to adjust the parameters of the PINN model to minimize the loss function.
- Implement a training loop that runs for a specified number of epochs. In each epoch, perform the following steps:
  - Zero the gradients of the model parameters.
  - Compute the model's predictions for the training data and calculate the data mismatch loss.
  - Compute the model's predictions for a set of "physics" points and calculate the physics-informed loss. The physics-informed loss ensures that the learned solution obeys the physics (the equations of motion).
  - Add the data mismatch loss and the physics informed loss to get the total loss.
  - Backpropagate the loss and update the model parameters using the optimizer.
- Plot the loss as a function of epoch number.

4 Test your solver

- Generate a set of test data using the same method as the training data.
- Plot the test data and the learned solution, what have you discovered?

5 Solutions

We see that in the plain neural network case, the predicted path is not able to extrapolate anything meaningful from the data, however, given the physics loss term we see that the PINNs algorithm is able to learn a close approximation to the actual projectile motion.

For the code solution see `pset_sol.ipynb` in the GitHub repository along with the graphs `pset_nn_sol.jpeg` and `pset_PINN_sol.jpeg`. Project repo