Report - Problem 3
Neural differential equations, model discovery, and transfer learning

Guido Putignano

June 26, 2024

1 Task 1

The purpose of task 1 was to train an NN to control an inverted pendulum. In this case, the challenge also involved learning how to use Google JAX.

1.1 Solving the coupled ODE system

This task has been possible using an integrator found in the tutorial named as Tsit5 and the ODE that was possible following the obtaining this function.

```
# Define the system of ODEs
def pendulum_ode(t, y, args):
    x, x_dot, theta, theta_dot = y
    x_ddot = (F(t) - m * g * jnp.cos(theta) * jnp.sin(theta) + m * l * theta_dot**2 * jnp.sin(theta)) / (M + m - m * jnp.cos(theta)**2)
    theta_ddot = (g * jnp.sin(theta) - x_ddot * jnp.cos(theta)) / l
    return jnp.array([x_dot, x_ddot, theta_dot, theta_ddot])
```

F was depending on time.

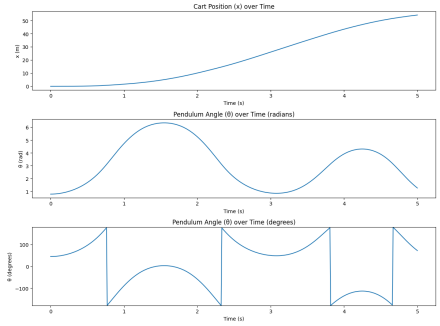


Figure 1: Plot for the first task

1.2 Learning to balance the pendulum

In the second case, multiple libraries have been used to understand which one would have been the most effective. After using stax and other alternatives, but equinox ended up being far faster and easier to use than others. As in the previous code, the angles have been changed from -180 to 180 degrees as without the transformation there was a particular change in the angle that could have been improved. In the assignment, it was written that "F(t) = NN(t, θ), where θ is the network's learnable parameters." It was possible to train the NN either using the parameters of the system, or the time. In both cases, the loss function was described as

```
np.mean(jnp.square(angle_error)) + 0.3 * jnp.mean(jnp.square(
    F_sol))
```

where angle error only considers the last quarter.

It's possible to notice some differences such as the cart position may be too far given forces are too strong towards one direction. However the final result tends to be acceptable in both cases.

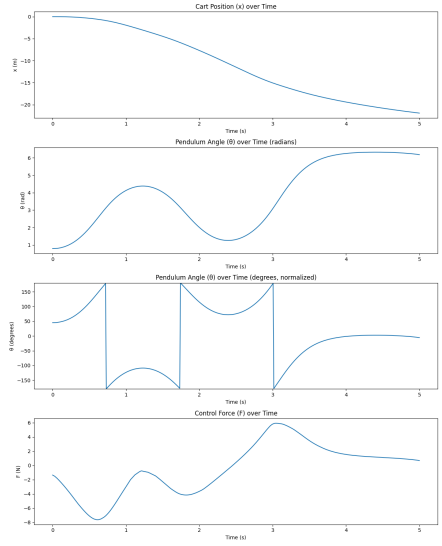


Figure 2: Plot for the force considering the four states

When there is only the time taken into account, it may be noticeable that the cart is harder to control. Indeed, with the original four inputs (space, velocity, angular position and angular velocity) it's easier to control the cart.

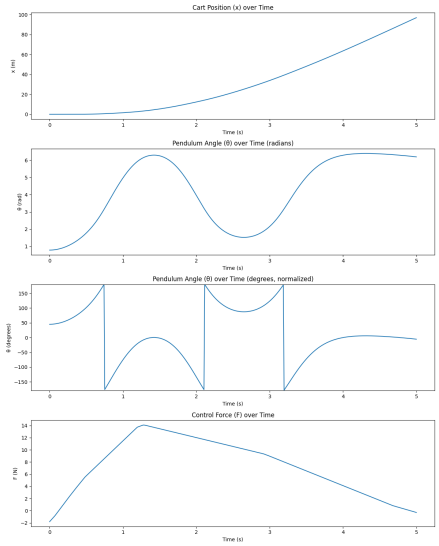


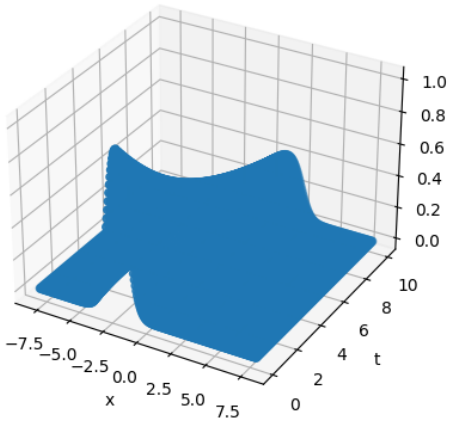
Figure 3: Plot for the force considering time

2 Task 2

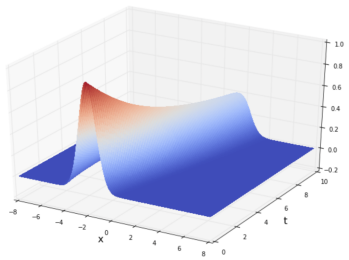
In Task 2, the objective was to use symbolic regression to find a mathematical expression that fits the data. The task has been divided into: - visualise the data to understand what type of function it was - calculate the underlying PDE

2.1 Data Visualisation

First Dataset



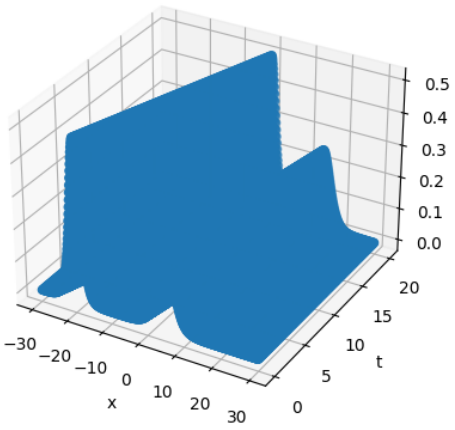
(a) Numerical Solution first dataset



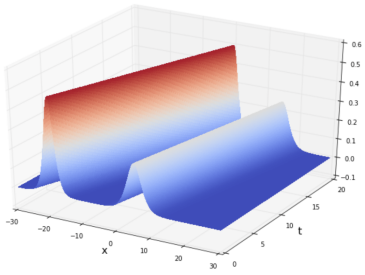
(b) Numerical Solution Burger's equation

Figure 4: Comparison between datasets

Second Dataset



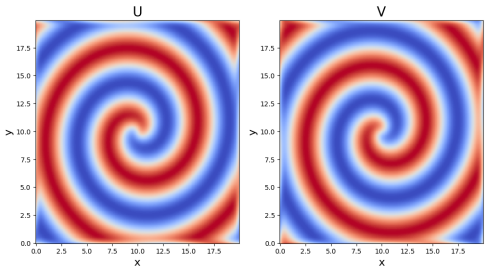
(a) Numerical Solution second dataset



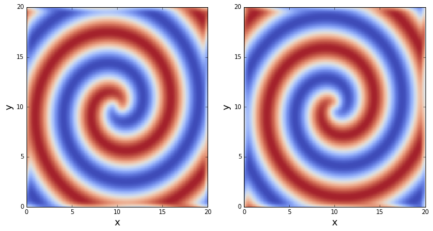
(b) Numerical Solution Korteweg–De Vries equation

Figure 5: Comparison between datasets

Third Dataset



(a) Numerical Solution third dataset



(b) Numerical Solution Reaction-Diffusion equations

Figure 6: Comparison between datasets

2.2 Solving the PDE

To solve the PDE using PDE-FIND there were many available online tutorials. In this case, it was possible to have a ground truth understanding of how the results would have been far from what we were calculating. In this case, the comparison was with and without added noise. Adding noise was useful to understand if the algorithm performance would have changed increasing the error. A robust algorithm would have avoided this problem. Moreover, when considering the PDE, the convergence was given by the grid and the number of points we were taking into account. For example, there was a subsampling of points in the reaction and diffusion equations to obtain the desired PDE. Moreover, given we are employing a linear regression, it was possible that the best model would not have been the right one, as there are multiple close models. For this reason, having a ranking of potential PDE would have also been beneficial.

2.2.1 Burger's equation

Real Equation

$$u_t + uu_x = u_{xx}$$

Calculated Equation

$$u_t = (-1.000403 + 0.000000i)uu_x + (0.100145 + 0.000000i)u_{xx}$$

Calculated Equation with added noise

$$u_t = (-1.007779 + 0.000000i)uu_x + (0.103338 + 0.000000i)u_{xx}$$

2.2.2 KdV's equation

Real Equation

$$u_t + 6uu_x + u_{xxx} = 0$$

Calculated Equation 1

$$u_t = (-5.955399 + 0.000000i)uu_x + (-0.987754 + 0.000000i)u_{xxx}$$

Calculated Equation 2

$$u_t = (-6.173902 + 0.000000i)uu_x + (-1.137839 + 0.000000i)u_{xxx}$$

2.2.3 Reaction-Diffusion Equations

Original Equations

$$\begin{aligned} u_t &= 0.1\nabla^2 u + \lambda(A)u - \omega(A)v \\ v_t &= 0.1\nabla^2 v + \omega(A)u + \lambda(A)v \\ A^2 &= u^2 + v^2, \quad \omega(A) = -\beta A^2, \quad \lambda(A) = 1 - A^2 \end{aligned}$$

Calculated Equations

$$\begin{aligned} u_t = {} & (0.099972 + 0.000000i)u_{xx} + (0.100004 + 0.000000i)u_{yy} \\ & + (0.999913 + 0.000000i)u + (0.999979 + 0.000000i)v^3 \\ & + (-0.999882 + 0.000000i)uv^2 + (0.999977 + 0.000000i)u^2 v \\ & + (-0.999900 + 0.000000i)u^3 \\ v_t = {} & (0.100032 + 0.000000i)v_{xx} + (0.099968 + 0.000000i)v_{yy} \\ & + (1.000278 + 0.000000i)v + (-1.000293 + 0.000000i)v^3 \\ & + (-0.999971 + 0.000000i)uv^2 + (-1.000268 + 0.000000i)u^2 v \\ & + (-0.999978 + 0.000000i)u^3 \end{aligned}$$

Calculated Equations with added noise

$$
\begin{aligned}
u_t = {} & (0.094923 + 0.000000i)u_{xx} + (0.095022 + 0.000000i)u_{yy} \\
& + (0.945835 + 0.000000i)u + (0.999732 + 0.000000i)v^3 \\
& + (-0.945751 + 0.000000i)uv^2 + (0.999784 + 0.000000i)u^2v \\
& + (-0.945507 + 0.000000i)u^3 \\
v_t = {} & (0.094980 + 0.000000i)v_{xx} + (0.094898 + 0.000000i)v_{yy} \\
& + (0.946891 + 0.000000i)v + (-0.946863 + 0.000000i)v^3 \\
& + (-0.999398 + 0.000000i)uv^2 + (-0.946404 + 0.000000i)u^2v \\
& + (-0.999661 + 0.000000i)u^3
\end{aligned}
$$

3 Bonus

The purpose of the task was to gain practical experience in the full pipeline of ML for solving PDE – this will include a trip into data generation, training, and fine-tuning. After implementing stochastic samplers and using a different finite solver, it was possible to perform zero-shot evaluations and evaluations after fine-tuning your model.

3.1 Implementing stochastic samplers

Most of the code has been written from scratch for the implementation of stochastic samplers. However, the GP has been calculated using sklearn. Also, CP has been calculated using numpy polynomial chebyshev.

3.2 Training and NN

For the NN, the purpose was to create something not highly complex mostly to have practice on the challenge rather than achieving a particular target.

3.3 Results

In this case, the final result was

Train Class	**Eval Class**	**Relative L2 Distance**
CP	GP	0.9460
CP	PL	0.3222
GP	CP	0.7872
GP	PL	0.2323
PL	CP	0.4698
PL	GP	0.6167

Table 1: Relative L2 Distances between Train and Eval Classes