SF3: Machine Learning
Interim Report

Raghavendra Narayan Rao

May 22, 2025

Abstract

This report investigated the cart-pole system. In the first half, simulations of the system were carried out using Euler integration. The initial conditions resulting in oscillations and rotations were determined. In the second half, a linear model was built and used to forecast the behaviour of the system. The linear model could not predict the system's states 100 steps into the future.

1 Introduction

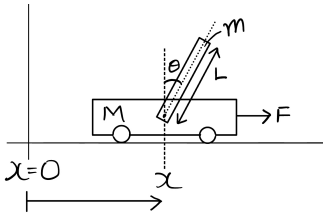

Figure 1: Cartpole (Pendulum on a Cart).

In this project, a cartpole system in Figure 1 is modelled and then used to control the system. The cartpole system behaves according to the following equations:

$$\begin{aligned} 3\ddot{x}\cos\theta + 2L\ddot{\theta} &= 3g\sin\theta - 6\mu_\theta\dot{\theta}/mL, \\ (m+M)\ddot{x} + \tfrac{1}{2}mL\ddot{\theta}\cos\theta - \tfrac{1}{2}mL\dot{\theta}^2\sin\theta &= F - \mu_x\dot{x}. \end{aligned} \tag{1}$$

The interim report focuses on using a linear model to describe the cartpole system.

2 Task 1

2.1 Task 1.1: Dynamical Simulation

To carry out a dynamical simulation, the following two conditions must be established:

1. **Choice of a State**
   The state, $X = [x, \dot{x}, \theta, \dot{\theta}]$, has been chosen for this system. This choice is not arbitrary, as from the equations of motion (Equation 1), the states of the system obey the *Markov property*: the future state is independent of the previous states, given the current state.

2. **Time Invariance**
   We have assumed that the dynamics of the system remain unchanged regardless of when we start the simulation. This may not be the case in real life, where due to wear and tear, the coefficients of friction increase, thereby altering the system's dynamics.

A rollout was simulated for some initial conditions that resulted in oscillations and rotations. The code for performing a rollout is provided in the appendix (Listing 1).

2.1.1 Simple Oscillation

To produce a simple oscillation, an initial state $X_0 = [0, 1, \pi, 8]$ was used. From Figure 2, we can conclude that this initial state results in a periodic behaviour in all four state variables. Furthermore, the phase plots in Figure 3 form a closed loop, providing further evidence of a simple oscillation. The cart's phase diagram (left plot of Figure 3) shows oscillation about the stable equilibrium ($x = 0, \dot{x} = 0$). The pole's phase diagram (right plot of Figure 3) shows oscillation about ($\theta = \pm\pi, \dot{\theta} = 0$) if we tessellate the phase plot to the left and right to account for angle remap.

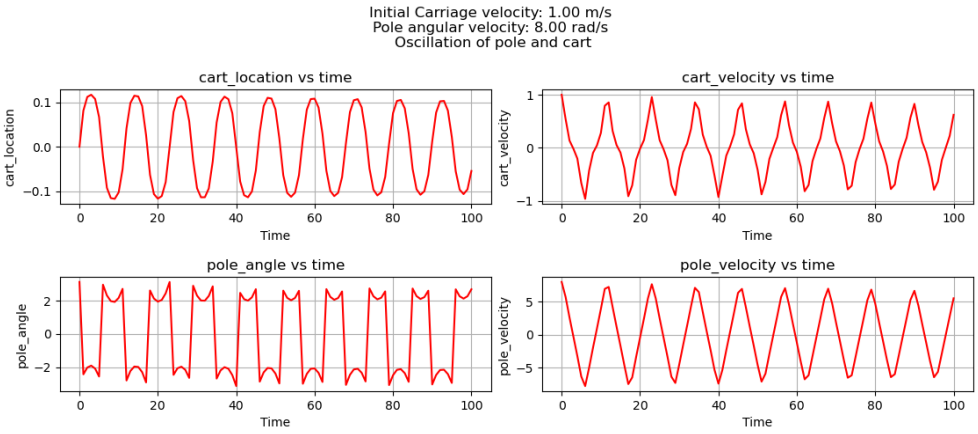

Figure 2: Transient response (Simple Oscillation).

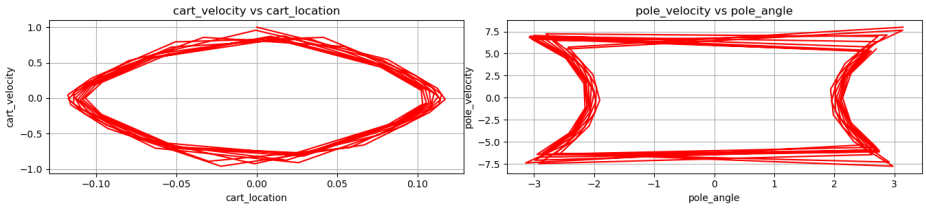

Figure 3: Phase Diagram (Simple Oscillation).

2.1.2 Complete revolution

To produce a revolution, an initial state $X_0 = [0, 10, \pi, 14]$ was used. From the pole angle plot (bottom left of Figure 4), the first 25s show 2 revolutions. Furthermore, the pole phase plot (right plot in Figure 5) shows an unstable equilibrium point $\theta = 0$ (pole is upright).
To further analyse the initial conditions required for a complete revolution, a heatmap, Figure 6, has been plotted where $\dot{x}$ and $\dot{\theta}$ are varied and the number of revolutions is indicated by colour. The vertical contours

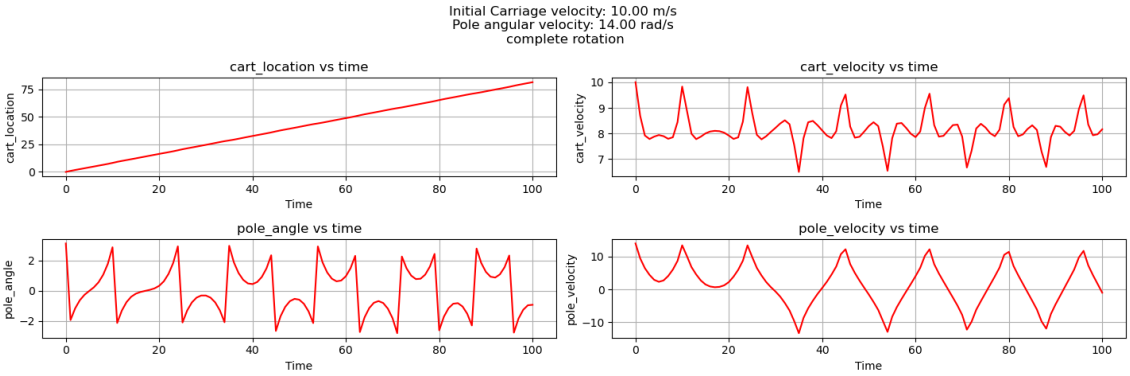

Figure 4: Transient response (Complete Revolution).

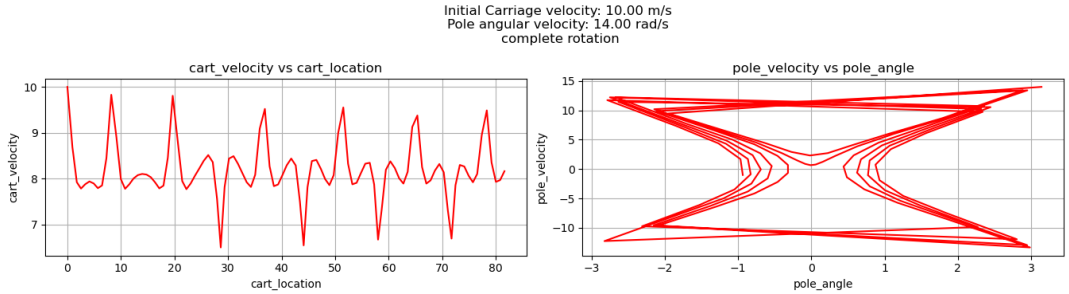

Figure 5: Phase Diagram (Complete Revolution).

indicate that only the pole velocity dictates if a revolution occurs when starting from the equilibrium position and angle. The minimum angular velocity required for a revolution is about 13.85 rad/s.

2.2 Task 1.2: Change of State

2.2.1 Validity of linear assumption

Note that the valid ranges; Cart velocity: $[-10, 10]$, pole angle: $[-\pi, \pi]$, pole (angular) velocity: $[-15, 15]$. Scans of all four variables were performed, producing 4 set of states. The set produced from scanning across cart location is $\{[a_n, r_1, r_2, r_3]\}_{n=1}^{100}$. Values of a_n are equally spaced within their valid range. r_1, r_2, r_3 are randomly sampled from their respective valid ranges.

A new output $Y = X(T) - X(0)$ (change in state) is defined where T is the time corresponding to a single 'PerformAction' function, set at 0.1s and represents 50 updates to the state with Euler integration. If T is infinitesimally small, then $Y \to C$ for some constant C. At $T = 0.1$, only the change in cart position, Y_x and change in velocity $Y_{\dot{x}}$ exhibit a linear (constant) relation (top of Figure 7). This can be understood from the equations of motion (Equation 1) where $T = 0.1$ is too large to make $\sin\theta$ and $\cos\theta$ linear.
The code for performing a sweep is provided in the appendix (Listing 2).

2.2.2 Variable that has no effect on next step (Y)

The dynamics of the cartpole system is invariant to the location of the pole. This can be shown in two ways. Firstly, the equations of motion (Equation 1) has no explicit dependence on x. Secondly, the Lagrangian does not depend on x as x does not affect the system's potential energy. Therefore, Noether's theorem can

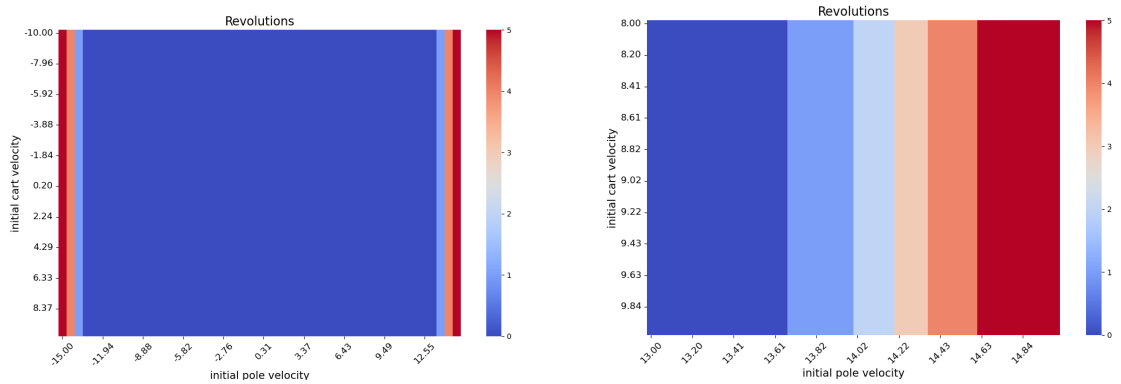

Figure 6: Heatmap of number of revolutions. A fullscale map is on the left and a zoomed map is on the right
.

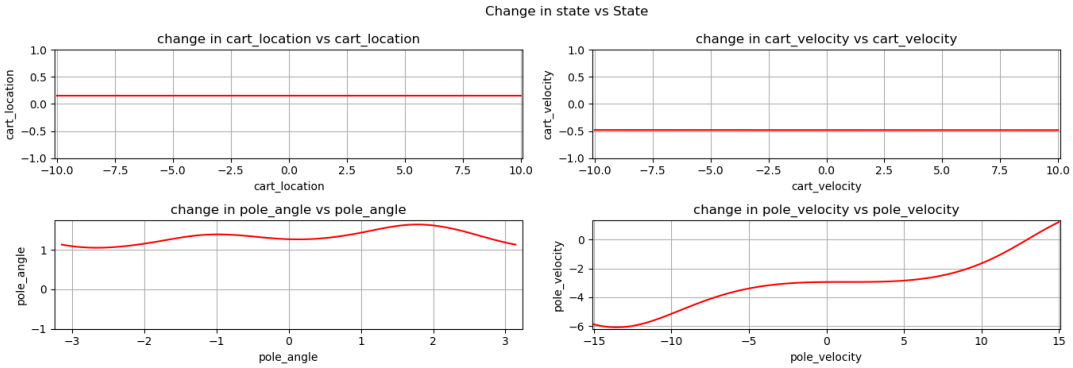

Figure 7: Relation between Y and X.

be invoked to state that the system is symmetric (invariant) to x. This also results in a conservation law.

Experimentally, this is determined from the controur plots of change in states. The contours are always parallel to the axis of cart location, indicating that cart location does not affect the change in state. Figure 8 shows a contour plot that is parallel to the location of the cart (left) and contour plots that are not parallel to other variables (middle and right) for comparison.

The cart location however does change with the dynamics so it is correlated with the change in state so how can cart location not affect the change in state? Conditional independence can resolve this. Conditioned on the other states, the change in state is independent of cart location.

2.3 Task 1.3: Train Linear Model

2.3.1 Methodology

500 datapoints were collected, $\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^{500}$, where $\mathbf{x}_n$ are randomly initialised states and $\mathbf{y}_n$ are change in states due to a single call to `performAction`. By appending all $\mathbf{x}_n$ and $\mathbf{y}_n$ as rows to form matrices $\mathbf{X}$ and $\mathbf{Y}$, the linear regression problem can be simplified to solving for the least squares solution $\hat{\mathbf{C}}$ to the equation $\mathbf{Y} = \mathbf{X}\mathbf{C}$.

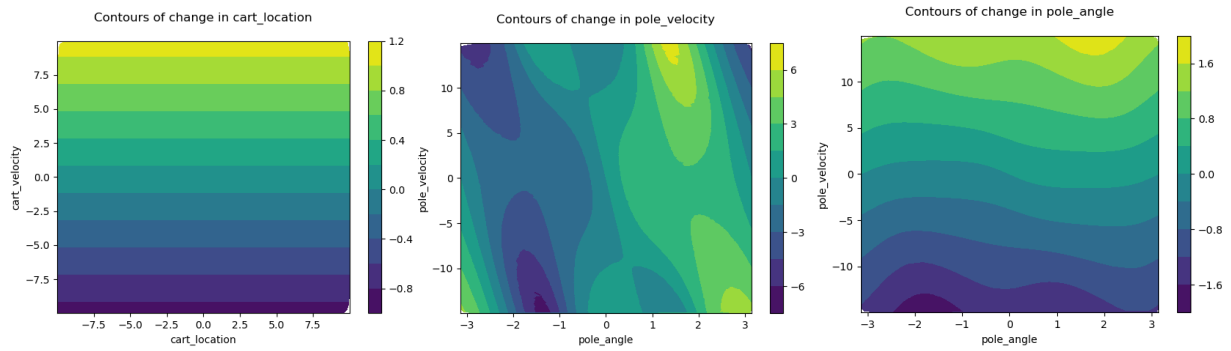

Figure 8: Various 2D slices of change in state (contour plots).

The exact form $\hat{\mathbf{C}}$ is $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$. However, the average condition number for $\mathbf{X}^T\mathbf{X}$ is 22.38 (after repeating data collection 30 times). As a rule of thumb [1], $\mathbf{X}^T\mathbf{X}$ is not a well-conditioned matrix as its condition number is larger than 10 (much larger than 1). Therefore, `np.linalg.lstsq`, which does not take matrix inverses, is used instead to solve for $\hat{\mathbf{C}}$. The code for performing linear regression is provided in the appendix (Listing 3).

2.3.2 Predicting one step ahead

The same scans from Task 1.2 are used to forecast the states one step ahead. The angle (and its velocity) are inherently non-linear so it is expected that a linear model will struggle more with predicting these values. The top four plots in Figure 9 show that the actual and predicted states match very closely with the linear model. However, the pole angle shows a large discrepancy in the predicted and actual states due to angles being remapped. The actual angle has a (discontinuity) jump from π to $-\pi$ so the linear model approximately plots a best fit line across this jump. The bottom four plots in Figure 9 depict a very similar story where the predicted states and the actual states match a $y = x$ line, indicating a very good fit, except for the pole angle due to remap.

Also, note that the higher order derivatives contain higher frequency terms in their fourier transforms so they should be harder to predict with a linear model. This is also observed in Figure 9 as the cart location has a better match than cart velocity.

Figure 10 compares the direct output of the linear model against actual changes in states. This provides a different perspective on why the linear model fails. The actual plots for cart location and velocity are both linear so the linear model is able to predict this accurately. However, the angle plot contains discontinuities while the angular velocity is periodic so the linear model fails at predicting these trends.

2.3.3 Reducing time step to improve prediction

From the equations of motion, we know that the states are at least twice differentiable (physically, we know that they are infinitely differentiable). From a linear approximation of taylor series, when the simulation time step approaches 0, the predicted state after one time step should match the actual state. By reducing `self.delta_time` to 0.0001, the predicted and actual states have again been plotted in Figure 11. Now, the linear model much better predicts states one step ahead, with a perfect match.

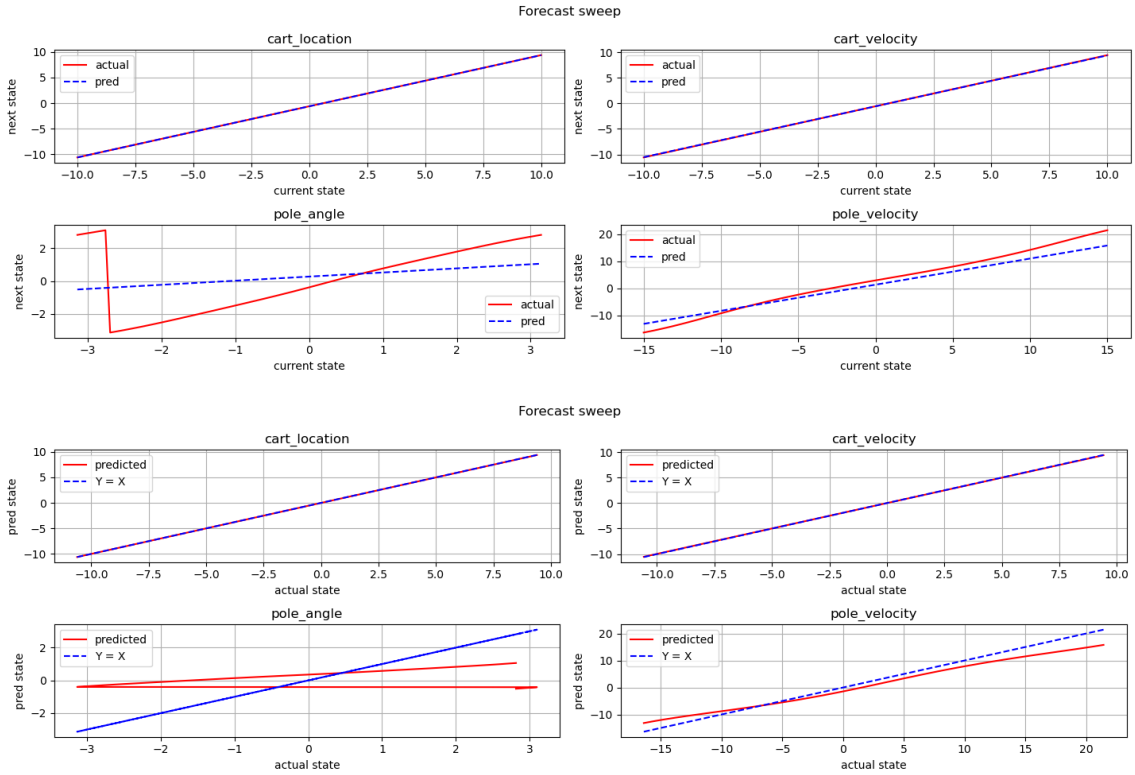

Figure 9: Single step forecast from scanning data with time step at 0.1s

2.4 Task 1.4: Forecast using Linear Model

2.4.1 Methodology

In this task, a forecast of 100 steps into the future is carried out with initial points that result in a simple oscillation as well as complete revolutions. The update equation used for the states is $\mathbf{X}_{n+1} \leftarrow \mathbf{X}_n + \mathbf{X}_n\hat{\mathbf{C}}$. Note that an additional remap of the angle is carried out after every iteration to ensure the angle stays within $[-\pi, \pi]$. The angle diverges if remap is not carried out (for example in complete revolution). In real dynamics, the sine or cosine of this angle is taken so it does not affect the other state variables. However, in the linear model, each state variable is a linear combination of the state variables of the past state. This results in state variables diverging if the angle diverges. Mathematically,

$$\begin{aligned}\lim_{\theta_n \to 0} x_{n+1} &= \lim_{\theta_n \to 0} \hat{C}_{1,1}x_n + \hat{C}_{2,1}\dot{x}_n + \hat{C}_{3,1}\theta_n + \hat{C}_{4,1}\dot{\theta}_n \\ &= \infty \text{ (as } \hat{C}_{3,1} \neq 0)\end{aligned}$$

2.5 Predicting 100 steps ahead

The linear model forecasts poorly for 2 different initial conditions (plotted in Figure 12. There are two reasons for observed behaviour. Firstly, the slight discrepancy from a single step as observed in Figure 9 adds up quickly and hence the predicted behaviour quickly diverges from the system. Secondly, the linear model inherently is unable to generalise well due to the non-linearities in the true dynamics so it cannot forecast well far into the future. The code for forecasting is provided in the appendix (Listing 4).

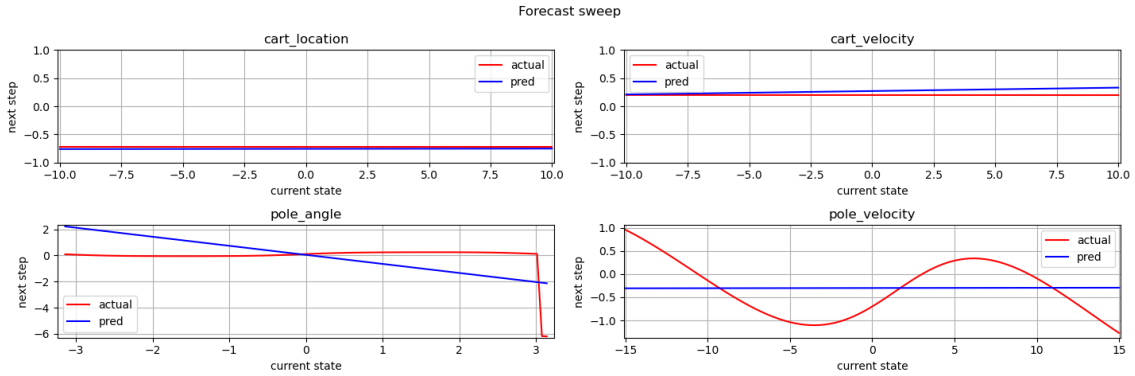

Figure 10: Output of linear model vs actual rollout from scanning data with time step at 0.1s

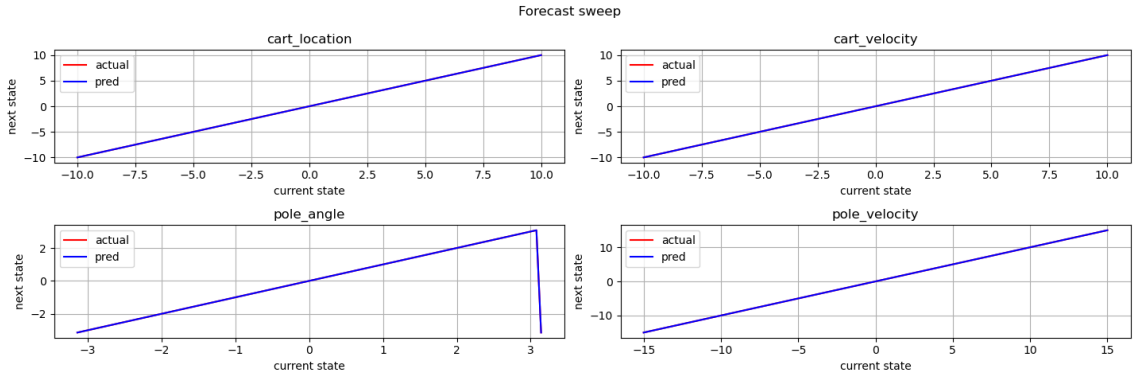

Figure 11: Single step forecast from scanning data with time step at 0.0001s

Similar to Task 1.3, after reducing the time step, the forecast for simple oscillation is plotted in Figure 13. The forecast of location and angle has improved significantly but the forecast of the higher derivatives still remains poor.

3 Conclusions

In Task 1.2, it was observed that the change in state is not linear with respect to the state. Only change in cart location and velocity are linearly related to the original states while change in angle and angular velocity are not. In Task 1.3, a linear model was built to predict the change in state. The model was successful in predicting a single step ahead. When the time step was further reduced, the linear model matched the data perfectly. However, when forecasting 100 steps ahead in Task 1.4, the linear model performed poorly.

References

[1] *Matrix Condition Number.* Nist.gov, 2025 (Accessed 21 May 2025). Available at www.itl.nist.gov.

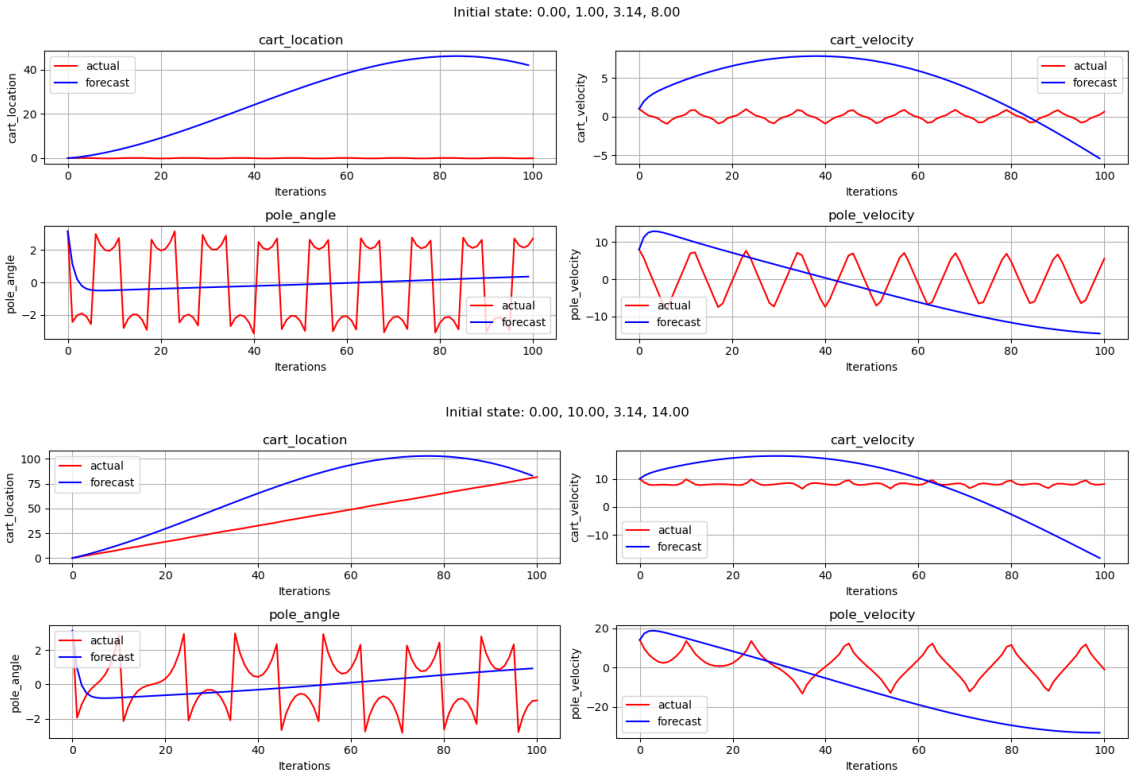

Figure 12: Forecast of cart and pole oscillation (top) and complete rotation (bottom)

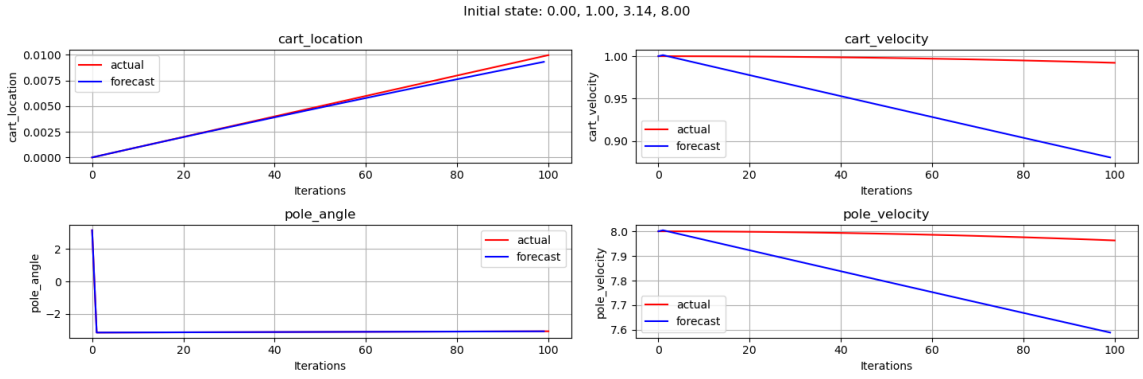

Figure 13: Forecast of cart and pole oscillation with very low time step

4 Appendix

```python
def rollout(initial_state, initial_force, num_steps, visual=True):
    """
    Simulate the CartPole environment for a given number of steps.

    Args:
        initial_state (tuple): The initial state of the environment.
        it should be a tuple of the form (cart_location, cart_velocity,
                                          pole_angle, pole_velocity).

        initial_force (float): The initial force applied to the cart.
        num_steps (int): The number of steps to simulate.

    Returns:
        data: A dictionary containing the cart location, cart velocity,
              pole angle and pole angular velocity at each step.
    """
    env = CartPole(visual=visual)
    env.reset()

    data = {'cart_location': [],
            'cart_velocity': [],
            'pole_angle': [],
            'pole_velocity': []
        }

    # Set the initial state
    env.setState(initial_state)

    # Perform the action for the specified number of steps
    for step in range(num_steps + 1):
        # Store the current state
        data['cart_location'].append(env.cart_location)
        data['cart_velocity'].append(env.cart_velocity)
        data['pole_angle'].append(env.pole_angle)
        data['pole_velocity'].append(env.pole_velocity)

        # Perform the action
        env.performAction(initial_force)

        # remap the angle to be between -pi and pi
        env.remap_angle()

    # close the plot
    if visual:
        env.close_plot()
        plt.close()

    return data
```

Listing 1: Rollout

```python
def difference_sweep(num_steps, initial_force):
    """
    Perform a sweep of the CartPole environment for a given number of steps.

    Args:
        num_steps (int): The number of steps to simulate.
        initial_force (float): The initial force applied to the cart.

    Returns:
        None: plots relation between current state and state after performing
        action
    """
    state_sweeps = {
        'cart_location': np.linspace(-10, 10, num_steps),
        'cart_velocity': np.linspace(-10, 10, num_steps),
        'pole_angle': np.linspace(-np.pi, np.pi, num_steps),
        'pole_velocity': np.linspace(-15, 15, num_steps)
    }

    titles = ['cart_location', 'cart_velocity', 'pole_angle', 'pole_velocity']

    env = CartPole(visual=False)

    # this makes angle a lot more linear
    # env.setSimParams(sim_steps=5, delta_time=0.02)

    state = [np.random.uniform(-10, 10), np.random.uniform(-10, 10),
             np.random.uniform(-np.pi, np.pi), np.random.uniform(-15, 15)]

    y_data = {
        'cart_location': [],
        'cart_velocity': [],
        'pole_angle': [],
        'pole_velocity': []
    }

    for i in range(len(state)):
        title = titles[i]
        for j in state_sweeps[title]:
            state[i] = j
            env.reset()
            env.setState(state)
            env.performAction(initial_force)
            new_state = env.getState()
            y_data[title].append(new_state[i] - state[i])

        # plot_x_y(state_sweeps[title], y_data[title], title, "change in " + title)

        # reset state to random values
        state = [np.random.uniform(-10, 10), np.random.uniform(-10, 10),
             np.random.uniform(-np.pi, np.pi), np.random.uniform(-15, 15)]

    plot_state(state_sweeps, y_data)
```

Listing 2: Sweep

```python
def generate_data_random(num_steps, initial_force):
    env = CartPole(visual=False)
    env.reset()
    x_data = {
        'cart_location': [],
        'cart_velocity': [],
        'pole_angle': [],
        'pole_velocity': []
    }
    y_data = {
        'cart_location': [],
        'cart_velocity': [],
        'pole_angle': [],
        'pole_velocity': []
    }
    for i in range(num_steps):
        initial_state = [np.random.uniform(-10, 10), np.random.uniform(-10, 10),
                         np.random.uniform(-np.pi, np.pi), np.random.uniform(-15, 15)]
        env.reset()
        env.setState(initial_state)
        env.performAction(initial_force)

        # remap the angle to be between -pi and pi
        env.remap_angle()

        next_state = env.getState()

        x_data['cart_location'].append(initial_state[0])
        x_data['cart_velocity'].append(initial_state[1])
        x_data['pole_angle'].append(initial_state[2])
        x_data['pole_velocity'].append(initial_state[3])

        y_data['cart_location'].append(next_state[0] - initial_state[0])
        y_data['cart_velocity'].append(next_state[1] - initial_state[1])
        y_data['pole_angle'].append(next_state[2] - initial_state[2])
        y_data['pole_velocity'].append(next_state[3] - initial_state[3])

    X = convert_dict_to_array(x_data)
    Y = convert_dict_to_array(y_data)
    return X, Y

def linear_regression(X, Y, intercept=False):
    """
    Args:
        X (numpy.ndarray): The input data.
        Y (numpy.ndarray): The output data.

    Returns:
        W (numpy.ndarray): The parameters of the model
        Y_pred (numpy.ndarray): The predicted output data
    """
    # Add a column of ones to X for the intercept term
    if intercept:
        X = np.hstack((np.ones((X.shape[0], 1)), X))

    # Calculate the weights using the normal equation
    # W = np.linalg.inv(X.T @ X) @ X.T @ Y
    W = np.linalg.lstsq(X, Y)[0]

    # Make predictions
    Y_pred = X @ W

    return W, Y_pred
```

Listing 3: Linear regression

```python
def forecast(initial_state, num_steps, W):
    """
    Forecast the future state of the CartPole environment using the learned model.

    Args:
        initial_state (list): The initial state of the environment.
        num_steps (int): The number of steps to forecast.
        W (numpy.ndarray): The learned model weights.

    Returns:
        X_forecast (dict): A dictionary containing the forecasted states.
    """

    # initialise the forecasted state
    X_forecast = {
        'cart_location': [],
        'cart_velocity': [],
        'pole_angle': [],
        'pole_velocity': []
    }

    state = np.array(initial_state)

    # perform the action for the specified number of steps
    for i in range(num_steps):
        X_forecast['cart_location'].append(state[0])
        X_forecast['cart_velocity'].append(state[1])
        X_forecast['pole_angle'].append(state[2])
        X_forecast['pole_velocity'].append(state[3])

        # forecast the next state
        state = state + (state @ W)

        # remap the angle to be between -pi and pi
        state[2] = remap_angle(state[2])

    return X_forecast

# Example initial states for testing
initial_states = [[0, 0, np.pi, 5], [0, 0, np.pi, 14], [0, 1, np.pi, 8], [0, 10, np.pi, 14],
     [0, -2, 3, 0]]

# set the initial force to 0
initial_force = 0

# set the number of steps to forecast
num_steps = 100

# Forecast the future state of the CartPole environment using the learned model.
for initial_state in initial_states:
    # obtain the forecasted state
    X_forecast = forecast(initial_state, num_steps, W)

    # obtain the actual state
    X_actual = rollout(initial_state, initial_force, num_steps, visual=False)

    # plot with graph title as initial state formatted to 2 decimal places
    graph_title = f"Initial state: {initial_state[0]:.2f}, {initial_state[1]:.2f}, {
    initial_state[2]:.2f}, {initial_state[3]:.2f}"
    plot_data_vs_forecast_time(X_actual, X_forecast, graph_title=graph_title)
```

Listing 4: Forecast