SF3: Machine Learning Project IIA

Aditya Jain (aj563)

Christ’s College

18/05/2021

Final report

# Introduction

In this report I will model the dynamics of a cart with a pole attached, with known equations of motion. By scanning over initial conditions of variables, I will observe which variables affect the dynamics linearly. In the last 2 tasks, I will develop a simple linear regression model to predict the cart dynamics, evaluate which variables it can predict well and why, and forecast using 2 scenarios.

# Task 1

## Task 1.1

In this task, I wrote a function (start\_the\_cart) that plotted the 4 measures of cart dynamics over a specified number of steps. I was curious how the trajectories of the different cart dynamics (cart location, cart velocity, pole angle, pole velocity) will be affected by the initial conditions, so I incorporated that into the aforementioned function. Note that in this section, the cart dynamics will be represented in square brackets in this order:

[Cart location, Cart velocity, Pole angle, Pole velocity].

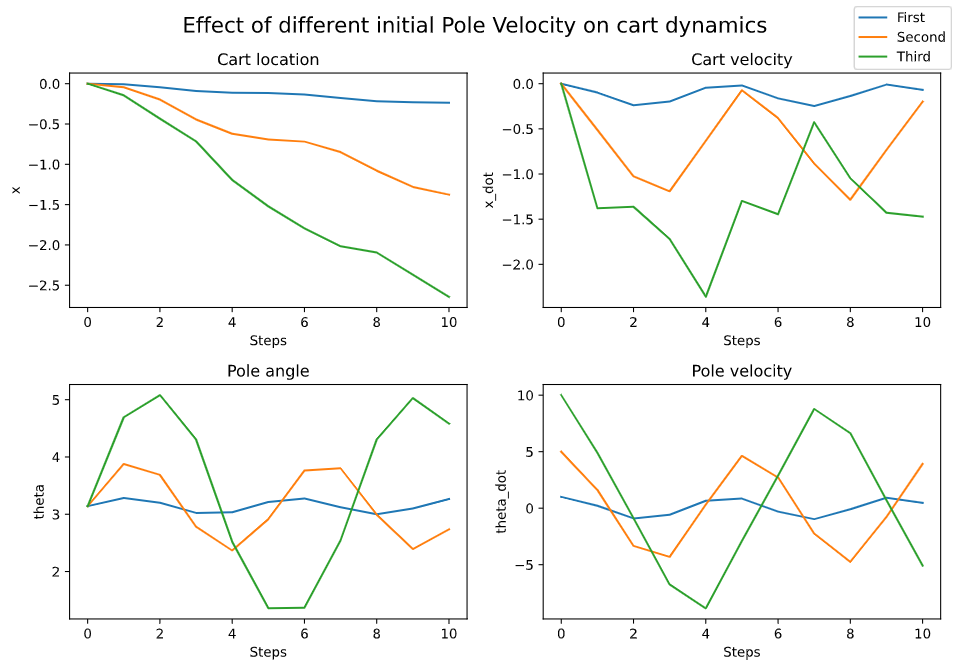


Figure . Cart dynamics (discreet) time evolution from a stable equilibrium. The different lines are different initial pole velocities.

Figure 1 shows how the cart behaves when at a stable equilibrium, [0,0,π,0], but the pole velocities are non-zero. The figure has the following initial conditions: First (blue) – [0,0,π,1]; Second (orange) - [0,0,π,5]; Third (green) - [0,0,π,10]. We see, as expected, the higher pole velocities lead to a larger change in cart location (supported by a higher cart velocity), and a large variation in pole angle. The function can also be called to vary a different variable. Note that the angle is not remapped. We see that in Figure 1, the pole angle never crosses 2π or 0, so I increased the initial conditions to [0,0,π,15]. This produced Figure 2, where we clearly see that multiple full rotations have occurred (from pole angle). Note that unlike Figure 1, the pole velocity is always positive as the pole never “falls back”.

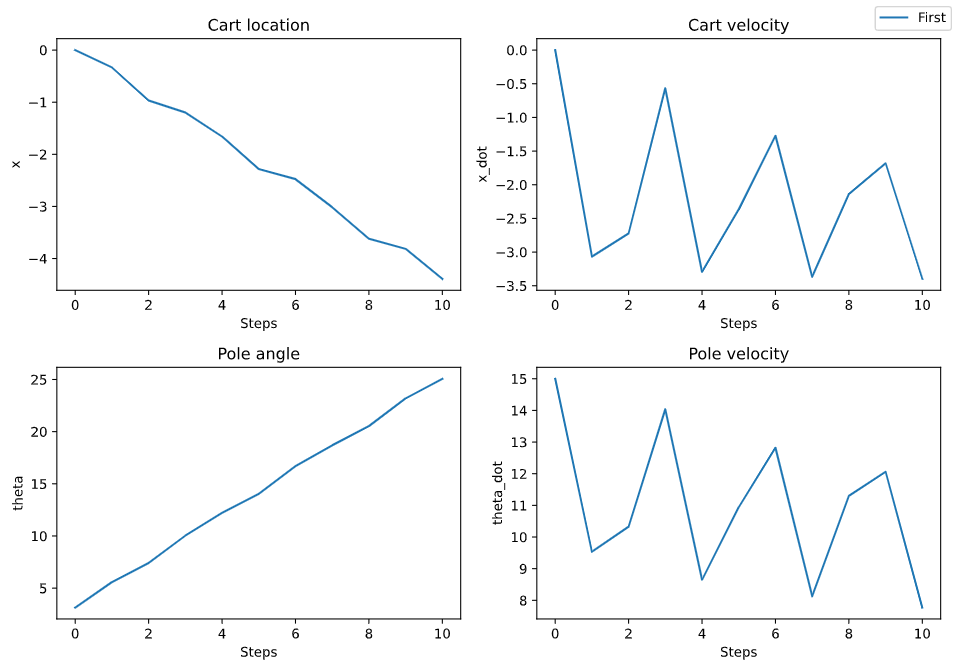


Figure . Cart dynamics after complete rotations of the pendulum.

## Task 1.2

I randomly initialized the starting conditions, [-0.24,-9.27,-1.07,9.09], and scanned over one variable at a time. I observed how the cart dynamics varied after one step with the (modified) initial conditions. Since one step didn’t cause a large change, the resulting plot was roughly linear. Figure 3 shows one of these plots, where I varied initial pole velocity.

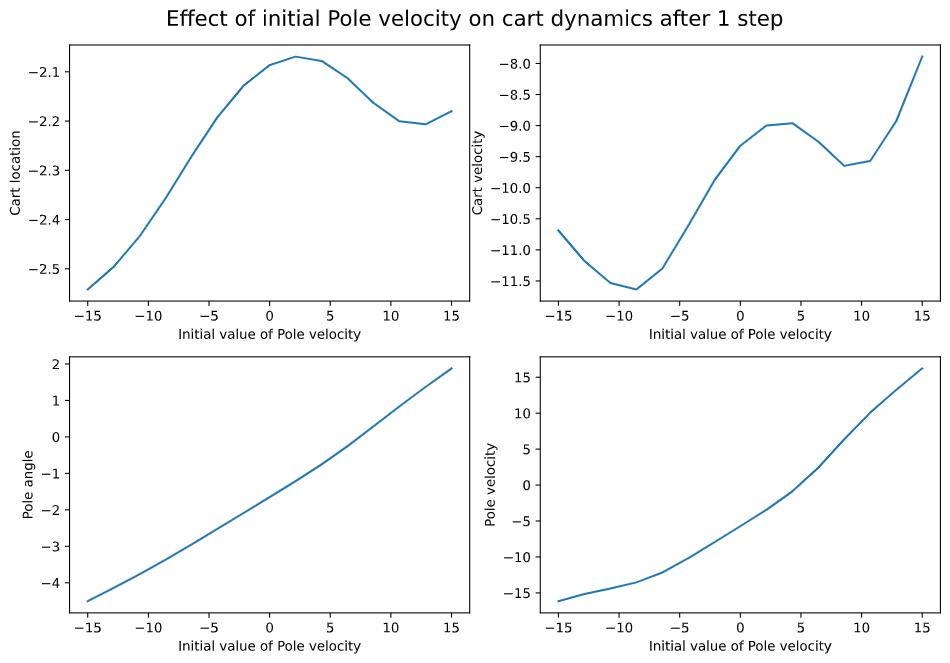


Figure . Cart dynamics after one step while varying initial pole velocity (and keeping other conditions constant).

We see the bottom-right graph in Figure 3 is roughly linear, as expected. To get a better idea of the effect of varying a single variable, we plot the *change* in cart dynamics after 1 step. Figure 4 shows a similar graph to Figure 3, but now the vertical axis displays the difference instead of the next value. We see that the dynamics depend non-linearly on the pole angle and velocity. We can also see that cart location doesn’t affect the next step (top left graph).

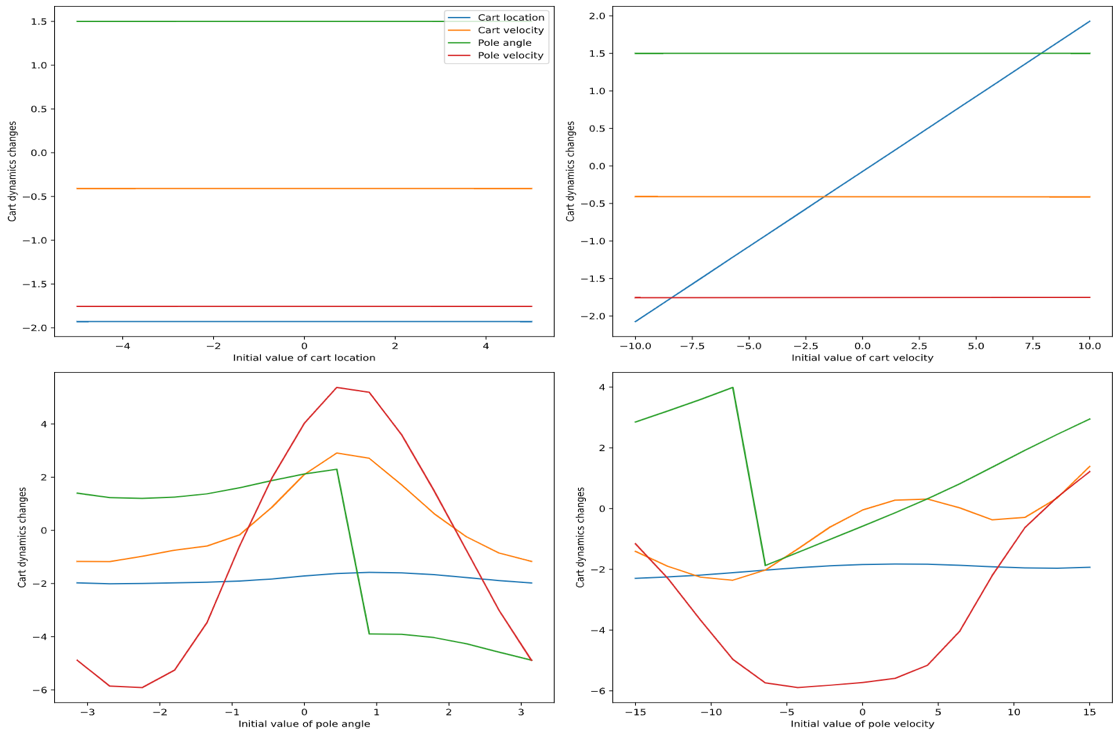


Figure . Change in cart dynamics after one step

Another interesting way to visualise these results is to scan over 2 variables (keeping others fixed) and observe the change in cart dynamics after 1 step, shown in Figure 5.

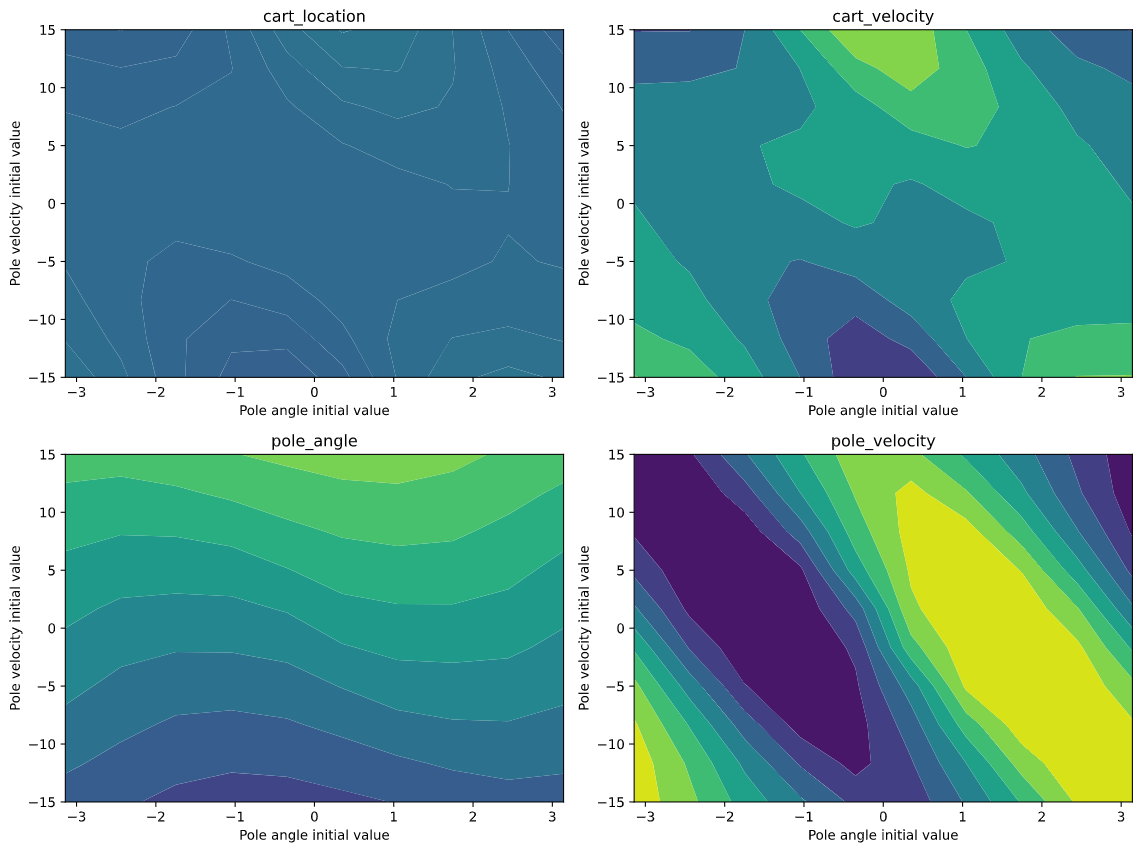


Figure 5. Contour plots displaying the change in cart dynamics with 2 changing initial variables – Pole angle and Pole velocity.

In my code, I have plotted all combinations of the 4 variables and in each one it is clear that cart location doesn’t have an effect. I have included Figure 5 here to display the fact that pole angle and pole velocity have a non-linear effect on the change in cart dynamics after 1 step.

## Task 1.3

To perform linear regression, I generated 500 (X) data points randomly and got 1-step change (Y) points by running the perform\_action function once. I split these (X,Y) pairs into a train and a test dataset. To perform linear regression, I tried 2 ways. Firstly, using the numpy pseudoinverse function, I obtained the optimal weights W, where:

Secondly, out of interest, I used the “sklearn” package to perform linear regression as well. The results were nearly identical and can be found in my code. Throughout this report I will be discussing the results of the first method.

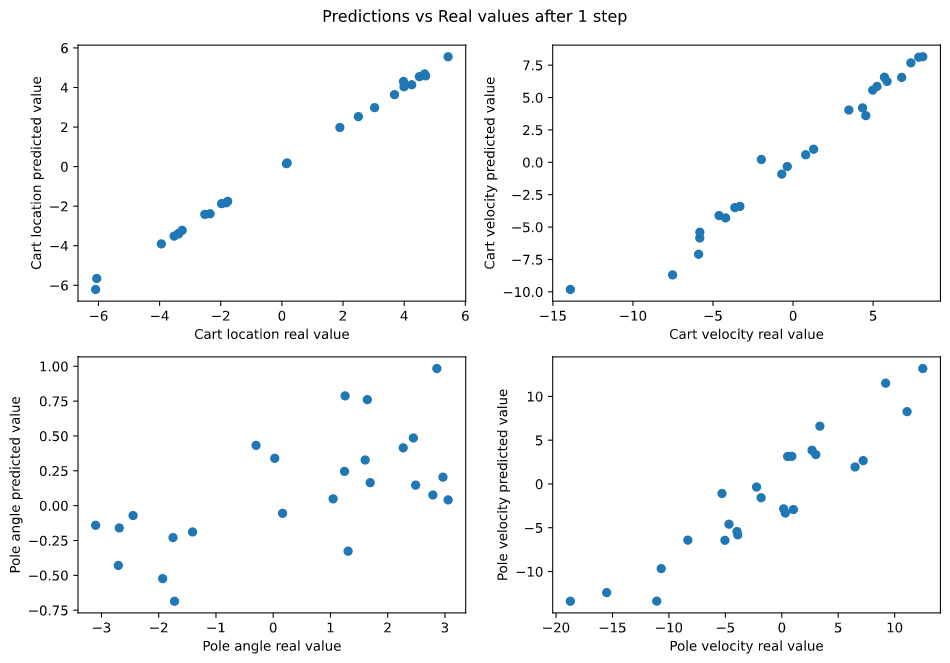


Figure 6. The next real value on the x-axis and the next predicted value on the y-axis for all variables.

In Figure 6 I have plotted the predictions after 1 step versus the real cart dynamics after 1 step. In Figure 6, the perfect predictor would give a straight line of gradient 1. Visually, we see that the linear regression results work relatively well for cart location, and to an extent, cart velocity too. However, scanning through the pole angle and velocity did not produce predictions close to the real values. This is evident visually through the bottom charts of Figure 6, but I wanted to quantify that gap (since the axes have different scales). For this, I wrote the function “rmse\_calc” that calculates the root mean square error (RMSE) of each variable. The results were [0.12,1.06,1.83,2.79]. These figures confirm my visual observations. It is worth noting that due to the remap\_angle function, the range of pole angles is only [-π, π]. Hence, a RMSE of 1.83 is relatively large to that of 2.79 (pole velocity) when we consider the range of the scans. This explains why the pole velocity charts in Figure 6 seem better fit than those of pole angle, even though they have a higher RMSE.

From Figure 7, we can see why cart location has such a small RMSE – as we vary any variable, the real cart velocity remains roughly linear, thus can be predicted by a linear model. The other 3 variables vary non-linearly over the scans, so can’t be predicted accurately by a linear model. In the top 2 graphs, there is an offset with the predictions, even though the real scans are linear. This might occur due to the non-linearity in the scans of the other variables. The non-linear contours for cart velocity, pole angle, and pole velocity in Task 1.2 support the results shown in Figure 7.

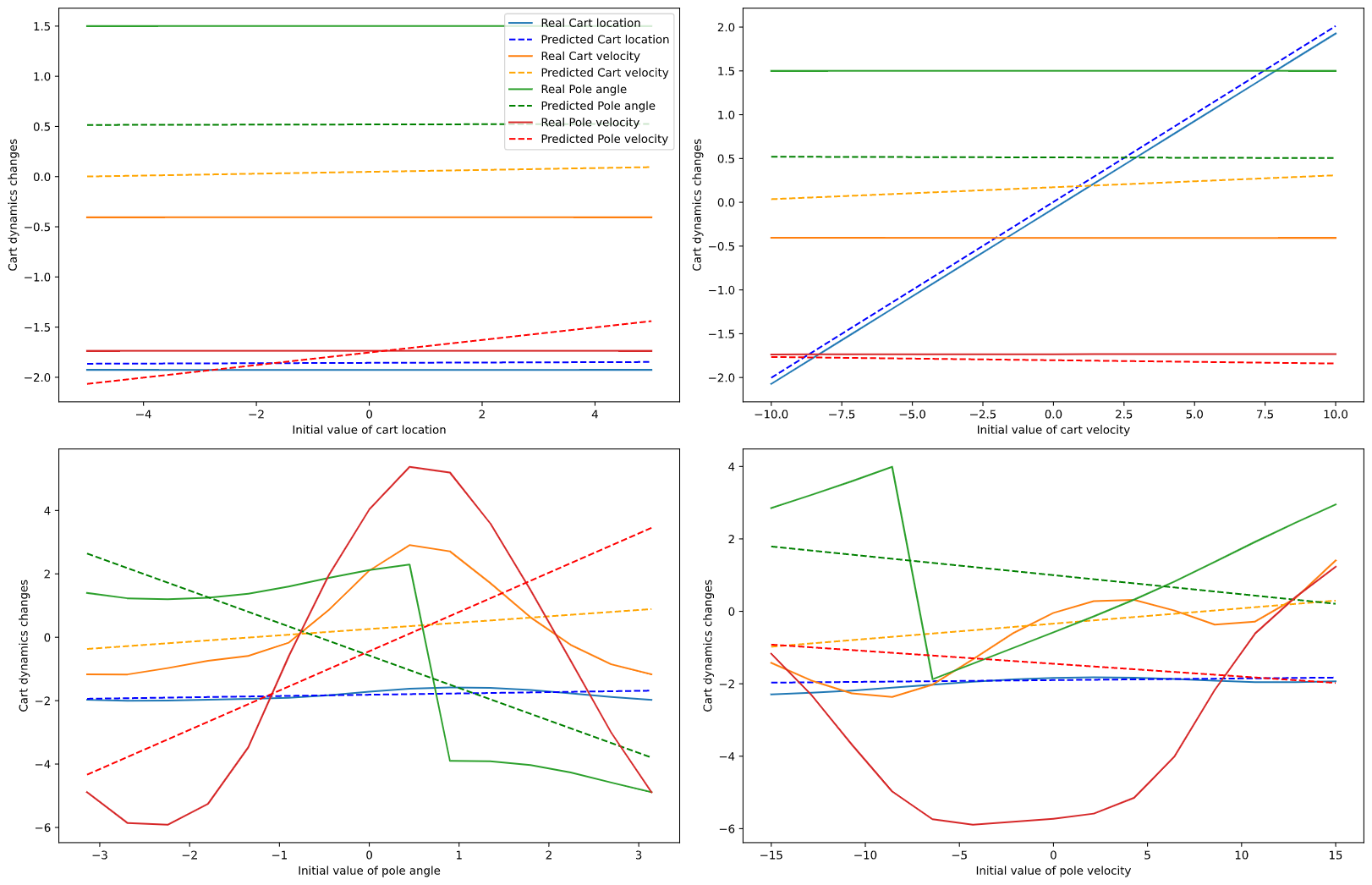


Figure 7. Predictions vs real cart dynamics after 1 step change, scanned over all variables.

## Task 1.4

To model how well the linear model predicts into the future, I decided to explore 2 paths. Firstly, I used real dynamics from the previous step to predict the next step - displayed in Figure 8. Secondly, I used the predicted value from the previous step to predict the next step, displayed in Figure 9.

We see from Figure 8 that the linear model performs well on cart location given the previous real dynamics (this is what we saw in Task 3, that the model performs well for 1 step). The cart velocity and pole velocity are also predicted well, but an obvious delay is noticeable. This might be the model registering the previous value and adjusting its prediction for the next step. The pole angle prediction is making small adjustments but is nowhere near the true value. That is understandable given the RMSE of 1.83 and the sudden changes between -π and π around the equilibrium.

The lines in Figure 9 are much smoother than those in Figure 8. This is expected since the model doesn’t receive any “correction” data, but just propagates the trend it started. In general, the predictions are worse after the first step, which is expected since our model was only trained on data after one step. More complex (non-linear) models could make the results better. Note that this model performs significantly worse in the case where the pendulum makes a full round.

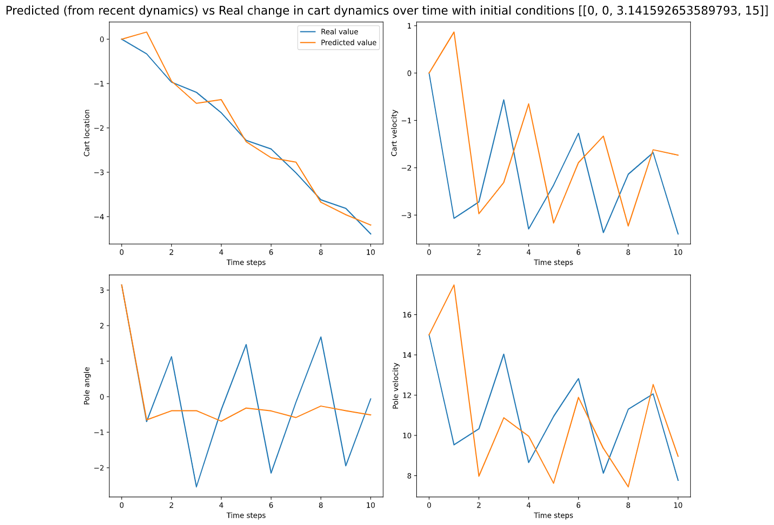
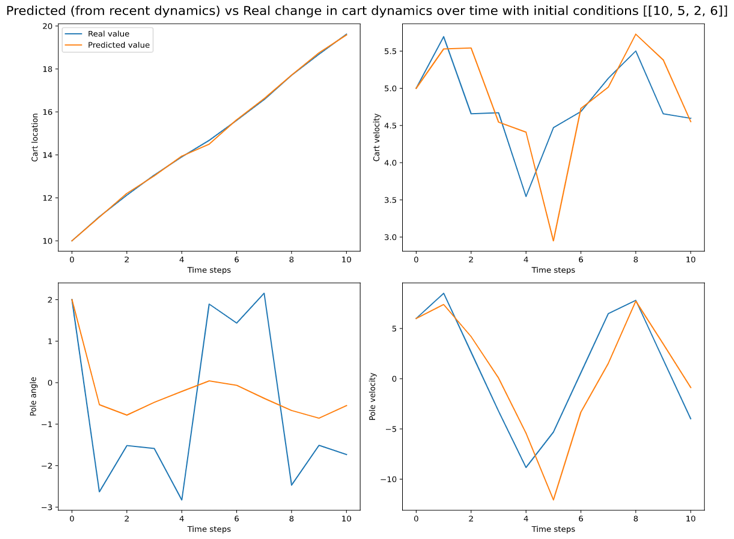


Figure 8a. Time forecasting using real change in dynamics. Initial conditions [10,5,2,6].

Figure 8b. Time forecasting using real change in dynamics. Initial conditions [0,0, π,15]. A full rotation of the pendulum.

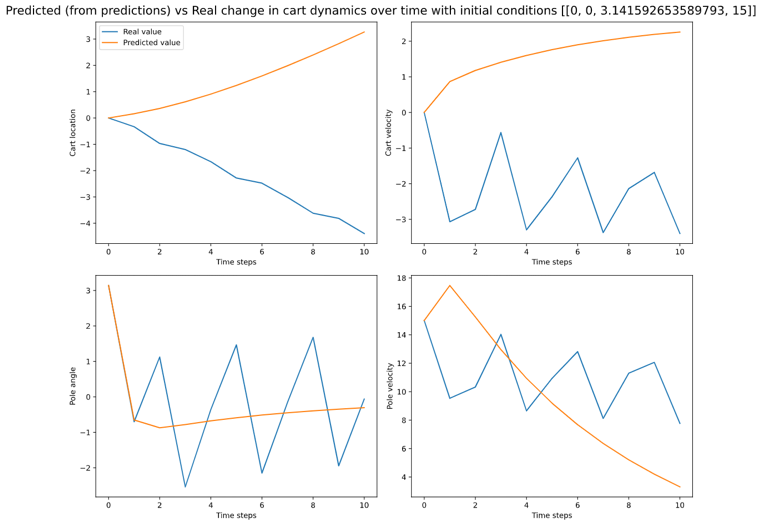
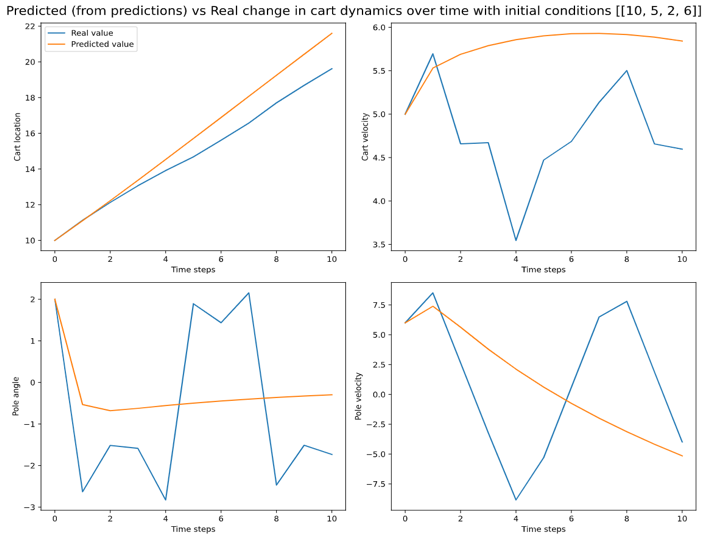


Figure 9a. Time forecasting using previous prediction. Initial conditions [10,5,2,6].

Figure 9b. Time forecasting using previous prediction. Initial conditions [0,0, π,15]. A full rotation of the pendulum.

If the angle was not remapped during the model training and prediction, the predictions diverge from the real dynamics. This is due to the fact that the equations of motion only contain θ in trigonometric functions. Hence, for the real dynamics, it won’t matter if the angle is remapped or not. For instance, a pole angle of 3π will result in the same next step as an angle of π. However, for a linear model, an angle of 3π is vastly different to π. Therefore, without remapping, the real dynamics and predictions will diverge; all the variables will diverge since pole angle affects all variables.

This raised a question in my head. Will the prediction diverge absolutely (ie. to infinity), or will it only diverge from the real dynamics but stay stable? To answer this, I created a model that is trained and deployed without the angle being remapped. The coefficient matrices produced by the remapped and the non-remapped models both have stable eigenvalues (within the unit circle). This can be seen in my code. Hence, we can conclude that although the prediction diverges from the real dynamics without remapping, it stays stable.

# Task 2

## Task 2.1 and 2.2

In Task 2.1, I introduced a non-linear model in order to improve upon our linear predictions from Task 1. This new model made use of a Gaussian Kernel function to define the non-linear basis functions. Performing linear regression on these basis functions should lead to increased accuracy. The target function was, as before, the change in state after 1 step.

In Task 2.2, I expanded on the previous model by introducing a 5th state vector – the action. While an extra variable implies that our model’s accuracy should decrease, this was not largely evident through the results. Hence, instead of showing very similar graphs (from Task 2.1 and 2.2), in this section I will mainly analyse the results of Task 2.2, with some reference to Task 2.1.

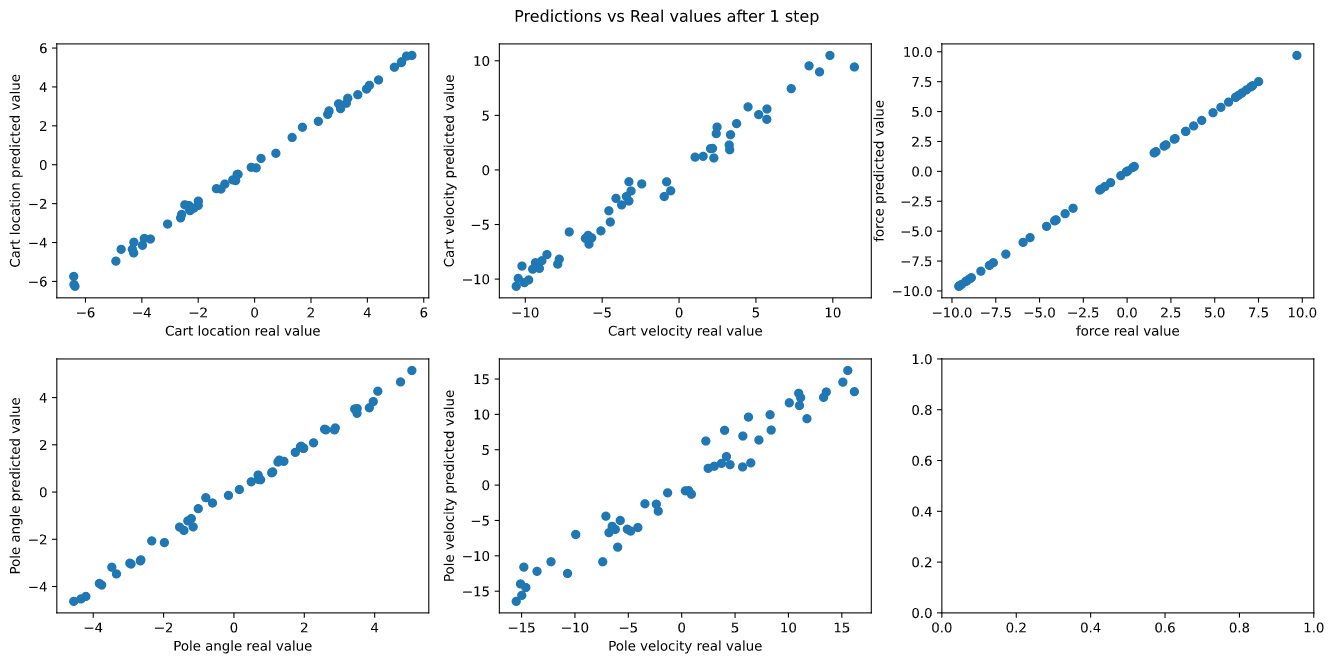


Figure 10. The next real value on the x-axis and the next predicted value on the y-axis for all variables, including force.

From Figure 10, we see lines of gradient approximately equal to 1. This indicates that the predicted value after 1 step (y-axis) is similar to the real next step (x-axis). The predictions of force are 100% accurate, since in this case the force is constant. In order to quantify the accuracy of these predictions, we look at the RMSE.

Figure 11 shows the RMSE evolution with regards to the number of data points (n) and number of basis centres (m) for each variable. Initially, the RMSE reduces dramatically for all variables with increasing m. As m increases, the rate of RMSE reduction slows down. For instance, the change in RMSE between m=10 and m=20 is much greater than between m=160 and m=320. Since increasing m leads to larger calculation times, there is a trade-off between accuracy and time. For the rest of this section, I will be using 320 basis centres. Looking at the y-axis values, we see that the RMSE for cart location and pole angle are relatively low, whereas pole velocity is high. This corresponds to Figure 10, where the pole velocity scatter plot does not follow the ideal gradient line as closely as the other variables.

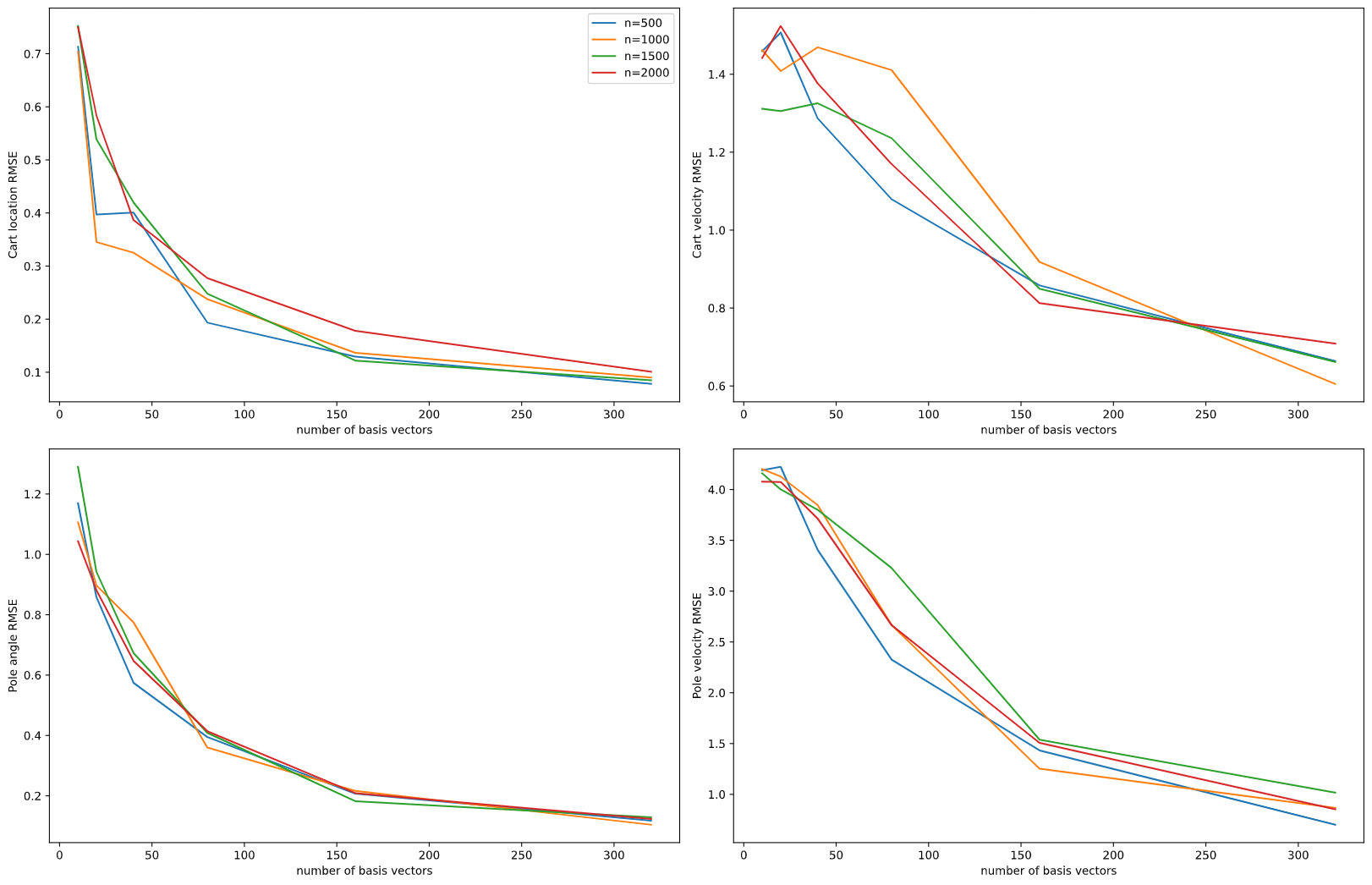


Figure 11. Convergence of RMSE with varying number of data points (n) and basis centres (m).

Interestingly, increasing the number of data points (n) does not decrease RMSE considerably. In fact, the RMSE generated by n=2000 is sometimes higher than the RMSE generated by lower values of n. This could be because proportionally, m represents a smaller subset of n when n is large. For instance, with m=320, that would mean 71% of training data is a basis centre for n=500 (assuming 95% training split). By comparison, only 17% of training data acts as a basis centre when n=2000. Using this logic, the model should perform similarly for the combinations (m=320, n=2000) and (m=80, n=500). However, Figure 11 clearly displays this is not the case. Hence, we conclude that increasing m reduces RMSE with diminishing returns and increasing n doesn’t affect it much. The small variation due to changing n might be attributed to random selection of basis vectors rather than n/m proportions.

Taking a step back, one important question that comes to mind is the hyperparameter selection. What values of sigma and lambda are best for our case? For sigma, I began with calculating the standard deviation of the variables in my generated data. This gave sigma = [2.89, 5.78, 1.85, 8.11, 5.78]. To optimize this, I imagined a 5-dimension grid, where we calculate the RMSE at each point, eventually settling on the sigma with the lowest RMSE. However, this would be computationally complex, especially considering that the optimum sigma could vary with state, so instead I decided to vary one component of sigma at a time while keeping the rest fixed. This is shown in Figures 12a and 12b.

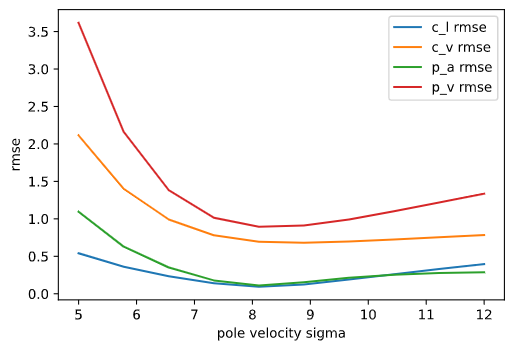
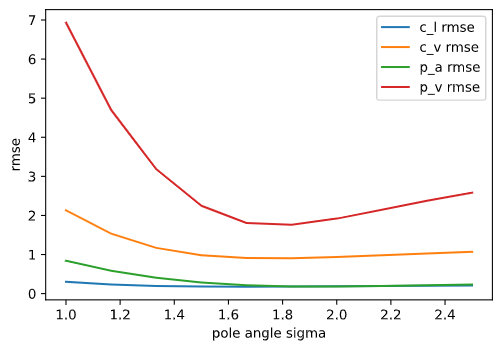


Figure 12a. Varying the sigma value corresponding to pole angle and observing effect on RMSE.

Figure 12b. Varying the sigma value corresponding to pole velocity and observing effect on RMSE.

From Figures 12a and 12b, we see that the minimum RMSE occurs very close to the original values obtained by standard deviation. This is assuming that the range provided is correct, so that there is no global minimum that lies out of range. Of course, this isn’t necessarily the minimum since we are only changing one variable at a time, but this cost-efficient method confirms that I can proceed with the original sigma. Scanning over the other 3 variables confirms this as well.

I plotted lambda on the x-axis using a logarithmic scale, and RMSE on the y-axis for all variables. From Figure 13a we see that the RMSE is low for all values of lambda lower than 10-2. This is illustrated in Figure 13b, which confirms that the predictions are inaccurate when we set lambda=0.1. Interpreting lambda as the data noise, this makes sense. If lambda is large, our model should perform worse. For the rest of this section, I will set lambda = 10-5.

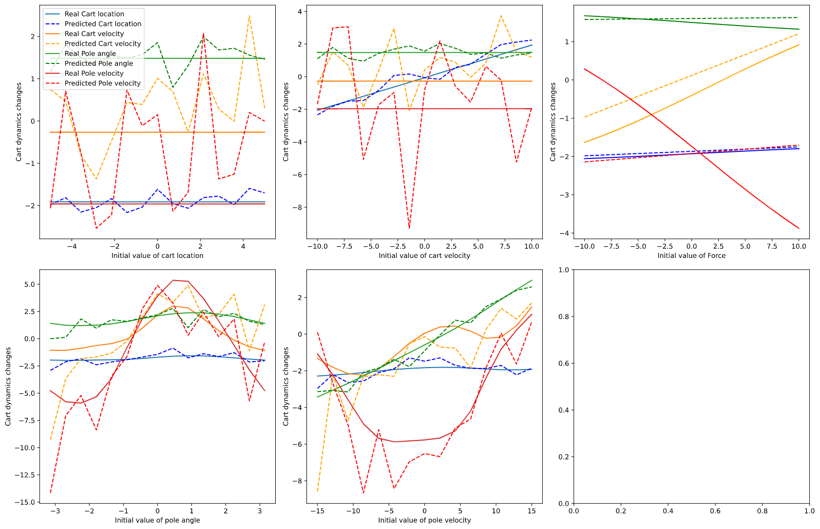
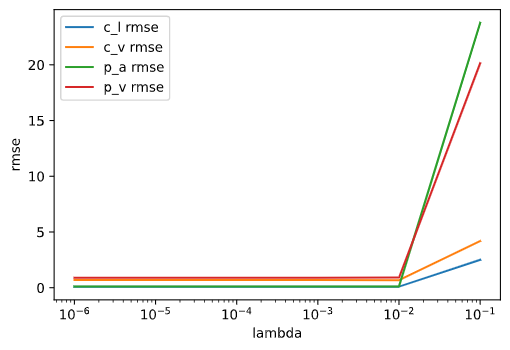


Figure 13a. RMSE with different values of lambda.

Figure 13b. Scans over variables with model trained with lambda=0.1.

Once the hyperparameters had been set (assuming that their optimum values don’t vary greatly with state), I plotted 1D scans over variables with initial conditions [-0.24,-9.27,-1.07,9.09,1], shown in Figure 14. We see that the pole angle and cart location are predicted accurately, and there is some variation in the cart velocity and the pole velocity. This is consistent with the higher RMSE of these 2 variables in Figure 11. Furthermore, the non-linear predictions in Figure 14 visually demonstrate that our non-linear model works (and is better than the linear model).

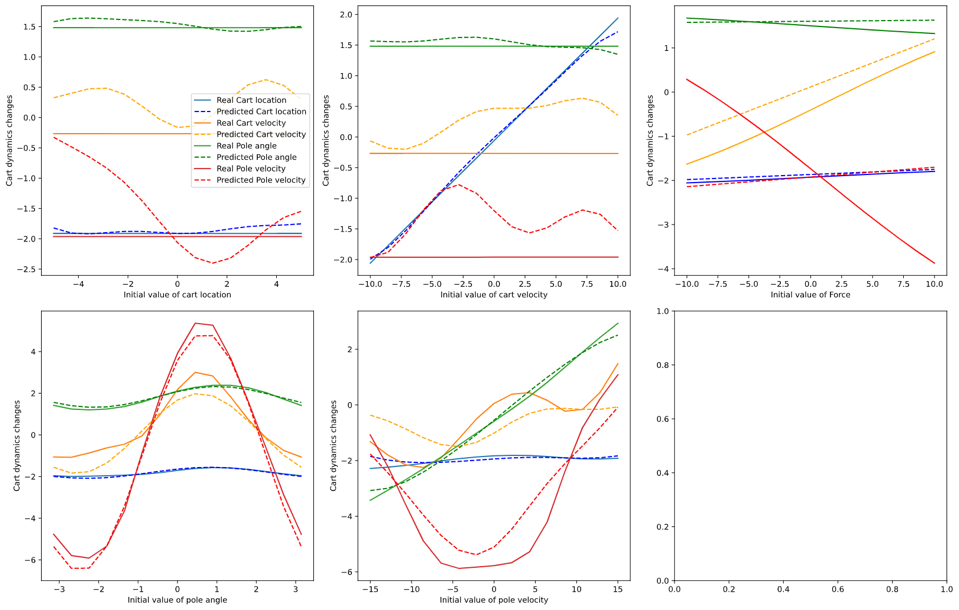


Figure 14. Scans over variables with real dynamics and predictions. Lambda = 10^-5.

Figures 15a and 16a show the predicted change in variables when we scan over 2 variables at a time. Figures 15b and 16b show the real change for those same variables. We see that the predicted contours are non-linear, and manage to capture the rough shape of the real contours. However, the scale bar shows some disparity between the values. The larger disparities occur in cart velocity and pole velocity; this is consistent with Figure 11 and Figure 14 because they show these variables having a larger RMSE. The difference in scale but similarity in shape can also suggest an offset between predictions and the real dynamics, which is also observed in Figure 14. In the 1D scans, we see the predictions of pole velocity and cart velocity roughly follow the real shape, but oftentimes with an offset.

Next, I investigated rollouts to see how closely the predictions matched the real dynamics with time. Figure 17 shows these rollouts with different initial conditions when the model took real data as its last input. Figure 18 shows these rollouts when the model took its previous prediction as the input.

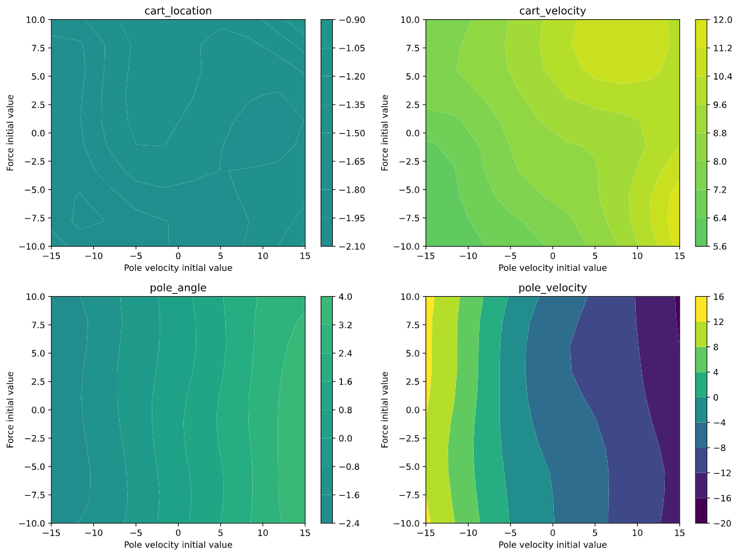
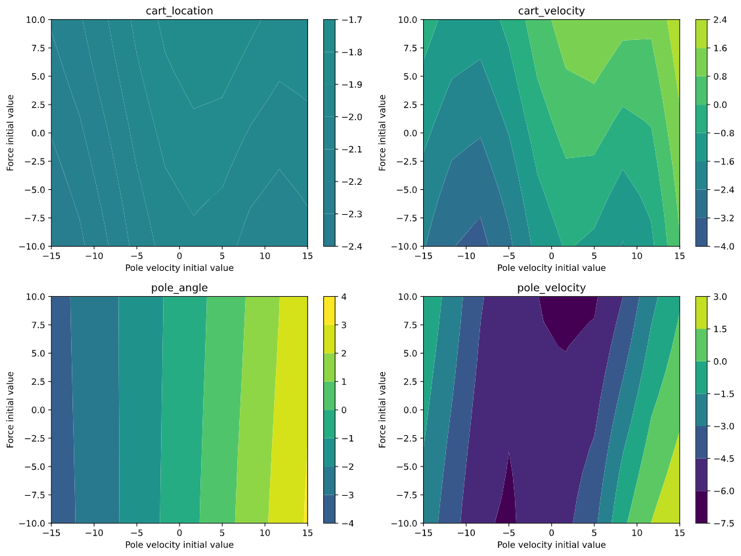


Figure 15a. 2D scan predictions while varying Pole velocity and Force.

Figure 15b. 2D scan real dynamics while varying Pole velocity and Force.

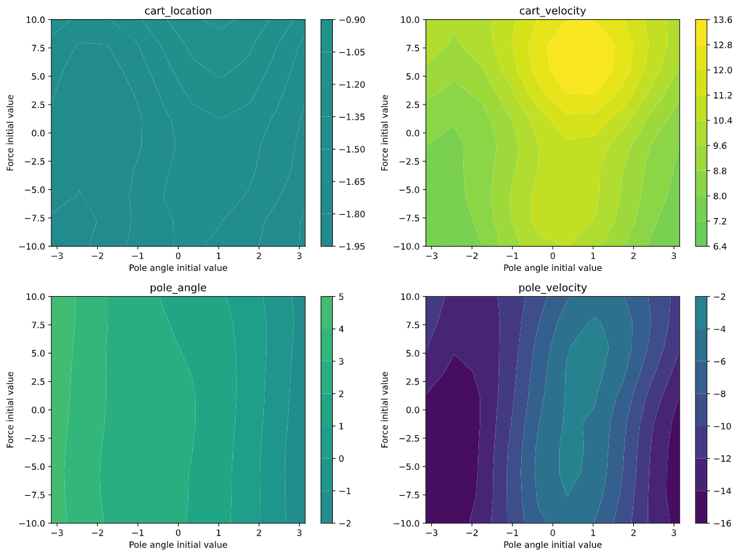
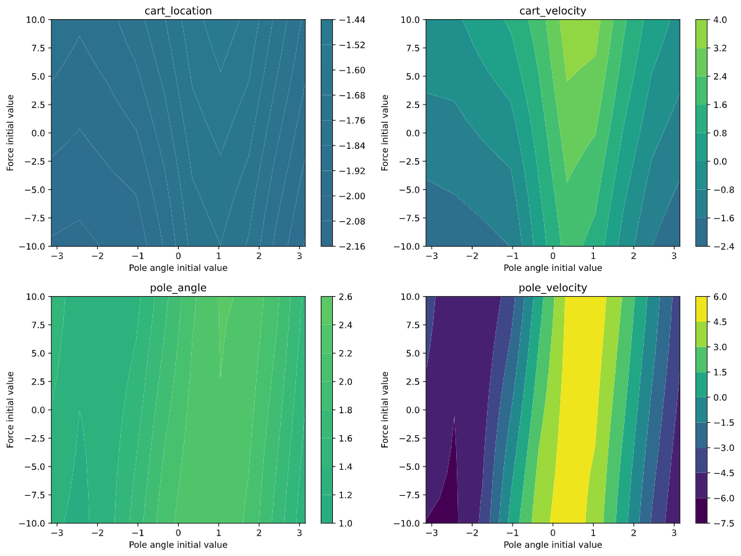


Figure 16a. 2D scan predictions while varying Pole angle and Force.

Figure 16b. 2D scan real dynamics while varying Pole angle and Force.

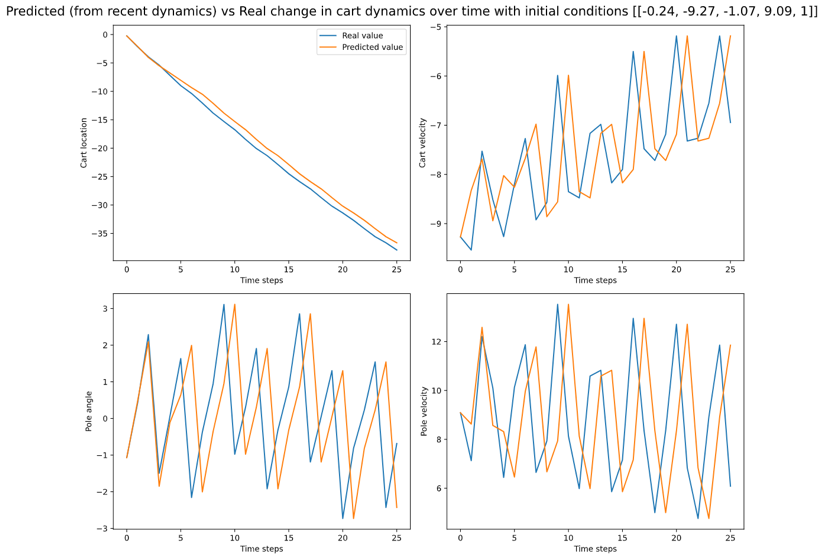
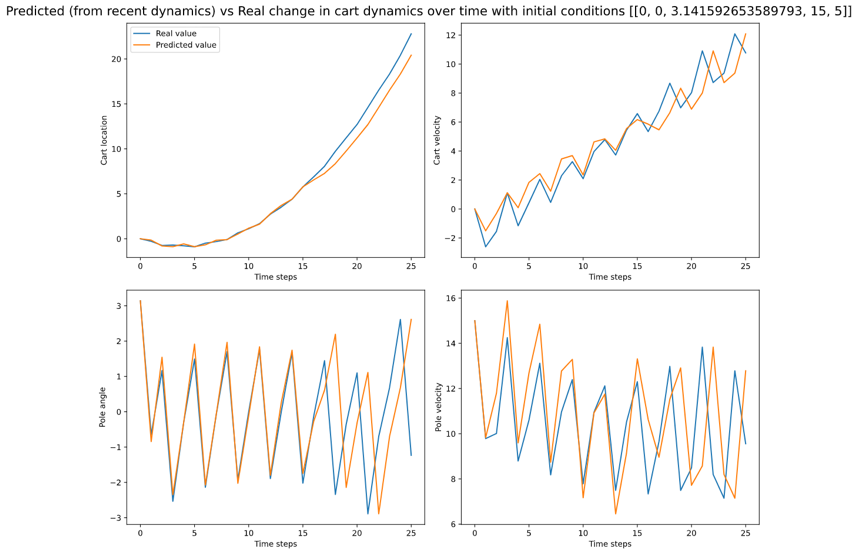


Figure 17a. Predictions vs Real dynamics using previous real dynamics. Initial condition [-0.24,-9.27,-1.07,9.09,1]

Figure 17b. Predictions vs Real dynamics using previous real dynamics. Initial condition [0,0,π,15,5]

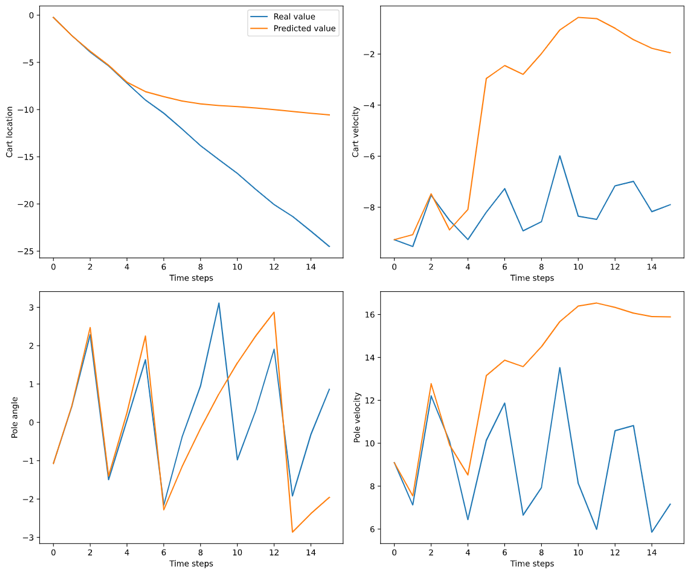
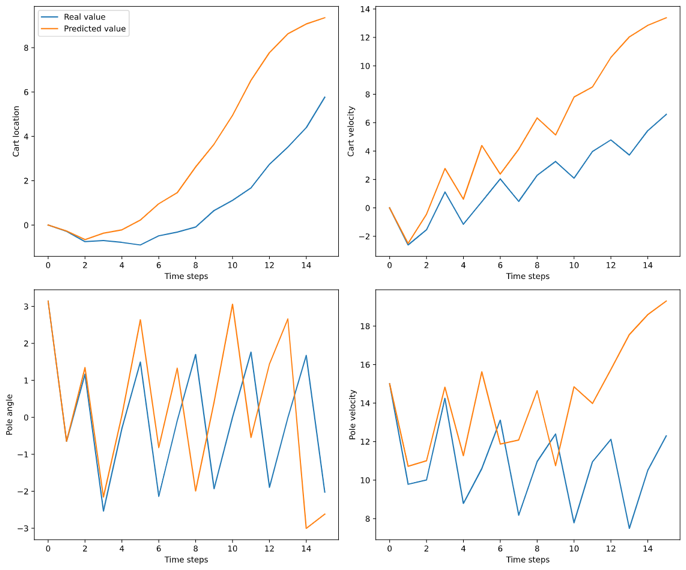


Figure 18a. Predictions vs Real dynamics using previous prediction. Initial condition [-0.24,-9.27,-1.07,9.09,1]

Figure 18b. Predictions vs Real dynamics using previous prediction. Initial condition [0,0,π,15,5]

Figure 17 shows that the model predicts the real dynamics extremely well for one initial condition (17b), and relatively less well for another starting condition (17a). With both starting conditions, the model predicts cart location and pole angle with greater accuracy than cart velocity and pole velocity; this is consistent with the RMSE values measured in the test data. The model predicts cart location and pole angle exactly for 4 time steps (roughly 1 oscillation), but stays close for the rest of the graph of 25 steps. In 17b, the model predicts nearly exactly for 15 time steps (roughly 5 oscillations). Perhaps this might be due to its pole angle starting from a stable position.

Figure 18 shows that if the model uses its own predictions to forecast the dynamics, the forecast will diverge from the real dynamics after 4 to 8 steps (2 to 3 oscillations).

## Task 2.3

Here we are introduced to linear policies – a function that determines the next action based on the current state in order to get the desired position. We define the desired state as [0,0,0,0]. We also introduce a loss function that is higher when the state is further away from our desired state. Currently, we will keep the same weightage for each variable when calculating loss, but this assumption will be relaxed later.

Figure 19 shows 1D and 2D scans of components of the p vector and their effect on loss. The p vector is used to determine the action for the next step (which is passed through a tanh function to get the final force). We see that there are certain clear optimums in our choice of p that reduce loss. Naturally, this will depend on our state vector as well.

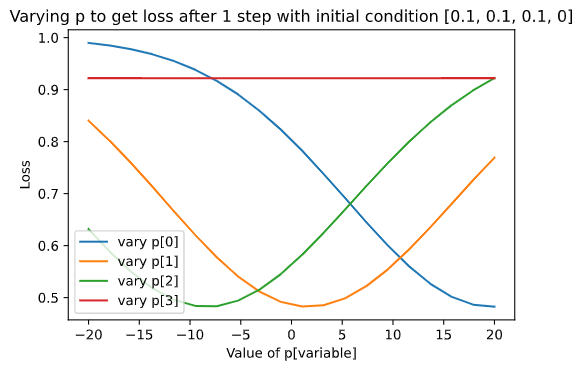
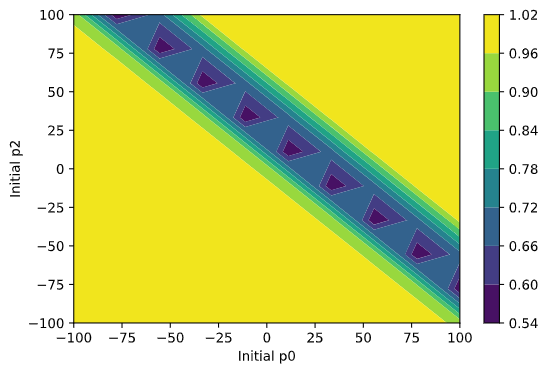


Figure 19a. Varying single components of p=[0.5,2,10,10] and measuring loss

Figure 19b. Varying 2 components of p=[0.5,2,10,10] and measuring loss

In order to get the best value of p for states close to our desired state, I generated 50 training points with all components between 0 and 0.2. Then I used scipy to optimize the p vector so that it generates the lowest cumulative loss over the training data. Since the data was generated randomly, I got several p vectors, of which the best was p=[-1.5124,-0.4646,103.45,13.5855].

Figure 20 shows the rollout of the cart dynamics using actions dictated by our policy. Figure 20a shows an initial condition that leads to the pole being upright even after 15 time steps. It is important to note that the cart velocity is non-zero, hence it is not completely at our desired state. Figure 20b shows a rollout that diverges from our ideal state within 2 time steps – this might be because the initial conditions are above the 0.2 limit our p vector was optimized upon.

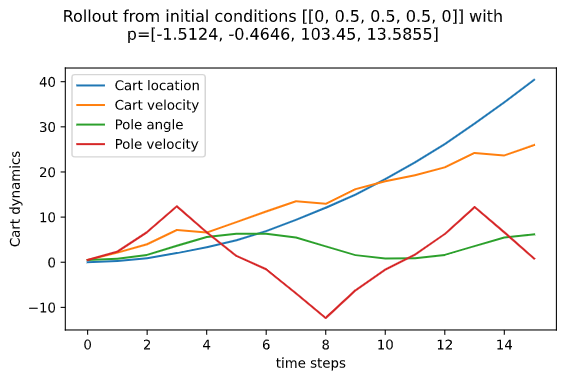
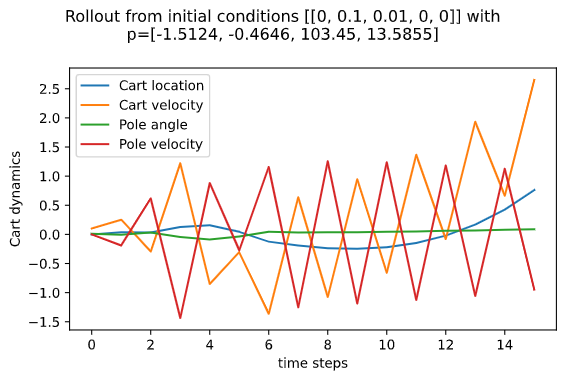


Figure 20a. Rollout from initial conditions in title.

Figure 20b. Rollout from initial conditions in title

## Task 2.4

In this task, I plotted rollouts using data from the non-linear model developed in Task 2.2. Due to the results in Figure 18, I set the number of time steps to 8, which is when the model might diverge from the real dynamics.

Figure 20a and 20b show that the pole stays upright for 8 time steps using the predictions made by the model. What does this say?

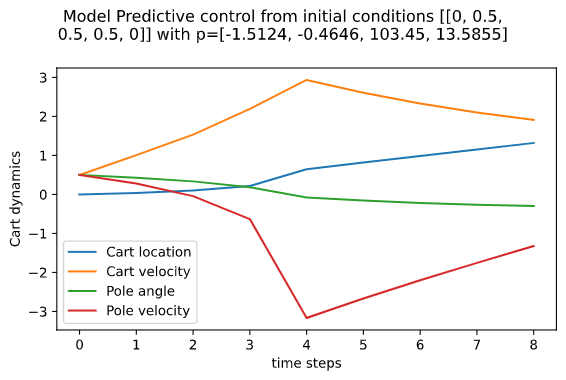
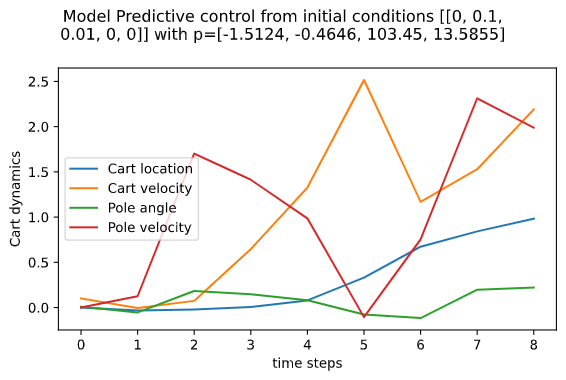


Figure 20a. Rollout using model predictive control from initial conditions in title.

Figure 20b. Rollout using model predictive control from initial conditions in title.

## Task 2 extra investigation

Coming back to the loss function, we want our cart to ideally be still and our pole to be upright. This means that we must place more emphasis on pole angle and cart velocity to be 0 than on cart location to be 0. I experimented by editing the loss function. The old loss sigma used above was [0.5,0.5,0.5,0.5]. I changed that to be [200,0.005,0.005,0.5]. This meant that the loss would be minimal if cart location was far from 0, but the loss would be great if the pole angle or cart velocity were far from 0.

Figure 21a shows the rollout from an initial state where cart location is high. We see that the policy doesn’t make the cart location go to 0 quickly, but rather focuses on keeping the cart velocity and pole angle close to 0. To test this new value of p, I initialised the system with a large cart velocity, shown in Figure 21b, but the rollout didn’t make that tend to 0. However, Figure 21c shows the same initial system but with the p value from Task 2.3. Here we see that cart velocity increases, rather than remaining constant, so perhaps our new p value is working.

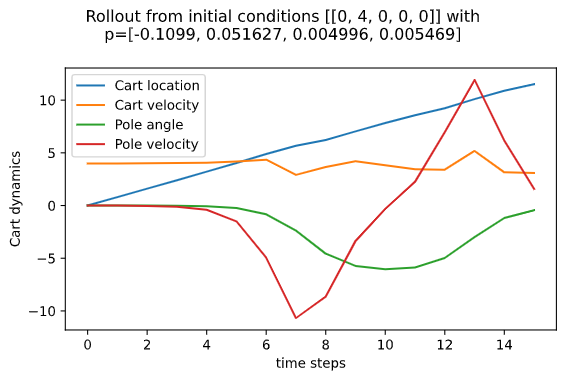
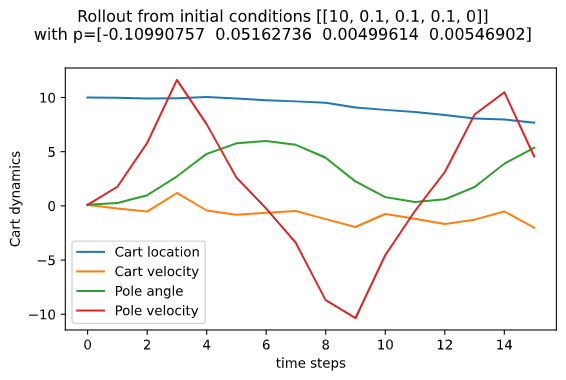


Figure 21a. Rollout after experimenting with loss sigma



Figure 21b. Rollout after experimenting with loss sigma

Figure 21c. Rollout after experimenting with loss sigma

Figure 21d. Rollout after experimenting with loss sigma

Figure 21d is the result of my experimentation with the loss sigma values. I wanted to see the rollouts if the loss sigma were to be edited, but not as heavily as in Figures 21a and 21b. So, I set the loss sigma to [50,0.05,0.05,0.5]. This produced Figure 21d, which shows that the pole angle is kept at 0, but cart velocity is allowed to rise; this is surprising since both variables have the same loss sigma.

# Task 3