IIA project SF3: Machine Learning

Abhishek Shenoy (ams289)

Easter 2021
Project Leader: Gabor Csanyi (gc121)

1 - Dynamical simulation ¶

Introduction

Consider the inverted pendulum system ("cartpole") drawn above, familiar from the coursework of 3F2, with a freely moving cart and freely rotating pendulum attached to the cart, moving under the action of an external action force and gravity.

The equations of motion of the system are

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

where the stationary points are $\theta = 0$ (unstable) and $\theta = \pi$ (stable), and F is the external *action* (force) on the cart, μ_x and μ_θ are the friction coefficients of the cart and the pole, respectively.

The state of the system is described by the following variables: position and velocity of the cart $x, \dot{x}$, angle and angular velocity of the pole $\theta, \dot{\theta}$, with the angle being periodic on $[-\pi, \pi]$. The center position of the cart corresponds to $x = 0$, and the pole hanging vertically down corresponds to $\theta = \pi$. The equations of motion can be derived using Lagrange's equation.

Our aim is to understand and qualitatively and quantitatively assess the use of a linear model for simulating the behaviour of the non-linear system.

1.1 - Free-Oscillating System

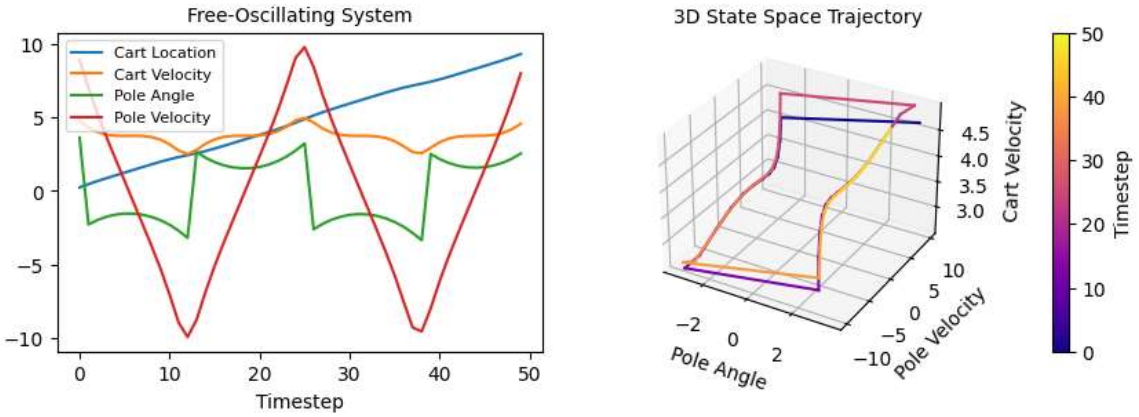
`CartPole.py` file contains a Python class to describe the system. Note the variables that describe to the state of system, and the `performAction()` function that updates the state variables using the *Euler* algorithm (it does a small number of steps), using a given force (which is the 'action') on the cart. Passing a zero value for the force corresponds to free dynamics.

By simulating a “rollout” (i.e. a run with specified initial condition simulated for a number of time steps) using the `performAction` function in a loop, starting from the stable equilibrium position and some nonzero initial cart velocity or angular velocity, and no applied force, we plot the resulting time evolution of the system variables as shown in **Fig 1**. We vary the size of the initial velocities to realize different behaviours: simple oscillation around the stable equilibrium (**Fig 1a**), and also the complete rotation of the pendulum (**Fig 1b**). State variables are best left in the ranges: cart velocity: $[-10, 10]$, pole angle: $[-\pi, \pi]$, pole (angular) velocity: $[-15, 15]$.

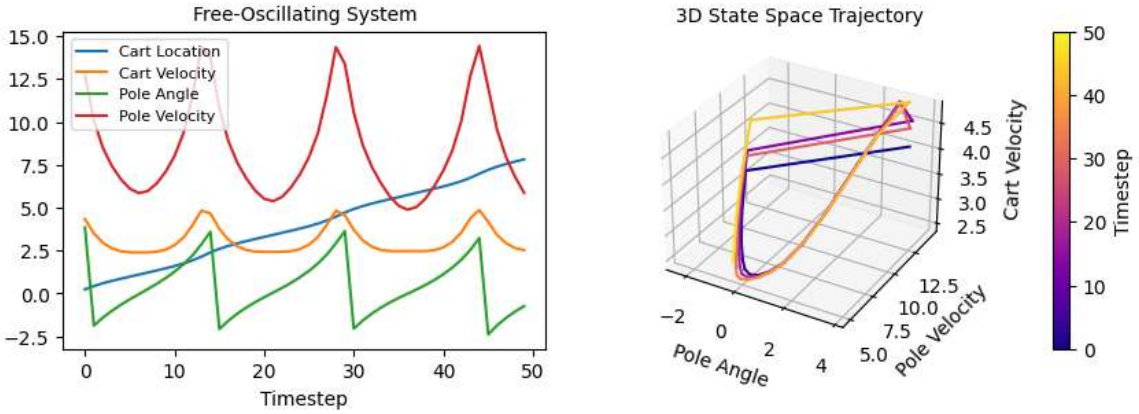
Note how the angle is used in the dynamics as a continous variable, rather than just in the range $[-\pi, \pi]$. There is a `remap_angle` function in the `CartPole` module that is used to convert the angle to the usual range. *This will be an important consideration later on when we develop models of the dynamics.*

The integration time `delta_time` was reduced to 0.05 such as to make the time evolution plots smoother.

`Fig 1a - Simple Oscillation`



`Fig 1b - Complete Rotation`



Plotting the time evolution (**Fig 1a, b**), it is possible to see that the cart location is roughly linearly increasing with time. Hence excluding it from the phase portrait, we are able to see the dynamics of the system with relation to the three main state variables *Cart Velocity*, *Pole Angle* and *Pole Velocity*. The phase portraits show that although the model is non-linear, the free system is not chaotic.

The 3-dimensional state-space trajectories are very useful since by viewing any 2 axes, we can get the 2-dimensional state space trajectory. The time colour scale is also useful in being able to relate the timeseries plot directly to the trajectory plot. Hence we have used this plot to convey all the related information of the dynamics of the system.

For free oscillation (**Fig 1a**), it is possible to see the effect of remapping the pole angle as it oscillates around $\theta = \pi$. The colour bar has been shown so that the trajectory can also be visualised in time.

For complete rotation (**Fig 1b**), it is possible to see that **Fig 1b (2)** differs from **Fig 1a (2)** by the smooth continous trajectory going from $\theta = -\pi$ in the top left hand corner to the minimum point at $\theta = 0$ and then to the top right corner at $\theta = \pi$, verifying that this is indeed a complete rotation. Hence in this trajectory, angle remapping is only performed once every cycle whereas it is performed twice during a free oscillation cycle. This can also be noted by the fact that the pole velocity can be either positive or negative in a free oscillation (swinging back and forth) but during a complete rotation the pole velocity must be either positive or negative exclusively (making consecutive complete rotations) and this is true as shown in **Fig 1a (1)** and **Fig 1a (2)**.

1.2 - State Relationships

Changes of State

From 3F2 we know that a simple linear controller works for this system, as long as you know where the stationary point is, and have access to the equations of motion so that you can linearise them. But in general, we do not know the equations behind the evolution of a physical system, and so we will take a different approach. What we do have are *observations* of the time evolution of the system. So we will use the simulations to gather data about the system, and develop a *model* for this time evolution.

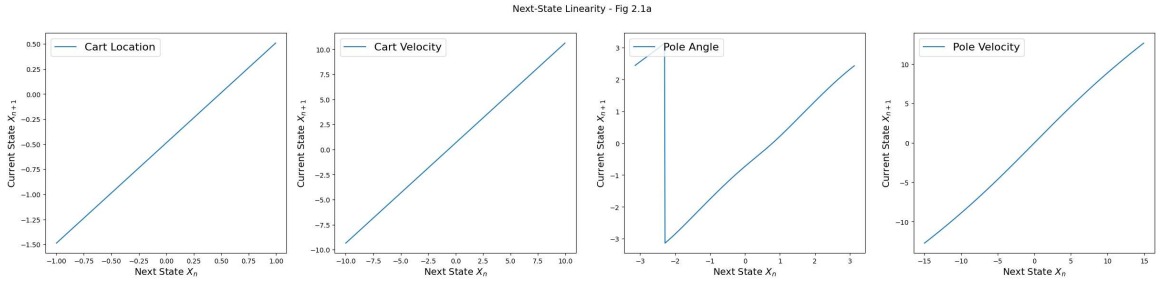
We want to build a *model* for the time evolution of the system. The model is a function $f(X)$ that takes the current state of the system, and maps it onto a new state, which is its prediction for the state at a later time. Let the state of the system be described by a vector X, given by

$$X = [x, \dot{x}, \theta, \dot{\theta}]$$

1.2.1 - Current State to Next State Analysis

Given the current state X_n, let us call X_{n+1} the state of the system after a single call to the `PerformAction` function (with 0.0 as the force argument, or no argument, which is equivalent).

To investigate and visualise the functional relationship between X_n and X_{n+1}, we initialise the system using a random value for all state variables, and then scan through one of the state variables in a suitable range resetting all the state variables after each call to `PerformAction`, and plot X_{n+1} as a function of our scan.



The relationship between X_n and X_{n+1} is nearly linear as expected since the change in one step is small as shown for all the state variables in **Fig 2.1a**. The pole angle however has a non-linearity at the point of remapping although if the angle was not remapped this would be continuously linear and increasing. Within a suitable range, the pole angle relationship is linear.

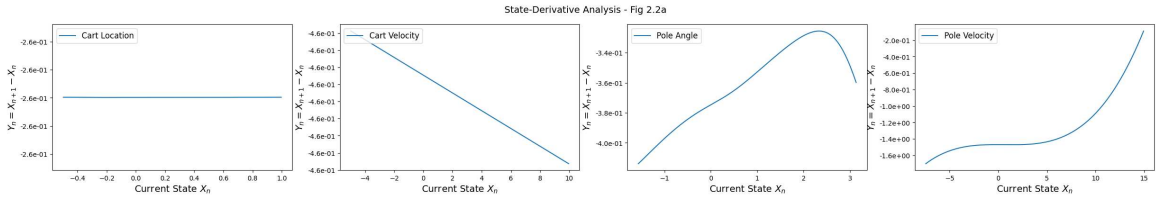
1.2.2 - Current State to Derivative Analysis

We can take account of this and model the *change* in state vector, rather than taking the new state vector itself as the target of our model. So we define the new target for the modelling as $Y \equiv X(T) - X(0)$, where $X(t)$ represents the time evolution of the state under the dynamics, and T corresponds to a single call to `PerformAction` . Note that in principle we could model changes corresponding arbitrary time shifts, rather than a single call to `PerformAction` , but the longer the time shift, the more complex the model would have to be.

Here we explore this new functional relationship again (i) using scans of single variables, and (ii) contour plots where slices of the data in two of the variables are taken while the other two variables are kept fixed.

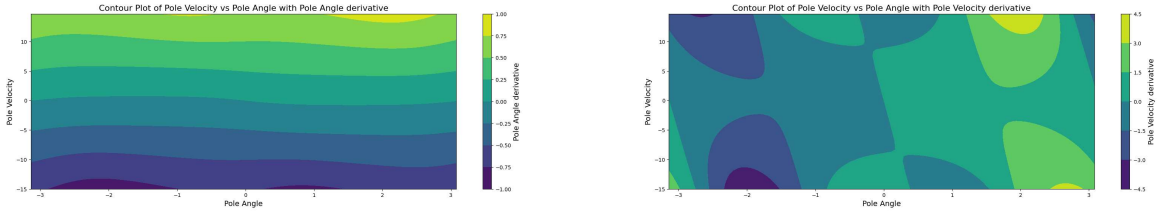
Now we can consider the approximate derivative (the difference with respect to the chosen timestep) $Y_n = X_{n+1} - X_n$ of a state variable to be mapped against the current state variable X_n. By performing separate scans of a single variable for each state variable, we get the following state-derivative relationships shown in **Fig 2.2a**.

Although not shown here, by plotting all 16 graphs instead of plotting only the 4 graphs (each of which shows the derivative with respect to a scan of itself shown in **Fig 2.2a**), it is possible to see that the *cart location* does not affect the *cart velocity*, *pole angle* or the *pole velocity* since the next state of the three variables is unchanged at any cartpole location (ie. the unscanned variable graphs have a single point when cart location is scanned). This can also be intuitively understood as the cart location cannot affect the dynamics since we could simply reconsider this after remapping the position to a small range or specifically 0 keeping the velocities and the pole angle the same. Therefore the cart location has no impact on the next step.



The cart location has a roughly constant (and almost zero) derivative as we saw in the time evolution of the free system. The derivative of the cart velocity is non-linear however can be approximated by a linear derivative for small scan ranges. The pole angle and pole velocity derivatives are non-linear and hence have the greatest impact on the system. Hence by plotting the *pole velocity* vs *pole angle* contours (of either variable's derivative) we can visualise this non-linearity.

Contour Analysis - Fig 2.2b



In **Fig 2.2b**, the greatest non-linearity can be seen in the derivative of the pole velocity which means that it is an important variable for a model to accurately replicate the system.

Since the derivative of the *cart location* is almost constant, this is of least relevance in a contour plot and hence we can ignore this variable. If we consider the derivative of the *cart velocity* with respect to the *cart location* and one of either the *pole angle* or the *pole velocity*, we also find that the *cart velocity* contour is constant across the *cart velocity* axis (y-axis) meaning that it does not affect the other variables (as mentioned before) and hence plotting cart velocity is not inherently useful.

Therefore by plotting the *pole angle* against *pole velocity*, we can see that the most significant plot is the one mapping the derivative of the *pole velocity*. **Fig 2.2b (1)** shows that the *pole angle derivative* is almost constant across the x-axis (ie. with the variation of pole angle) (excluding the remapping regions) and hence this plot is not as significant as **Fig 2.2b (2)**. However this plot is important to note in itself as for non-linear control, the angle remapping must play an important role for stabilising the system.

1.3 - Linear Model

The simplest model is a linear one, where the target Y is assumed to be linear function of the current state X,

$$f(X) = \mathbf{M}X$$

where $\mathbf{M}$ is a 4×4 matrix of coefficients.

By initialising the simulator in a completely random state (using suitable ranges) and running it for *one* step, we gather data in the form of pairs of state vectors (X_n, Y_n), where X_n represents a state of the system at step n and Y_n represent the change in state after a single call to `performAction` (with zero force), such that $Y_n \equiv X_{n+1} - X_n$.

1.3.1 - Linear Model Fitting

Using the generated data set, we perform linear regression to find the optimal coefficient matrix. We can then test our predictions against the collected data by plotting the results with the current state variable $(X_n)_i$ on the horizontal axis and the predicted state variable $(\hat{X}_n)_i$ on the vertical axis for each variable i. We also plot the respective derivatives $(Y_n)_i$ and $(\hat{Y}_n)_i$ on the x-axis and y-axis respectively. In this plot, a perfect prediction would correspond to a perfect straight line as annotated by the *benchmark*.

Least Squares Method for Linear Regression

$\underline{y_i} = \underline{m_i} X + \underline{c_i} \qquad \therefore Y = MX + C$

Rewrite the above as $AP = B$ where $B = Y$, $A = [\, X \;\; I \,]$ (In Python [[X I]]) and $P = \begin{bmatrix} M \\ C \end{bmatrix}$ (In Python [[M], [C]]) $P^* = \arg_P \min |B - AP|^2$

Attempting this did not significantly improve the outcome and possibly worsened the behaviour since the model was more overfitting to the generated dataset by compensating with the intercept matrix C.

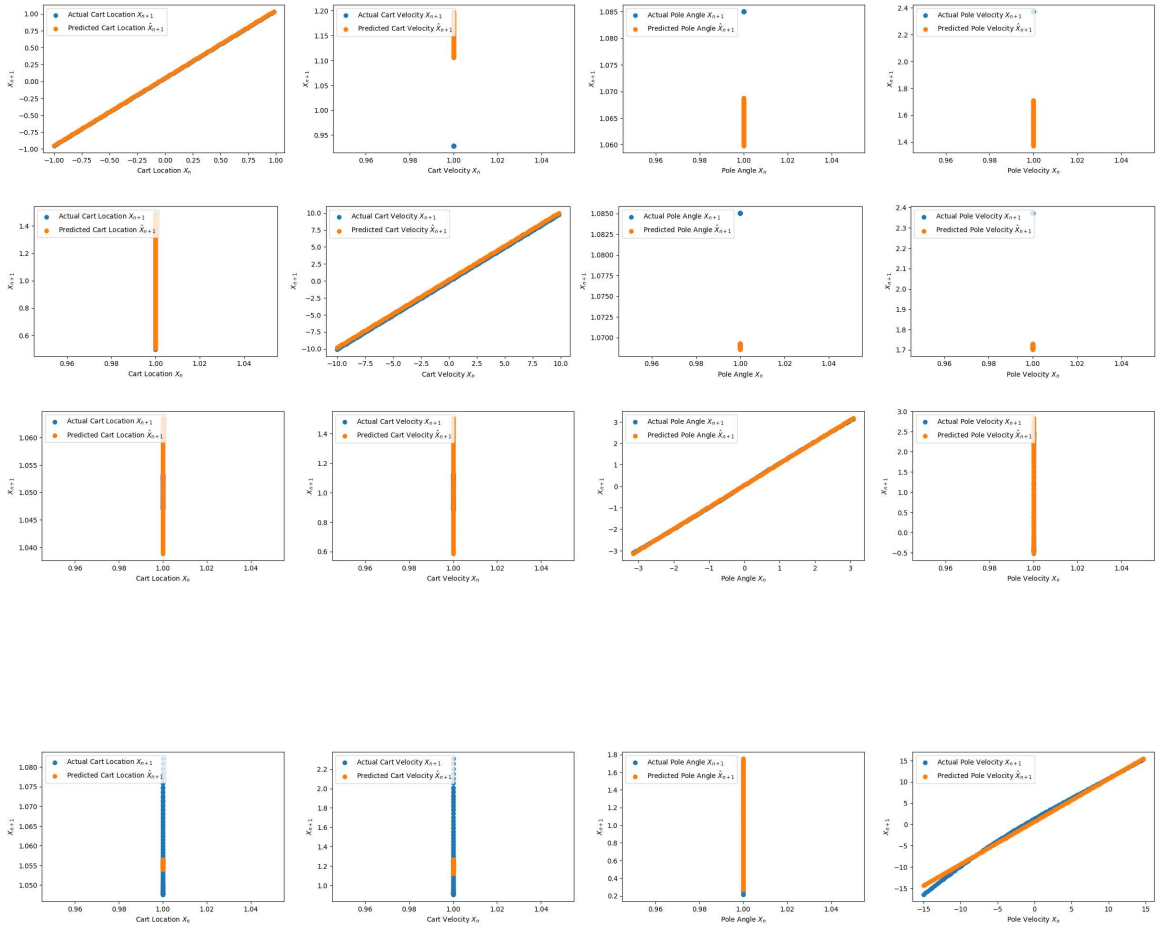
Hence instead assuming $Y = MX$ as suggested by the task:

$M^* = \arg_M \min |Y - MX|^2$ (This is the method numpy np.linalg.lstsq uses)

Using the pseudoinverse (np.linalg.pinv) is not viable for a large number of data points.

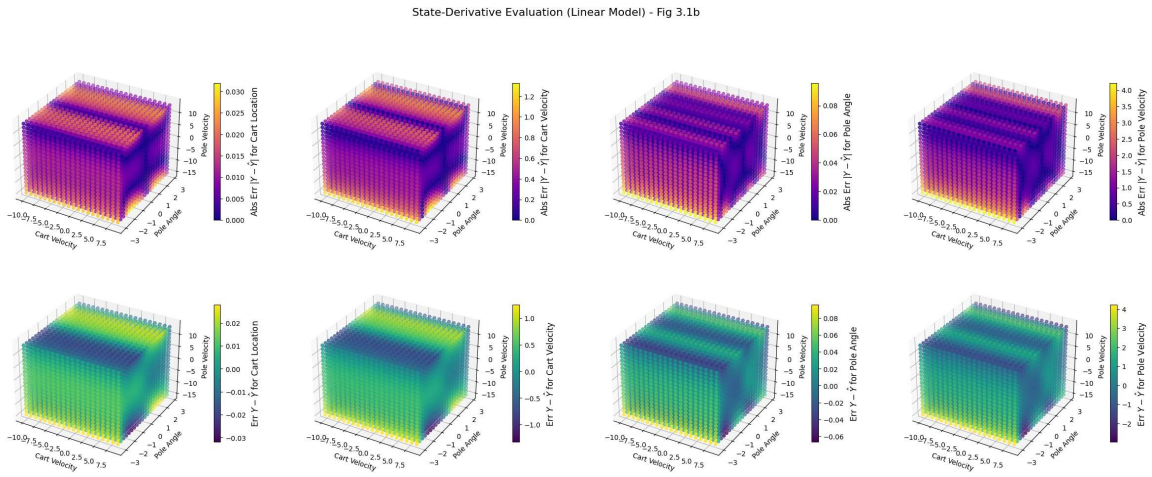
```
Coefficients Matrix
 [[ 1.26486678e-03  4.71491487e-02  4.55843584e-03  1.71801441e-01]
 [ 4.99881768e-02 -4.45589863e-04 -3.59263584e-05 -1.34990723e-03]
 [ 3.97369618e-03  1.48123419e-01  1.43207485e-02  5.39730145e-01]
 [ 8.44971490e-05  4.91685605e-03  5.00674059e-02  4.28497568e-03]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]

Mean-Squared Error [1.00962113e-04 1.59743085e-01 4.96030419e-04 8.4657378
2e-01]
```

Next-State Prediction Evaluation - Fig 3.1a



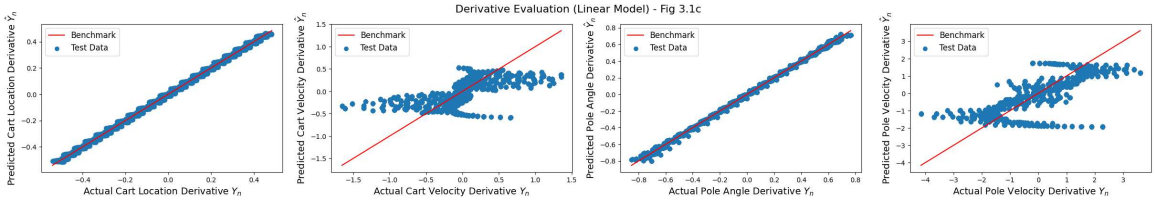
From the generated matrix and low mean-squared error we can see that linear regression as shown in **Fig 3.1a** fits well due to the small timestep and hence next step prediction is relatively accurate for our dataset. **Fig 3.1a** also shows that the cart location does not affect the other variables since the actual X_{n+1} for all the other variables remains the same. It is important to see that upon the variation of the other variables namely cart velocity, pole angle and pole velocity, the prediction and the actual next states can be very different hence showing that although the model maybe accurate at modelling a single variable individually, it is not adequate for modelling the entire system and only the specific variable change.

Moreover if we now map the derivatives for each specific initial state by constructing a grid in 3-dimensional space for the different initial conditions (and let the cart location be an arbitrary value) **Fig 3.1b**, we can see that the cart velocity and pole velocity fittings are actually rather poor (by the scale on the colorbar relative to the limits of the variables)! In fact this is a large downside in a non-linear system since at a specific moment a large divergence in the derivative results in a completely different state space trajectory. Hence we can consider this model to have overfit or atleast unsuitable for this problem.



The error plots in **Fig 3.1b** show that the plots are analogous in error for position and velocity meaning that the errors in cart location are similar to that of the cart velocity although at a much lower magnitude and the errors in pole angle are similar to that of the pole velocity although again at a much lower magnitude. We should also note that although the absolute error is symmmetrical, the actual error is antisymmetrical (ie. involving both positive and negative error). This is very difficult to take into account in a linear model. As shown by the error plots, the model does not effectively take into account the non-linearity during remapping specifically noted by the high error values around the stable equilibrium angle $\theta = \pm\pi$. Surprisingly the model performs well at high velocities around the unstable equilibrium and high velocities around the stable equilibrium are have much large error due to non-linear behaviour. Although it is important to understand that high velocity around $\theta = 0$ results in non-linear behaviour soon after.

By going further and plotting only the derivatives now (**Fig 3.1c**), we can see that the derivative performance for the pole-angle and the cart location is very good however the velocity derivatives cannot be accurately fitted by the linear regression model. Moreover since this is considering only the training data, we have yet to consider the behaviour of the model for a random scan which must be worse than the data the model was trained on. In general this is not the sign of a good model. Moreover we can clearly see the non-linearity of the derivative of the pole velocity (in the contour plot in **Fig 2.2b**) becoming a problem as shown by the fitted regression plot of the derivatives in the pole velocity derivative plot in **Fig 3.1c**.

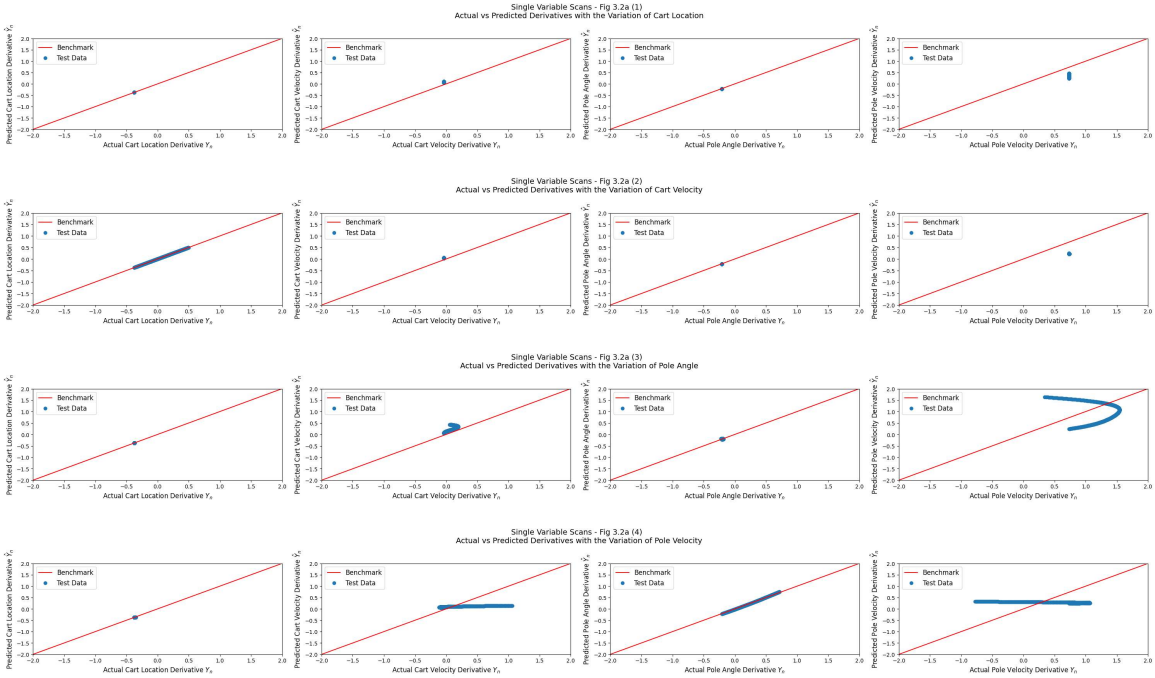


The best predicted derivatives are those of *cart location* and *pole angle*. Since the velocities themselves are derivatives, taking the derivative of the derivatives results in a far larger error. We could consider reducing the sampling time or the integrator time `delta_time` however the linear model will still be unable to account for the non-linearities.

1.3.2 - Random Scan Verification

Now that we have created a linear model, we need to evaluate the model on a test dataset and hence we will be performing random scans to evaluate the model.

Here we repeat the "scans" from section 1.2, and plot simultaneously the real change in state Y_n with the predicted change in state $\hat{Y}_n$ as a function of the scan and consider which variables are predicted well by the linear model, which ones are not and why.



From **Fig 3.2a**, it is possible to see that in general all the variables have poor fitting however relative to the other variables, the *pole angle* performs well except for variation with *cart velocity* (**plot(3,2)**). The derivative of *pole angle* is well fitted with the variation of *cart location* and *pole angle* since deviations are small around the benchmark line (**plot(3,1)** and **plot(3,3)** respectively). *Pole angle* performs poorly with *cart velocity* since the behaviour of the derivative of the *pole angle* becomes non-linear for large cart velocities.

The plots of *cart location* (**column 1**) are very similar to those of the *pole angle* (**column 3**) where the *cart location* fitting is poor with the variation of *pole velocity*. This is understandable since the deviations in cart location are caused by the non-linear momentum/energy transfer between the pole and the cart.

Overall the poor fitting occurs with the variation of *cart velocity* (**row 2**) and *pole velocity* (**row 4**). This shows us that the model is not very accurate at modelling the accelerations which are extremely important in being able to model a non-linear system since the velocity derivatives between timesteps do not have a linear relationship.

It is important to note the magnitude of errors too and this is most notable in **plot(4,2)** and **plot(4,4)** showing that the predicted derivatives of *cart velocity* and *pole velocity* have large errors with the variation of *pole velocity*. This means that accurately being able to model the velocity derivatives with respect to the pole velocity is key to making a good non-linear controller.

Dual Variable Scan

From the single variable scan, we were able to characterise the behaviour of changes in specific variables however we need to explore how different variables interact with each other. As before we identified the cart location is not necessary to characterise the behaviour. Hence by considering the three main variables *cart velocity*, *pole angle* and *pole velocity*, we can plot the 3 dual variable scanned variations for the predicted derivative and compare it with the expected derivative for each of the 4 state variables giving us 12 plots as shown in **Fig 3.2b**.

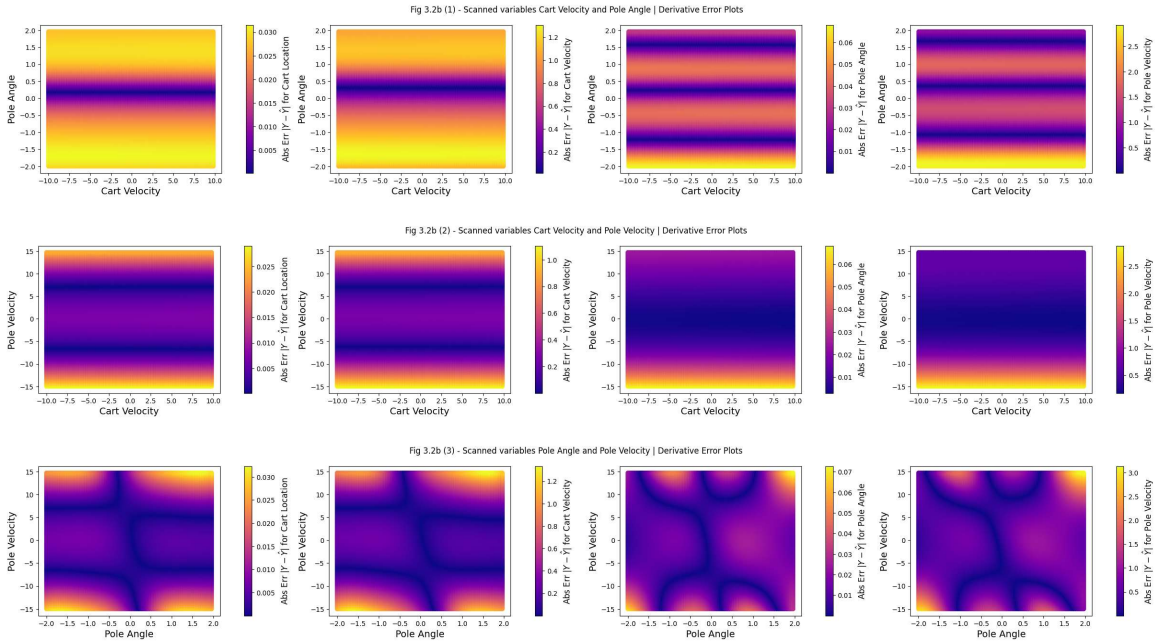


Fig 3.2b are the approximate slices of **Fig 3.1b**.

Again in the dual variable scan graphs, the *pole angle* and *cart location* (**columns 1 and 3**) have about equally good predictions as mentioned in the single variable scans (seen by the colour bar scales). This is because the deviations from the true derivative are rather small and the effect of cart velocity and pole velocity although present are not catastrophically affecting the variable's behaviour. As confirmed in the regression stage, it is possible to see that for any cart velocity the derivative of all the variables is accurate around $\theta \approx 0$.

Cart velocity and *pole velocity* (**columns 2 and 4**) have very similar derivative graphs in the sense that for variations in the pole angle and cart velocity (**row 1**) or for variations in the pole angle and pole velocity (**row 3**), the fitting is very poor as the model cannot model the non-linear behaviour and has atleast overfit to the original dataset. The next state variable for *cart velocity* and *pole velocity* cannot directly be predicted from a linear model since there are now two variables that are non-linearly affecting the next state from any given current state. For high derivatives of both of these variables, the model cannot accurately predict the derivative and for low derivatives, there exist multiple states which can reach these derivatives and hence there are multiple possible solutions which the linear model cannot take into account.

Summary

Linear regression can although accurately calculate the derivative of some variables variables, the greater the order of the derivative the more non-linear the behaviour becomes and the less accurate the model. For the cartpole system, the angle can be predicted to a good extent relative to the other variables but not to the accuracy of being able to simulate the pole angle using just linear regression. The cart location is a rather redundant variable and provides little to no insight in the behaviour of the system and although its derivative is just as accurate as the pole angle, the system would perform slightly better if the cart location was also remapped like the pole angle. Hence we see the analogous representation of the cart location and the pole angle but unlike the cart location, the pole angle has a huge impact on the dynamics of the system.

Similarly we can notice the analogous behaviours between cart velocity and pole velocity where the greatest non-linearity comes from the pole velocity. Since the pole angles are small, the pole velocities are even small and hence the relative errors in the derivative predictions are comparatively very large in comparison to the errors in the cart velocity derivatives. Moreover since the system itself is very much impacted by the pole angle, slight deviations in the pole velocity have a huge impact on the pole angle which in turn drastically affects the state-space trajectory of the system.

1.4 - Model Evaluation

The true test of the model is not how it matches with the gathered data it was fit to, but whether it can predict the time evolution of the physical system. We iterate the model to predict the time evolution of the system, and compare using various initial conditions how accurate the predictions are with respect to the true dynamics started from the same initial conditions. (Note that the model is being used deterministically, with no noise added)

Since the model above predicts the *change* in the state variable, the iterated time evolution is

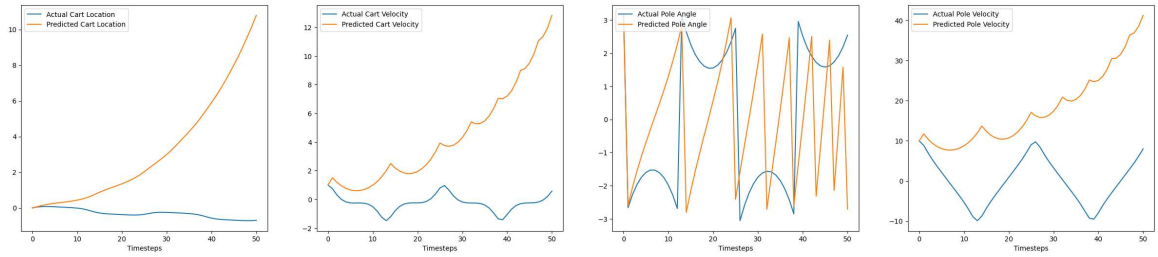
$$X_{n+1} \leftarrow X_n + f(X_n)$$

Below we plot the true time evolution of the system as well as that of our fitted model for many cycles, and for different initial conditions, including ones where the pole makes a full circle.

Predicted vs Real Time Evolution Comparison

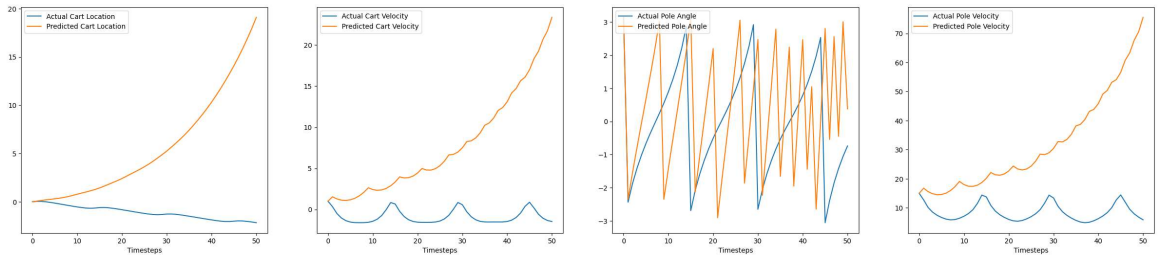
```
Initialised State: [0, 1, 3.141592653589793, 10, 0]
```

Fig 4a - Simple Oscillation | Time Evolution Graphs




```
Initialised State: [0, 1, 3.141592653589793, 15, 0]
```

Fig 4b - Complete Rotation | Time Evolution Graphs



From **Fig 4a** we can see that for simple oscillation, the pole angle prediction is much more incorrect than that of the pole angle prediction during a complete rotation. At first glance this may be surprising since we should be able to model motion about the equilibrium with a linear model however as seen in simple oscillation, the angle changes are non-linear around π whereas in a complete rotation **Fig 4b**, the addition of the difference works very well and the remapping effectively prevents the pole angle from exploding. Nonetheless the time evolution is only accurate for about 20 timesteps beyond which the predicted trajectory is very different.

If we leave the pole angle without remapping, the solution with the iterated model diverges since if a postive difference is continuously added to the pole angle at every timestep then the variable will diverge to $+\infty$ and $-\infty$ respectively when a negative difference is added. The remapping enforces periodicity by only increasing up to π and then returning to $-\pi$. Remapping is not needed in the true dynamics, since that is nonlinear and the angle only appears inside trigonometric functions that are periodic anyway. A non-periodic function will diverge and hence remapping prevents the pole angle from exploding.

All other variables diverge showing that linear regression is a pretty poor model for modelling a non-linear system. The cart location however does have a roughly accurate time evolution for several timesteps before diverging. The worst behaviours are again of course for the cart velocity and pole velocity. The linear model is unable to account for the periodicity and the non-linear relationships among the state variables between timesteps.

Linear Model Exploration Summary

In this section we covered the general dynamics of the cartpole model focussing on the time evolution and state trajectories and then attempted to apply a linear model fitting to simulate the system. By noticing the linearity for certain state variables between timesteps, we constructed a linear model using linear regression primarily one that tries to predict the derivative from the state variables.

To verify the model's performance, we performed random scans with single and dual state variables noting and explaining the behaviours that did not align with our model and considered the ramfications of the errors in certain state variables.

To visualise the performance of the model, we plot the time evolution of the model with the same time evolution plots presented in the dynamics section and notice that the performance is incomparably bad for more than a few timesteps and for most of the variables. By understanding the problems of the model that were analysed during verification, we can hope to build better models for simulating and controlling the non-linear cartpole system.

2 - Non-Linear Modelling and Linear Control

2.1 - Non-Linear Modelling

Nonlinear Model

As we observed above, the linear model is not particularly good. In order to do better, we need nonlinear modelling. Here we build a nonlinear model using a linear regression with nonlinear basis functions. Given a data set of (X, Y) pairs, the model function is given by

$$f(X) = \sum_i \alpha_i K(X, X_i)$$

where the sum runs over the basis functions, α_i are the corresponding coefficients, and K is a *kernel function* that is used to define the nonlinear basis. The kernel function takes two arguments, the first one X is the state vector where the basis function is evaluated, and the second argument, X_i is another state vector which is used to place the basis function somewhere in the state space. To make the basis functions relevant, we take the set of locations $\{X_i\}$ to be a subset of the gathered data points.

If you place a basis functions on every location at which you have data, the functional form above is equivalent to the mean of a *Gaussian process* with covariance given by the kernel *K*. This view of the regression problem is particularly helpful if the data is stochastic, i.e. it has some noise component which we can model.

For the present problem, let us use a Gaussian kernel function,

$$K(X, X') = e^{-\sum_j \frac{\left(X^{(j)} - X'^{(j)}\right)^2}{2\sigma_j^2}}$$

Here $X^{(j)}$ refers to the jth component of the state vector. There is one caveat for using this kernel function in our current situation: one of our state vector components, θ is periodic. It helps quite a bit if we introduce this periodicity in our kernel function, and we can do that by using $\sin^2((\theta - \theta')/2)$ in place of $(\theta - \theta')^2$ in the part of the kernel function that corresponds to the angle variable. The parameters σ_j are *length scale* hyperparameters of the model, and need to be known, guessed or fitted.

If we have N data locations (each is an (X,Y) pair but remember that here X is a vector and Y is a scalar), substituting this data into the model functional form yields the following linear system

$$K_{NN}\alpha_N = Y_N$$

where the subscript N was used to emphasize the size of the array rather than an index. The unknown coefficients are collected into the vector α_N, the elements of the matrix are given by the kernel function,

$$[K_{NN}]_{i,i'} = K(X_i, X_{i'})$$

and Y_N is a vector of the target function values. With Gaussian basis functions, the condition number of the matrix is enormous, and a direct solution of the above linear system would be rather unstable. One way to get around this is to *regularise* the linear system. Tikhonov regularisation is to modify it to

$$(K_{NN} + \lambda I)\alpha_N = Y_N$$

with solution

$$\alpha_N = [K_{NN} + \lambda I]^{-1} Y_N$$

where λ is a parameter. The smaller the values of λ, the closer is the fit to the data, but the more unstable the linear system. Interestingly, this is also exactly the form of the (mean) solution in the Gaussian process regression problem of inference in the presence of noisy input data, with the identification that λ is the variance of the data noise.

Suppose we collected N pairs of (X, Y) data pairs, and we choose a subset M of the X locations to serve as basis function centres. The linear system is then not square:

$$K_{NM}\alpha_M = Y_N$$

where again the subscripts indicate dimensions, K_{MN} is an $M \times N$ matrix with elements corresponding to the M basis locations and all the N data point locations. We have fewer unknown coefficients than data points, so the problem is over-determined. The least squares solution would be

$$\alpha_M = [K_{MN}K_{NM}]^{-1}K_{MN}Y_N$$

i.e. using the pseudoinverse rather than the inverse. The matrix in square brackets will in general be uninvertible or very badly conditioned, so again we need to regularise. Interestingly, rather just adding a multiple of the identity matrix like in the square case, we can look to Gaussian process inference for a better idea. It turns out that a *sparse Gaussian process* precisely corresponds to this case. There, the model is written as a conditional probability not on the original N data values directly, but on the unknown function values at the M data locations that are chosen for the basis function centers. The vector of linear coefficients of the fitted model are then given by

$$\alpha_M^{(j)} = (K_{MN}K_{NM} + \lambda K_{MM})^{-1}K_{MN}Y_N^{(j)}$$

where K_{MM} is an $M \times M$ matrix with elements that are given by the kernel function evaluated between the locations selected as basis locations. The interpretation of this is that the least squares system is Tikhonov regularised but the regulariser is evaluated in the kernel-norm, rather than the Euclidean norm.

To apply all the above to the cartpole system, we have to create four separate models for the four components of the state vector, these are indexed by j. In the absence of data noise to guide the selection of λ, you need to experiment with different values (e.g. between 1E-6 and 1E-1, on a log scale). You will also need to select the length scale parameters σ_j for each state variable. A good start is the standard deviation of the state variable in your dataset. Note that in the Gaussian process inference view of this problem, these hyperparameters would be optimised by minimising the log-likelihood, feel free to research this and experiment with it if you have time.

Note: You should never use the `np.linalg.inv` function to invert the matrix when solving a linear system, because that can be numerically unstable for such ill-conditioned matrices, but instead use `np.linalg.lstsq`, which solves equations of the form $Ax = b$ directly in a least-squares sense.

Using the same data we gathered earlier to train the linear model, we create a new non-linear model with the target function being the change in state after one step as before for the linear model.

```
Sigma Vector: [0.57732164 5.77267222 1.81362431 8.65897871 1.        ]
```

Scatterplot Verification

Here we verify using scatterplots that the nonlinear model indeed fits the data.

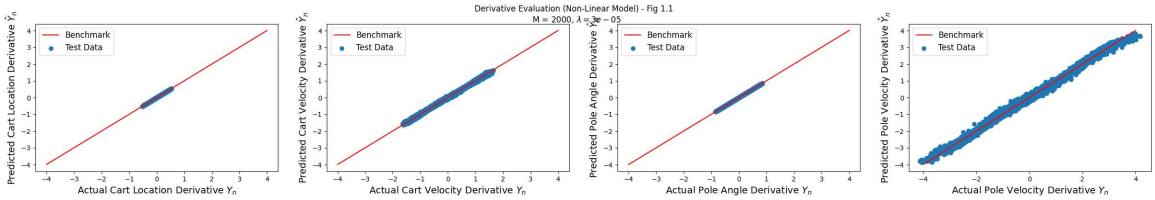


Fig 1.1 shows that the cart location and pole angle perform really well in comparison with the cart velocity and pole velocity, similar to the linear model. However there is a very large improvement in the predictive power of cart velocity and pole velocity with the non-linear model than with the linear model. This is what we required as shown by contour plots of the linear model in **Section 1** hence this improvement is significant for an accurate model system to be constructed.

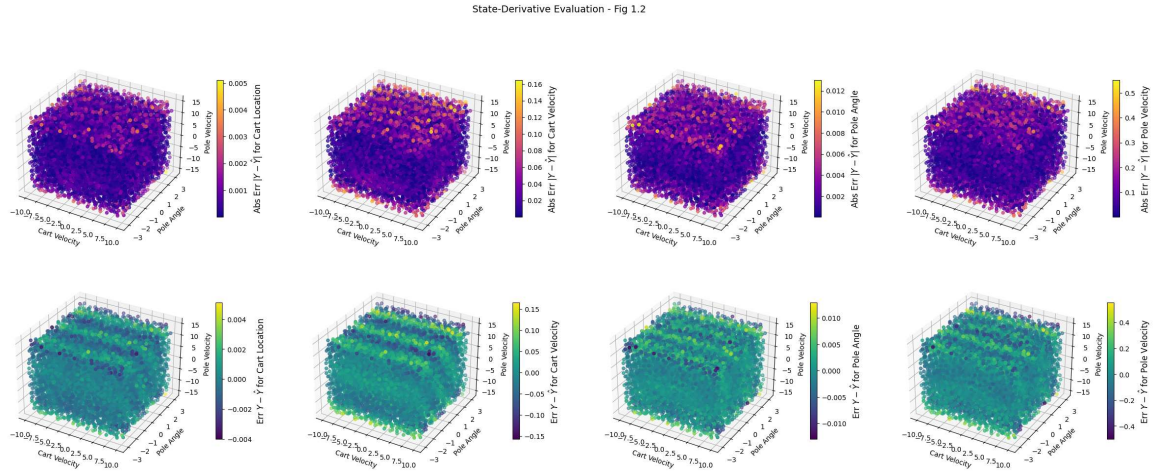


Fig 1.2 (1) shows that in general the absolute errors are very low. We can simply compare the magnitudes of the colour scale with that of the linear model in **Section 1** and see that both cart velocity and pole velocity have a large reduction in error and the magnitude of error in all the variables are comparably much smaller to those from the linear model. Generally the greatest error exists at the extreme points of each variable and hence the corners of the cube have the greatest error. This would be expected as this is where the system is most non-linear and where there is likely to be fewer basis function and training points due to the sobol generation method.

From the relative error plot **Fig 1.2 (2)**, we are able to see bands of positive and negative error with the variation of pole angle. This will have an adverse impact on the ability to control the system however this is still a very big improvement over the linear model for modelling the dynamics of the system.

Model Convergence

Above we created an appropriate non-linear model using the convergence properties we identified in this section. We considered the convergence of the model (i.e. the systematic reduction in error) as a function of increasing data amount, and the increasing number of basis functions as well as considering the impact of the regularisation parameter. We selected the data locations for the basis randomly from the data. We also tested sobol sequences for generating these basis vectors; these provide a set of locations in an arbitrary dimensional unit cube that are "nicely" spaced out for sampling, better than random draws. The `sobol_seq` module is used to generate such locations.

We considered multiple different methods for creating the basis and the dataset of this model:

1) Training dataset from random vectors in range and basis vectors picked randomly from the training set

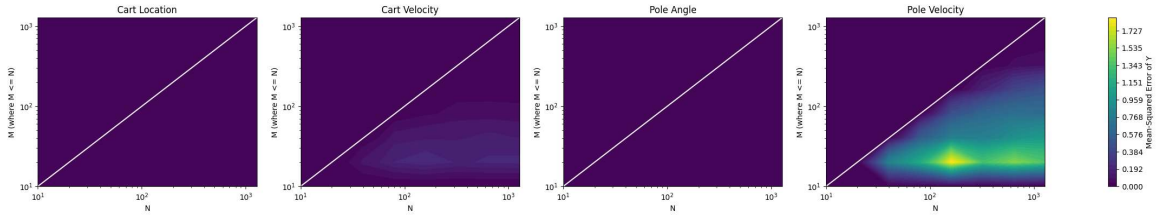
2) Training dataset from random vectors in range and basis vectors from sobol sequences

3) Training dataset from sobol sequences and basis vectors picked randomly from the training set

4) Training dataset from sobol sequences and basis vectors from sobol sequences

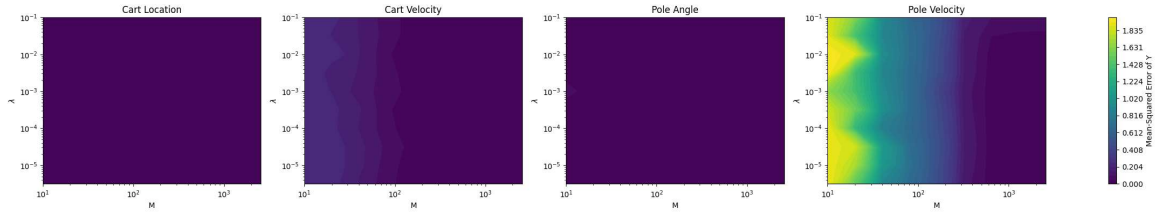
1) was clearly the least efficient and had the lowest performance. 2) was erratic in the sense that since the training set was random, certain initial conditions performed well at times and not at others and hence was unreliable for creating a working model. This could be seen by the contour plots during development. 2) is also not the most appropriate since we do want basis states relevant to our dataset. 3) This worked very well relative to 4) since we do not really want a set of sobol terms starting at the all-zero vector for the basis vectors. Sobol states worked well for the dataset as the data becomes more diverse with fewer points. Having a randomly picked basis vectors from the sobol training set meant that the basis functions were of the sobol form but did not necessarily have to occupy the entire space.

N-M Analysis - Fig 1.3



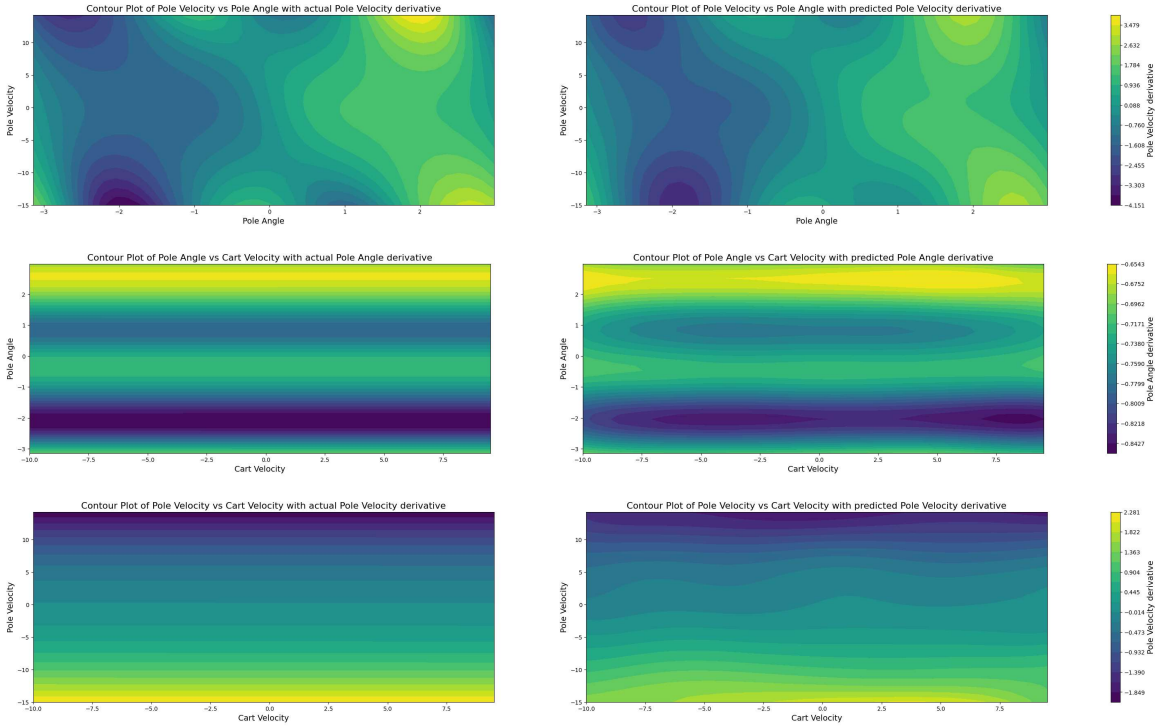
If we compare the scales, we see that the only relevant plot we need to be concerned about is that of the pole velocity. Here we see that the error reduces monotonically as both M and N are increased. Considering that this is the mean-squared error of Y, a value of 1 for the pole velocity is very high! It means that on average all state vectors have incorrect changes in pole velocity by $1ms^{-2}$. Of course this is not the case as some states have much larger errors and generally the basis functions perform well as seen by the contour plot. However we do have to use large amounts of data to circumvent such large errors. This is why we selected $M = 2 \cdot 10^3$ and $N = 10^4$ which is not actually represented on the graph here as the computation cost for generating the contour plot for more iterations with higher N and M was high.

λ-M Analysis - Fig 1.4



From the above contour plot, we see that the error is minimised for all state variables when M is large. The value of the regularisation parameter itself does not seem to affect the error as much and hence we simply use $\lambda = 3 \cdot 10^{-5}$ which is a minima for a constant low M according to our contour plot. We do again see that the most affected variable by the regularisation parameter is the pole velocity.

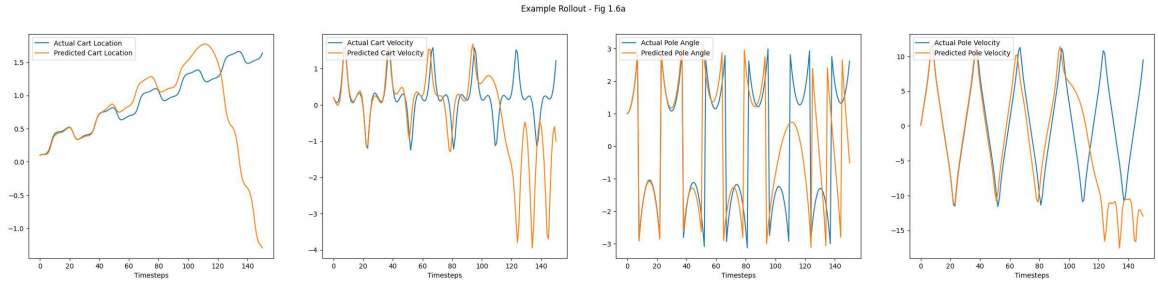
Contour Analysis - Fig 1.5



In **Fig 1.5**, we see that the non-linear model is excellent at emulating the state space. The highly independent bands are more difficult to model with gaussian basis functions although this could be achieved by handpicking the sigma vector of the basis rather than defaulting to the standard deviation of the dataset. Nonetheless the more important aspect of capturing the contour of the pole velocity with respect to pole angle was successful and shows that the fitting error is low unlike the linear model. However, this does not suggest anything about the dynamics and therefore we will need to explore this later.

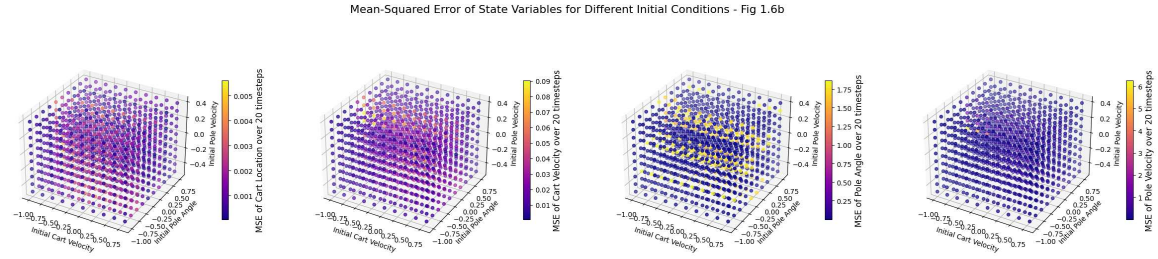
Model Testing

We perform roll-outs to see how closely the iterated model matches the real dynamics for a wide range of sensible initial conditions. After a series of testing, we find a well-performing set of initial conditions and visualise the extent of the performance by performing the rollout. The example rollout shows an initial condition for which the model performs very well.



In the case of our example rollout, the model is accurate for up to 3 cycles which in this is approximately 80 timesteps. If we compare this with the pointwise accuracy on a random test data, this is generally very poor. Our original training dataset used the sobol states and hence this would be expected.

The model accuracy is very much dependent on the initial condition. Hence we can generate rollouts for a grid of initial conditions independent of the cart location (mainly just to visualise the weak spots of the non-linear model) and quantify the error for a small number of timesteps. Selecting the duration of the rollout was a difficult task but the model performs well up to a maximum of 80 timesteps and the worst was approximately 10 timesteps. Therefore selecting 20 as the duration seemed appropriate such as to capture the error behaviour at the extremes while providing confidence for the well-performing initial conditions.



We see that the poorest performing zones include the cart velocity when the initial pole angle is around 0, the pole angle when the initial cart velocity and initial pole velocity have larger magnitudes and the pole velocity when the initial pole angle is around 0 and the cart velocity is larger. These are the same flaws as those that can be identified in the contour plots. Here we have zoomed into the initial conditions that most affect the pole around the unstable equilibrium. Analysing larger initial velocities and angles only resulted in larger errors in similar distributions but of greater magnitude such that patterns were indiscernible.

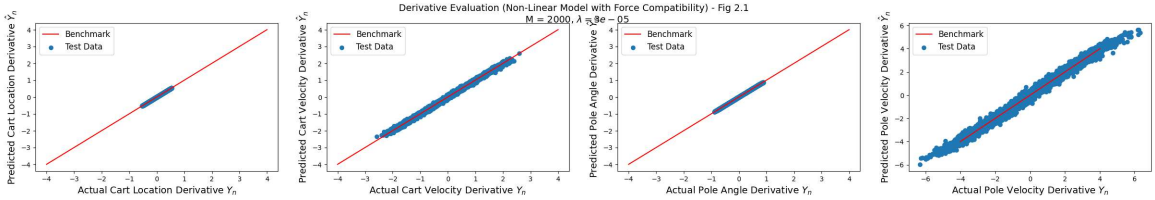
2.2 - Force Input Dynamics Modelling

When the `performAction` routine is called, it takes a signed scalar which is interpreted as an external force on the cart. The value is passed through the `tanh` function before being interpreted as a force, this prevents the application of excessively large forces (the transformation is controlled by the `max_force` variable inside the `CartPole` class.)

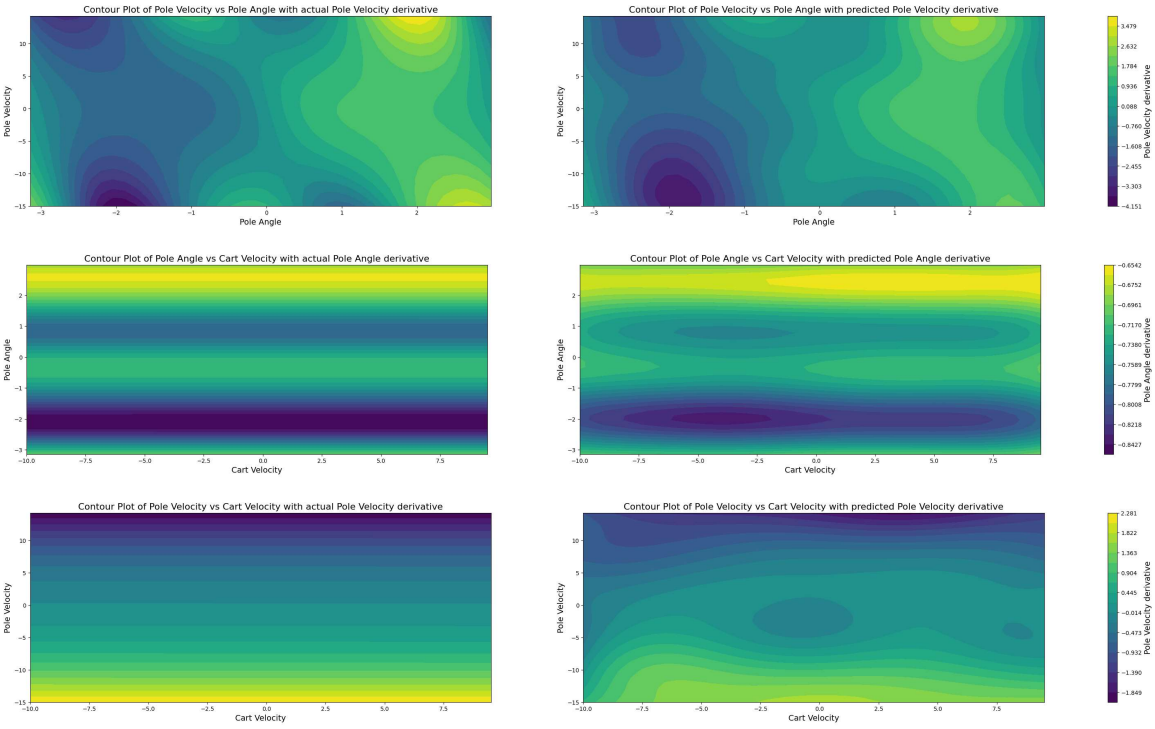
Our models (both the linear and nonlinear) have been modified to take account of the new state variable, the force action F (i.e. the system now has 5 inputs, including the force F, and 4 outputs after a call to `performAction`).

We collect new data using quasi-random sobol sequences but this time include the action. By plotting scatter-plots, comparing the contours of the model with the real system and performing roll-outs, we show that the models can predict the change in the state variables to a good degree of accuracy for a limited number of timesteps.

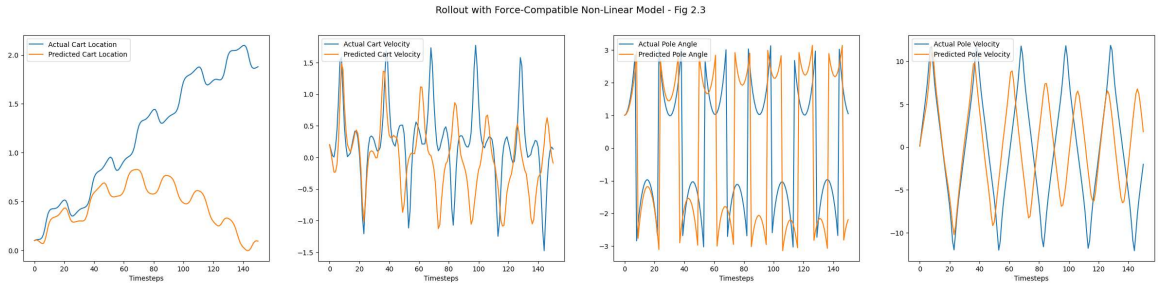
```
Sigma Vector: [ 0.57732164  5.77267222  1.81362431  8.65897871 11.5456875
7]
```



Contour Comparison of System with the Non-Linear Model - Fig 2.2



If we compare the force-compatible contour plots in **Fig 2.2** with that of the force-incompatible contour plots in **Fig 1.5** in this section, we see that the new force-compatible model is less accurate for all the variables. This is understandable as we now have a greater state space but the same amount of data. The main issue with this model currently is the relatively high inaccuracy in pole velocity against cart velocity. This is primarily the drawback of the designed model and would be best fixed by increasing the data and manually selecting the sigma vector of the model. Using a larger dataset was computationally slow and hence avoided. We continue to use the same amount of datapoints in the training set $N = 10000$ and the same number of basis functions $M = 2000$.



The model does not work well for cart location relative to the force-incompatible non-linear model **Fig 2.3**, however the accuracy for the other variables is relatively accurate for some initial conditions as analysed by the contour plots in **Fig 2.2**.

2.3 - Policy Iteration

Having developed a good model for the dynamics, now it is time to control the system. In this exercise we will create a controller to achieve a desired state by parametrising the actions of the controller, and optimising those parameters through evaluation of the performance. This is called *reinforcement learning* in general. Using the interaction of the controller with the system to improve it is "direct policy search". There are many other, more sophisticated strategies.

Policies

A *policy* is a function $p(X)$ that defines what the action should be given the other state variables. The goal is to find a policy function that when enacted, gives rise to the desired behaviour, in this case the pole being balanced around its unstable equilibrium position.

Objective

In order to optimize a policy, we need to define, mathematically, what we want to achieve. That is captured in an *objective function* (also called *loss function*), a measure of how close we are to the desired behaviour. In the present case, we want the pole to be upright, so we could use the *loss function*

$$l(X) = -\cos\theta$$

But it is better, and more general, to define a *target state*, X_0, which we want the system to achieve, and use a loss function that increases when the distance of the state from the target is larger. The following loss function achieves this,

$$l(X) = 1 - e^{-|X-X_0|^2/2\sigma_l^2}$$

where σ_l is a scaling factor (you could introduce a separate one for each component of the state variable). The target state for the cartpole system is $X_0 = [0, 0, 0, 0]$. The advantage of this form is that for large departures from the target, the loss is independent of the state. This expresses the notion that if the pole is far away from being upright and stationary, we do not much care what it is doing. The above loss functions are for a given state. We wish to keep the pole upright continuously, so the total loss of a trajectory should be a time integral (sum, in practice) of the pointwise loss of the state over some interval,

$$L = \sum_{i=1}^{N} l(X_i)$$

The `CartPole` class contains a function to evaluate the above pointwise loss $l(X)$.

Linear control

We start with defining a linear policy,

$$p(X) = \mathbf{p} \cdot \mathbf{X}$$

with unknown coefficent vector $\mathbf{p}$.

We evaluate the loss function for the trajectory of a rollout using a short time horizon long enough to capture 1-2 oscillation periods. We plot an initial policy grid search to visualise the policy optima and then we optimise the unknowns in the policy to minimise the loss function. Since there not too many variables, this can be very easily done. For low dimensional optimization problems without gradients available, we use the `Nelder-Mead` method.

We explore the loss as a function of the elements of $\mathbf{p}$ and find elements of $\mathbf{p}$ that are able to stabilize the pole when started just slightly displaced from the upright unstable position.

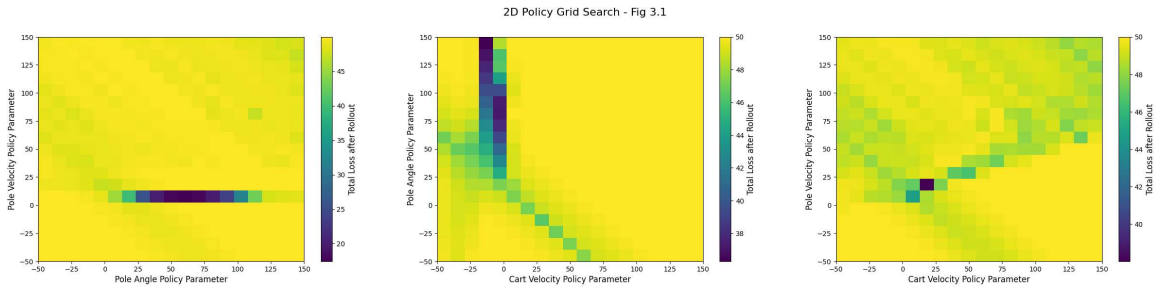


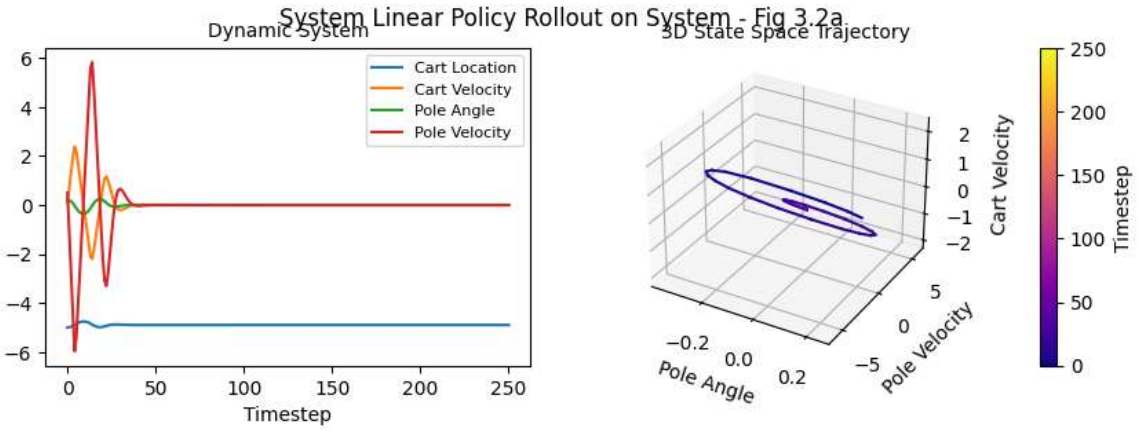
Fig 3.1 was used as a preliminary analysis for confirming expectations of the linear policy. Since this is a 2D policy search, the absolute minimum loss is not neccessarily what we desire but we can check the minima and recognise that the optimal parameters must be within reasonable distance to the grid search optima and exist in some local optima of the search.

Note: We know that the cart location does not affect the other variables and hence cart location has been omitted from the policy search. We can also consider the cart location policy parameter for the cart location to be 0 (ie. force the first policy parameter to equal 0) since the dynamics of the system does not depend on the cart location and therefore the control cannot be dependent on the cart location. Although the extra degree of freedom (by not omitting cart location) may provide better fitting for specific initial conditions, this would not necessarily be the best approach to policy iteration. It would be best to exclude the variables that should not affect the policy. This is dependent on what we desire for ideal control.

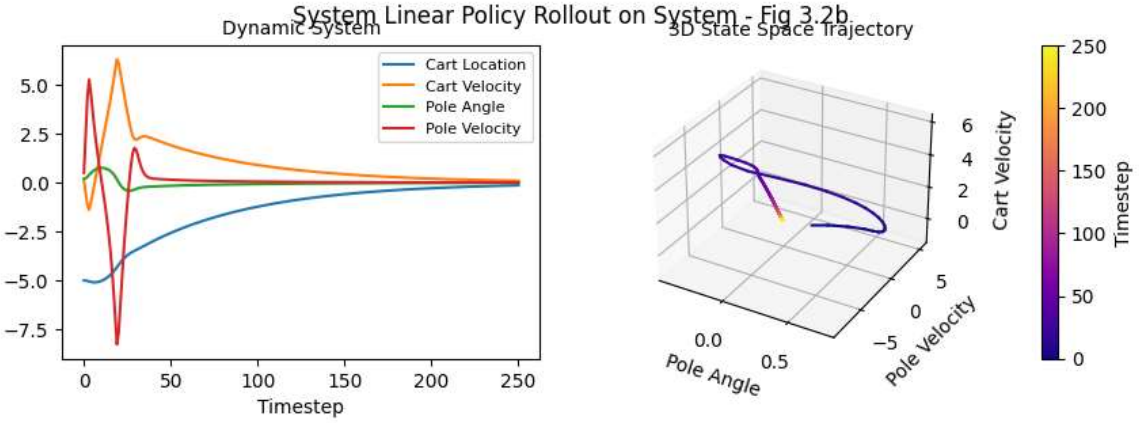
Below we consider two optimal policies, one independent and one dependent on the cart location and analyse their rollout properties.

Now we test out our optimal policy by performing rollouts to test if the policy is successful at stabilising the pole.

```
Optimal Policy Vector (Unaffected by Cart Location) [  0.         110.0163
2447 148.30573133  44.40498228]
```




```
Optimal Policy Vector  [ 20.29414328  36.61783872 143.43151235  21.0077210
7]
```

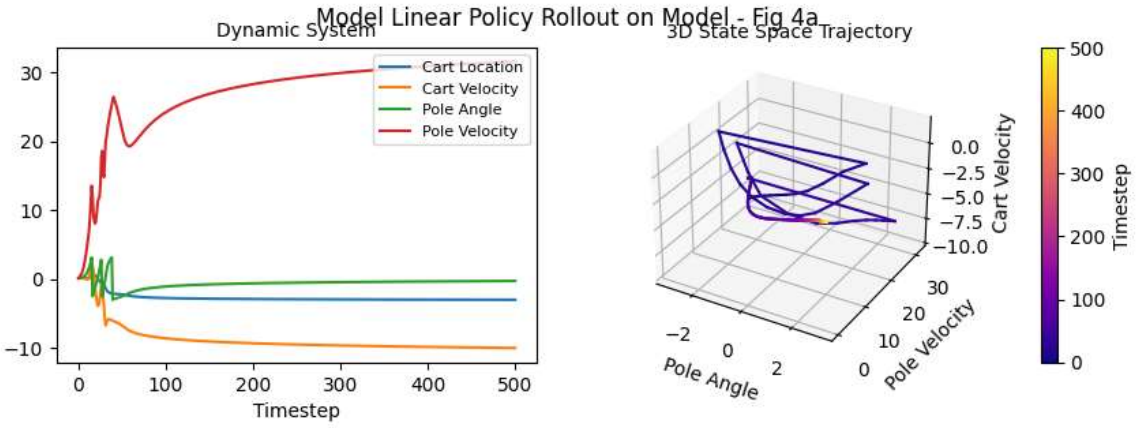


For the cart location independent policy, the timeseries plot and the state space trajectory shows convergence to approximately the init state $(\alpha_0, 0, 0, 0)$ for some arbitrary initial cart location α_0 for the optimal policy vector. In the rollouts, we have set the initial cart location $\alpha_0 = -5$ for better visibility of the other variables in the plot. The initial conditions have been chosen to have the largest possible deviation from the all-zero vector such that the pole remains stable with the optimal policy vector. The plots show that the cart location independent policy is actually a lot more well-behaved and robust than the dependent policy which requires a lot more timesteps for the benefit of bringing the cart location to 0 (which only works for some initial cart locations α_0 in a suitable range eg. $|\alpha_0| \lessapprox 5$). Beyond the safe cart location limit the dependent policy fails to stabilise the pole.

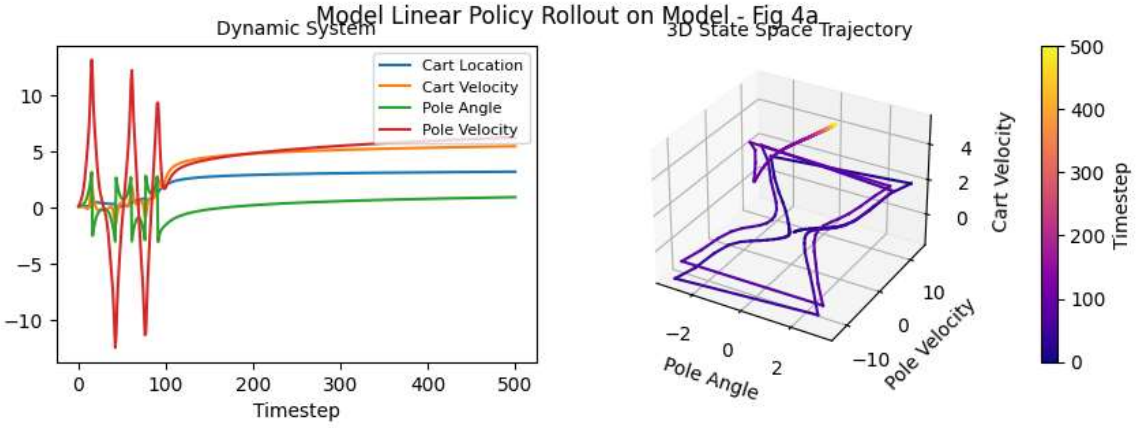
2.4 - Model Predictive Control

In the previous task we used the real dynamics to evaluate a policy but now we will use our constructed model of the dynamics (the nonlinear one from Task 2.1) to optimise the policy parameters by testing them on model-rollouts. The time horizon (number of steps) has been limited such that the model is still approximately accurate.

```
Optimal Policy Vector (Unaffected by Cart Location) [ 0.         -1.674942
16  5.33573274 -0.87333579]
```

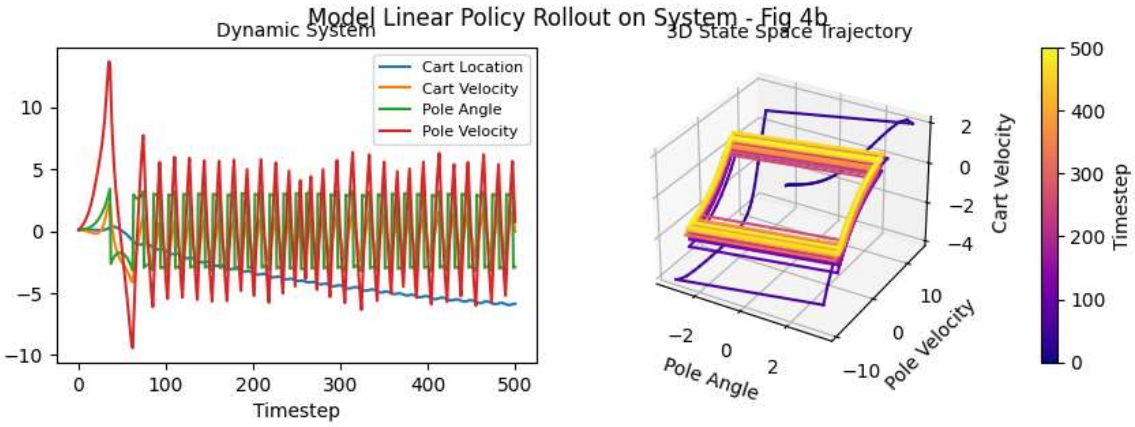



```
Optimal Policy Vector  [ 0.53744497  0.7811186   0.33574928 -0.08545683]
```

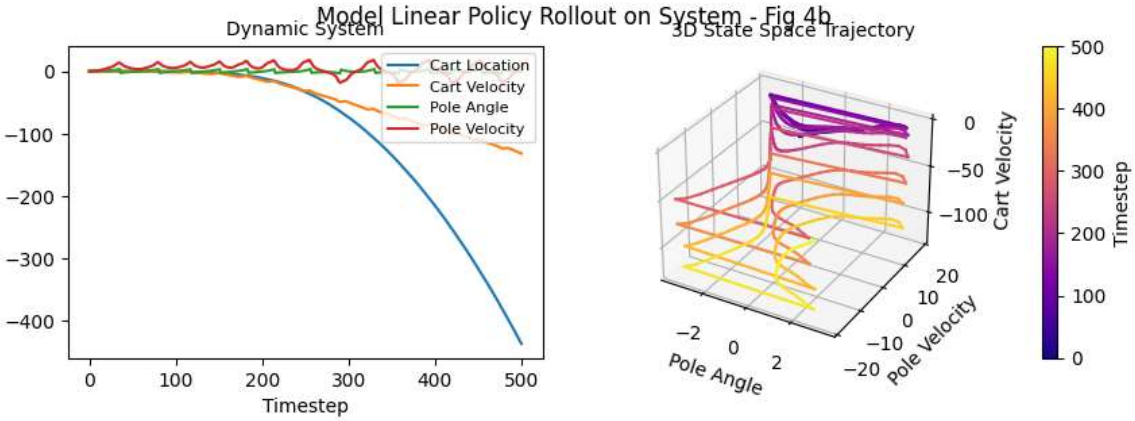


This clearly does not resemble the required control however now that we have created a policy using our modelled system, we can test this on our real system to check if the model system was adequate in capturing the behaviour and whether the non-linear model policy solution was enough for controlling the system.

```
Optimal Policy Vector (Unaffected by Cart Location) [ 0.         -1.674942
16  5.33573274 -0.87333579]
```




```
Optimal Policy Vector  [ 0.53744497  0.7811186   0.33574928 -0.08545683]
```



In the first policy we observe that the pole velocity is indeed decaying which is indeed useful behaviour however the pole angle slowly converges and oscillates just about the stable equilibrium. This can be seen by the state-space trajectory where ideally we want to be right at the centre near the initial condition and instead it oscillates around a similar cycle to a default rollout. the policy attempts to lift the pole with small forces with quick changes and hence is unsuccessful at lifting the pole. The state-space trajectory shows that the policy is unable to leave the rectangle and reach the unstable equilibrium. If we now consider the second policy we see exploding behaviouur and hence it is clear that model predictive control was unsuccessful.

3 - Sensitivity and Stability

In this section, we go back to the beginning of the project, modify the problem, introducing noise in various forms, and observe its effect. Our goal is to understand the effect of noise on both our linear model and non-linear model and then compare this with the performance of the linear controller.

3.1 - Noisy Observed Dynamics

Here we introduce noise in the *observed* dynamics (but not in the real dynamics of the system). We now refit the models (linear and nonlinear), and characterise the degradation in the prediction accuracy. This was achieved by creating a function `noise_added_data(Ys, factor)` which returns a noisy dataset for Y given the original dataset for Y.

3.1.1 Linear Model Degradation (Observation Noise)

We perform the linear regression on the data with added white gaussian noise (mean 0) with the standard deviation in some proportion to the standard deviation of the differences Y. If we add any noise to Y then the accuracy reduces and we can quantify this by noting the mean-squared error of the fitting.

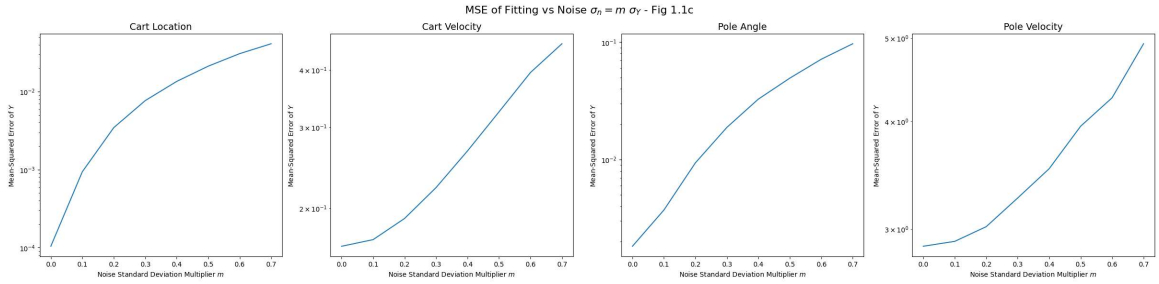
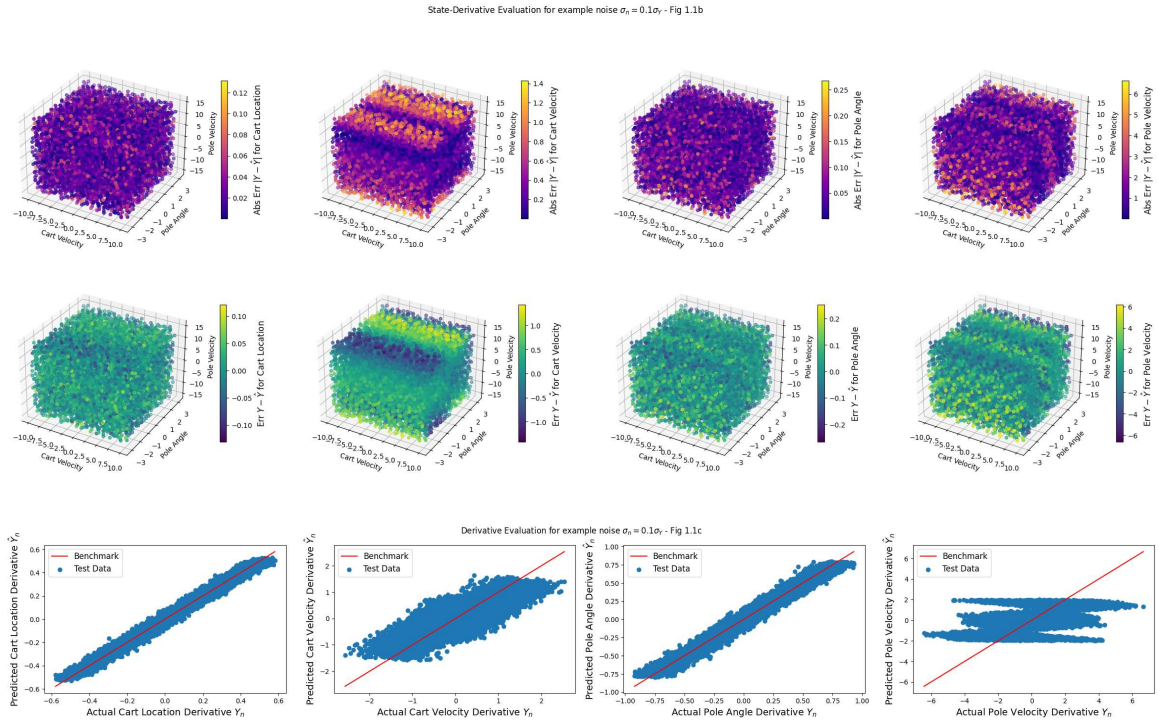


Fig 1.1a shows that as we increase the standard deviation of the noise, the mean-squared error of the fitting increase exponentially! This is very significant and it suggests that our model is not compatible with observed noise.

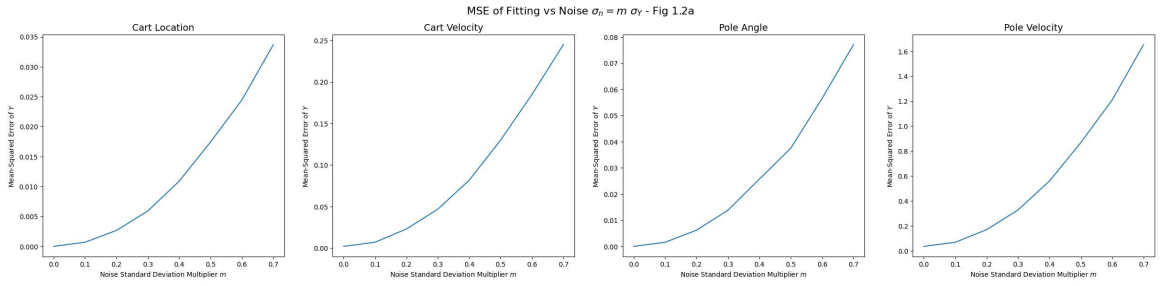


In **Fig 1.1b and Fig 1.1c**, we consider example noise to be only a tenth of the standard deviation of Y.

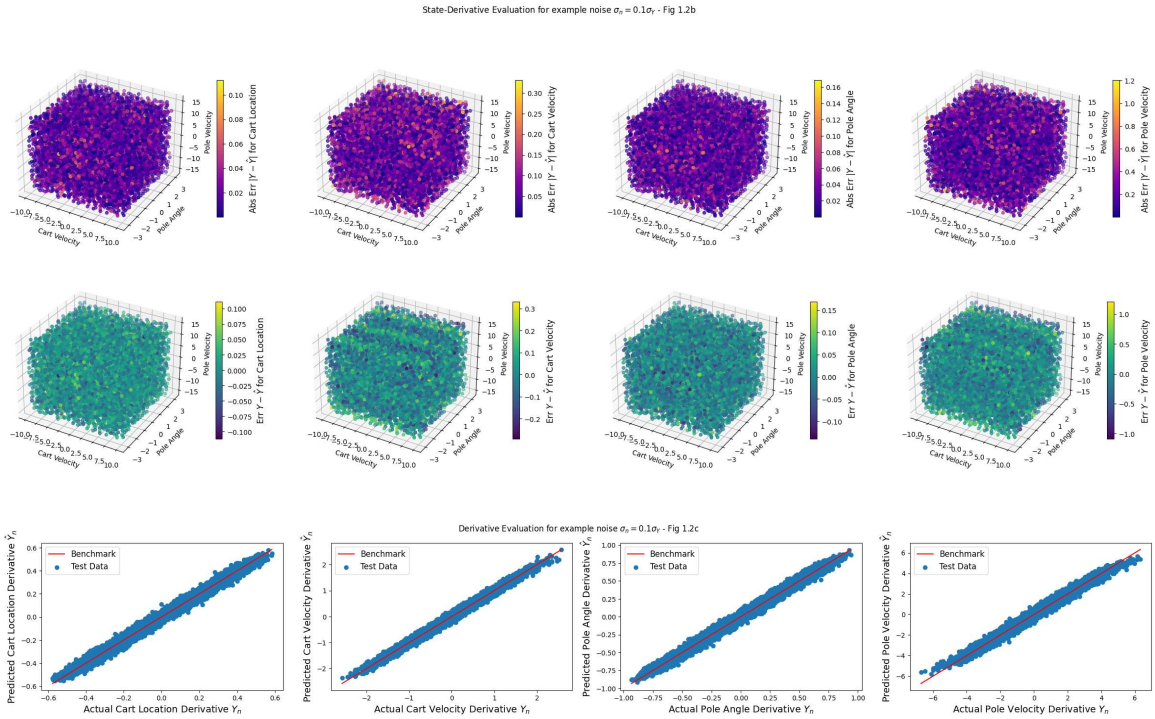
The plots in **Fig 1.1b** show that the degradation is severe for even very little noise. The error in Y becomes increasingly large and even the cart location and pole angle predictions worsen significantly. **Fig 1.1b** shows that the patterns emerging in the contour plots are now very distorted and barely visible with the variation of pole angle or pole velocity. This is because certain points are much more affected by the noise and hence the colour scale has now increased in magnitude. The predictions for the cart location and the cart velocity suffer less but this is mainly due to the fact that the magnitude of X is larger and less important for the state-space trajectory of the system. If we compare these to the plots in **Section 1** while considering the colour scales, the disparity is very clear. Generally the error has increased by large magnitudes for many of the points with the variation of pole angle and pole velocity while the cart location and cart velocity are much less affected.

If we now consider Y for each of the variables with the predicted value $\hat{Y}$ as in **Fig 1.1c**, we see that the cart velocity is also underperforming and is indeed very much affected by the addition of noise. However, if we compare the plots to those in Section 1, the error only increased by a smaller amount because the changes in error for the well performing points increased a lot more than the poorly performing points. This is not adequately shown by **Fig 1.1b** as we only see the absolute error and not in comparison with those of the noise-free case in Section 1.

3.1.2 - Non-Linear Model Degradation (Observation Noise)



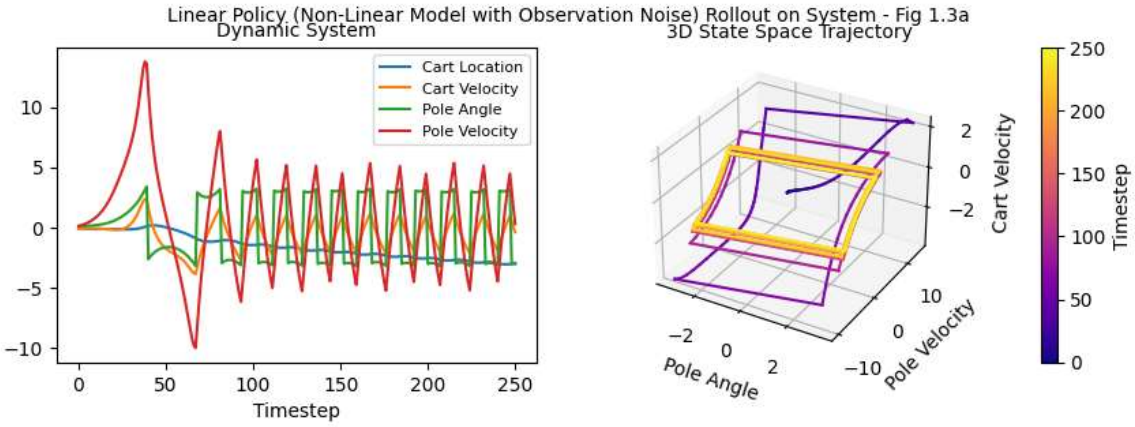
The non-linear model worsens polynomially (almost linearly) with the standard deviation of the noise. This is much better at being able to predict in the presence of noise in comparison to the linear model. Unlike the linear model, providing more data and increasing the number of basis states improves the robustness to noise. This is due to the fact that the function is highly malleable and diverse unlike the linear model and therefore the learning capability of the non-linear model is far greater than the linear model. The linear model quickly overfits to the data and is unable to consider the impact of variation across all the state variables.



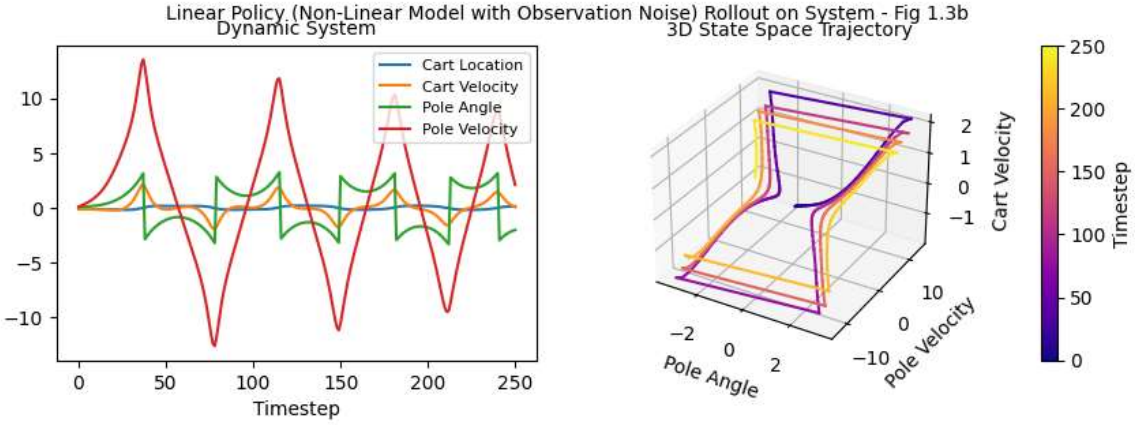
3.1.3 - Linear Policy using Non-Linear Model (Observation Noise)

To understand the effects of noise on the the linear policy, we re-optimise the policy on the new non-linear model trained on noisy observed data and then perform the rollout on the actual system to check its performance and contrast it with the noise-free case. Again we consider a small amount of gaussian noise with standard deviation $\sigma_n = 0.1\sigma_Y$.

```
Optimal Policy Vector (Unaffected by Cart Location) [ 0.         -1.566392
88  4.56656489 -0.6416674 ]
```




```
Optimal Policy Vector  [-0.7518163  -1.22761993  0.59205305  0.07714537]
```



The noisy case of model predictive control does indeed encapsulate some decaying oscillatory behaviour however it is unable to bring the pole to the unstable equilibrium. The failure to control can be attributed to the fact that the model system is not accurate but the policy itself is not by design very poor. It is possible to see in the second policy at approximately timestep 50 that the pole is almost brought to a stop. The pole velocity in general continues to decay although not continuously and eventually oscillates about the stable equilibrium. The model also lacks the ability to accurately simulate the system for a large number of timesteps meaning that the poor performance is in part due to the accuracy of the model but by comparing the noisy case with that of the noise-free model, we see that the behaviour is comparable to that of the noise-free case. By increasing the amount of data used to train the model, we can definitely improve the model however we cannot guarantee the creation of an optimal linear policy. The generated policy vector in the noisy case is highly different in comparison to that of the noise-free case.

3.2 - Noisy Actual Dynamics

If we introduce noise in the actual dynamics of the cartpole system, we are essentially adding noise to the variables before the integration. Hence the proportion of noise due to the summation during integration is actually much larger and the dynamics of the system is much more affected as the smaller the dt the better the integration but the greater the impact of noise on the integration. We will now consider the noise as having a standard deviation relative to the change in state at that timestep rather than the standard deviation of the whole dataset as we had considered before since we no longer have a dataset that we can use to add noise internally. We now quantify the behaviour by repeating the tests of the previous task.

The Cartpole class has been edited accordingly with a `performActionNoise(action, noise_factor)` method.

3.2.1 Linear Model Degradation (Integral Noise)

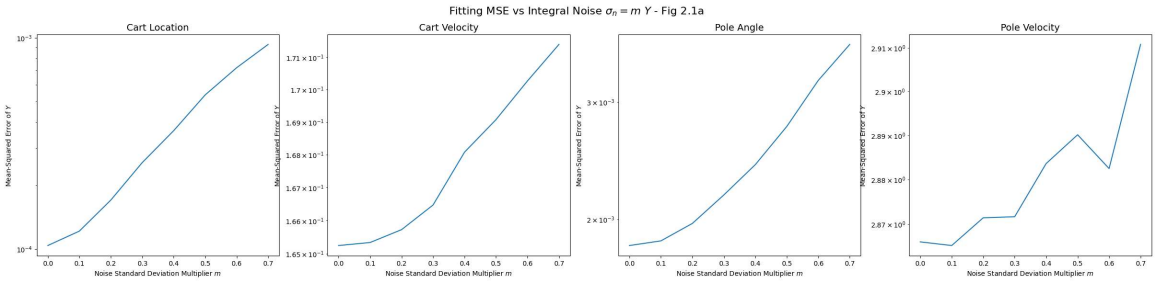
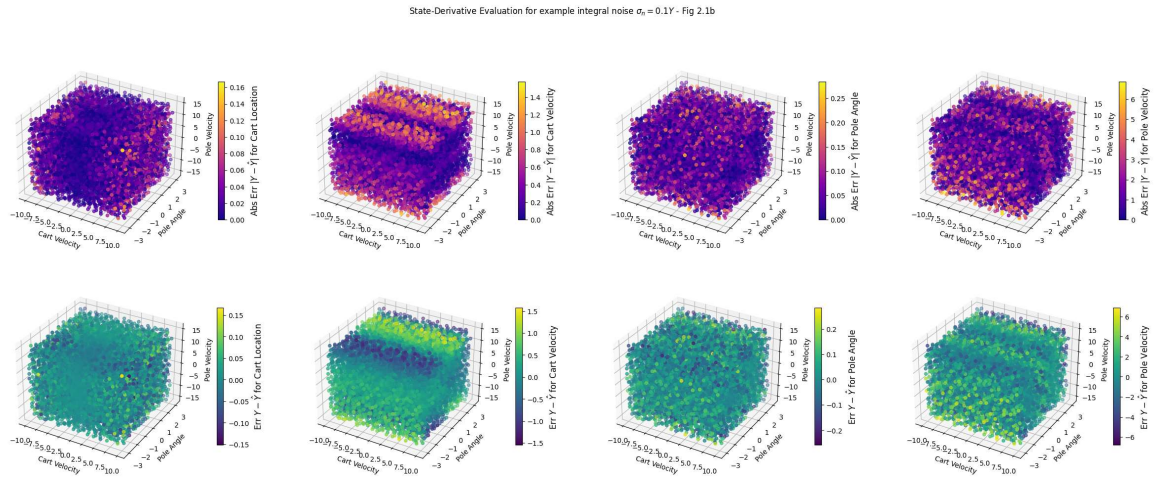


Fig 2.1a shows that the error in the cart location and the pole angle both increase exponentially with added noise however the cart velocity and the pole velocity both increase approximately linearly. This is somewhat surprising as we would expect the velocities to be worse. However we should that note the errors of cart location and pole angle are substantially smaller as given by the scale and hence only appear linear. Moreover the error calculations are not as smooth as for the noisy observations case and hence the results here have a large degree of uncertainty.



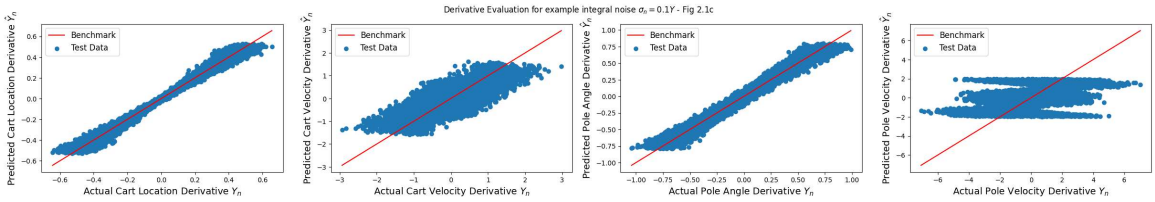


Fig 2.1c shows that the linear model is much worse for the integral-noise case than for the observed-noise case. If we compare this with the noise-free linear model, we see that both the cart location and pole angle have a significant increase in error and the cart velocity performs much poorly. The pole velocity was already bad in the noise-free case but the noise does worsen it which can be seen in **Fig 2.1b (1)** where the errors in pole velocity are distributed generally randomly in the space. The errors at the extremities are heightened and all variables are now poorly predicted. This shows that the linear model is a poor model as not only does it not fit well in the noise-free case but performs much worse in both noisy cases. The linear model substantially lacks the ability to learn the function accurately.

3.2.2 - Non-Linear Model Degradation (Integral Noise)

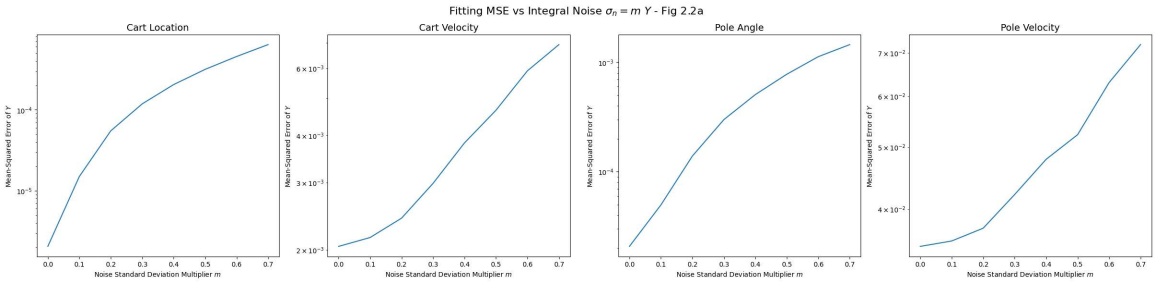
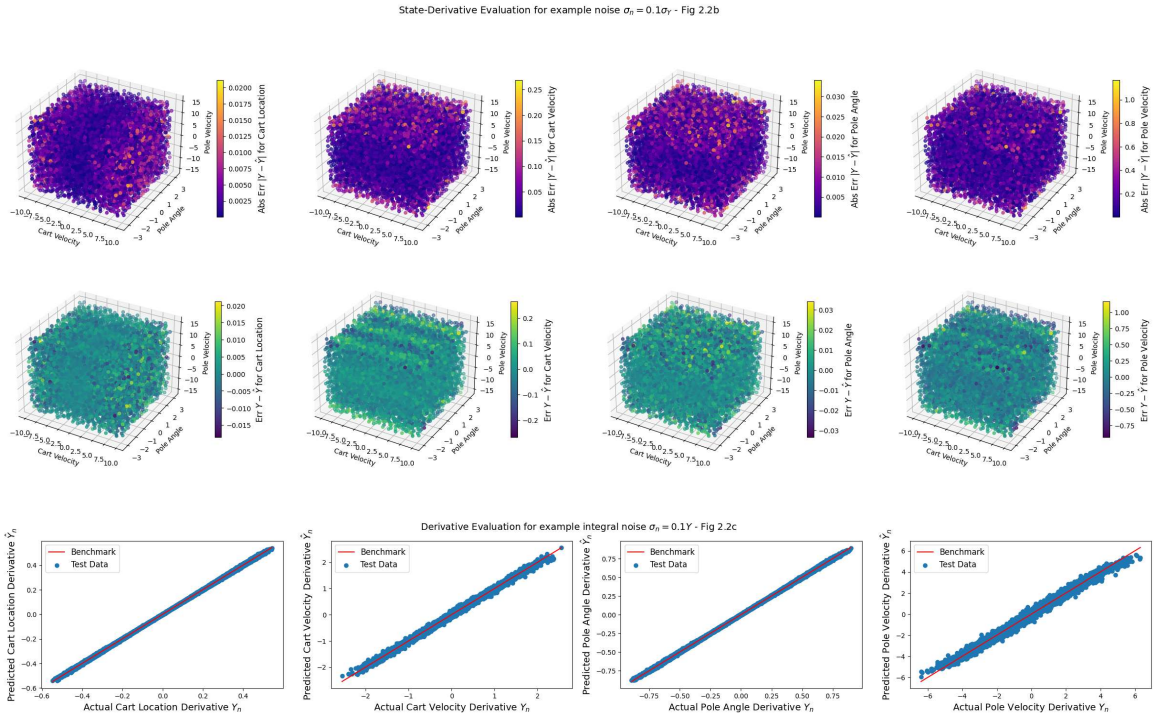


Fig 2.2a shows that the non-linear model with integration noise fits much better than with the observed noise **Fig 1.2a** (given by the scale). This is surprising since not only are we not using standard deviations of Y (and instead using Y) but we also perform integration. The errors still grow approximately exponentially with linearly increasing noise. However we should note that the standard deviations of Y are larger than Y and hence the noise in the observed case must be greater than that of the integral-case. The general relationship of the error remains approximately the same.

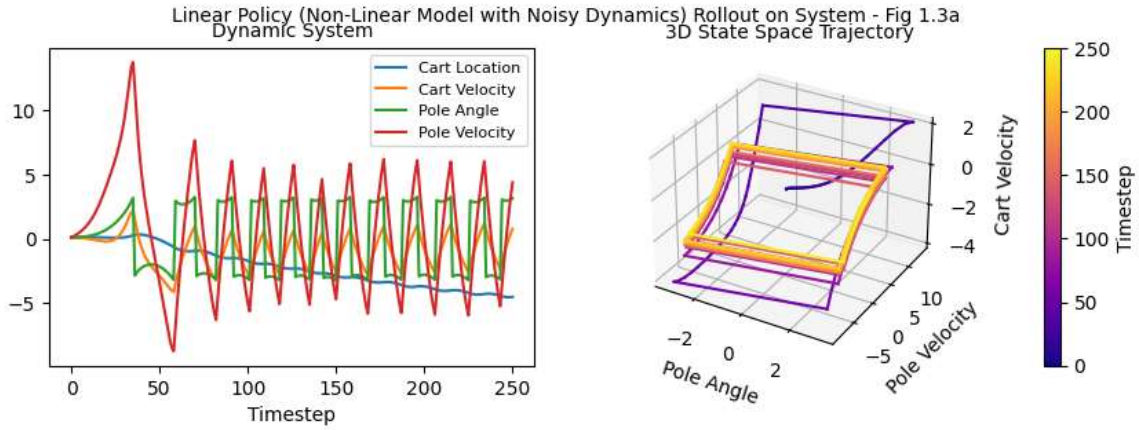


Comparing the non-linear model with integral noise **Fig 2.2c** with the non-linear model with observation noise **Fig 1.2c**, we can see that the cart location and pole angle fit better for the integral noise. This is partially due to overfitting on the noise but if we look at **Fig 2.2b (2)** for the cart location and the pole angle respectively, we see that the relative error which formed bands have been diminished by the noise and hence this justifies the better fitting. The cart velocity is slightly overfit too while the pole velocity is comparably similar to that of the observation noise case.

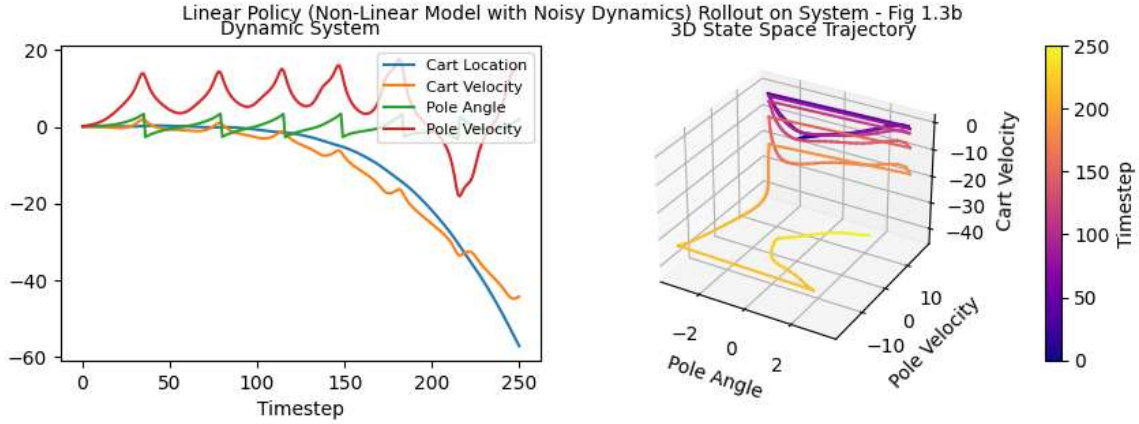
We can again see the extreme planes in **Fig 2.2b (1)** and notice that the cart location performs poorly when then the cart velocity is high, the pole angle performs poorly when then the cart velocity is high and more importantly the cart velocity performs poorly when then the pole velocity is high. The effect of the velocities on each other have been shown to be important in the contour plots in both **Section 1** and **Section 2**.

3.2.3 - Linear Policy using Non-Linear Model with Integral Noise

```
Optimal Policy Vector (Unaffected by Cart Location) [ 0.         -1.444331
92  4.87385943 -0.80146343]
```




```
Optimal Policy Vector  [ 0.757515    0.94595696  0.413669   -0.11438227]
```



The linear policy is simply not adequate enough for controlling on an uncertain model with noise with either observed noise or integral-noise and especially integral-noise. If we compare this with the unsuccessful linear policy designed on the non-linear model for the observation-noise case which simply oscillates for the policy independent of the cart location, we can see that the divergence in cart location and velocity means that the model had overfit to the noisy dataset.

4 - Non-Linear Control

We define a *nonlinear* policy, using a similar construction that we used for modelling the system, but this time we are modelling the `force` action as a function of the system state variables (i.e. the policy). The policy function is thus

$$p(X) = \sum_i w_i e^{-0.5(X-X_i)^T W (X-X_i)}$$

where the locations X_i, the weights w_i and the elements of the 4×4 symmetrix matrix W are free parameters to be optimised. Use between 5-20 basis functions. We optimise the parameters to try to obtain a policy that can keep the pole upright starting from the stable equilibrium (down) position.

We specifically consider the noise-free case as even in the noise-free case, this is a difficult task. Initial conditions for the loss function have been chosen reasonably such as to allow the pole to go from the stable equilibrium to the unstable equilibrium as best as possible in our search space (rather than our problem space).

4.1 - Policy Optimisation

We can now programmatically construct a policy vector again that performs the required for the given parameters. In this non-linaer control case, we have the following parameters:

$$W = \begin{bmatrix} a & b & c & d \\ b & e & f & g \\ c & f & h & i \\ d & g & i & j \end{bmatrix}$$

$\{w_i\}_{i=1}^B$ where B is the number of basis states.

Therefore we have a total of $(B + 10)$ policy parameters. Our policy vector is hence defined as $\underline{p} = [w_1, \ldots, w_B, a, \ldots, j]$ and we generate the required weight vector and matrix from this policy vector. This problem is a difficult task for optimisation as many local minima will exist and a clear termination condition for optimisation may not exist.

Optimsation Methods

The Nelder-Mead local optimsation algorithm is very useful for optimisation without using gradients however it does not peform well for multimodal optimisation problems (functions with multiple local optima) and even more so on noisy functions. Therefore unlike linear policy where we used local optimisation methods, here we will consider use basin-hopping as a method of global optimisation specifically intending to use a form of the Metropolis-Hastings algorithm alongside a local optimisation algorithm. The choice of the local optimisation algorithm is left as a hyperparameter for us to explore and Nelder-Mead may or may not be suitable. Using only a local optimisation method does not work as well as tested by trial-and-error. Note that our global optimisation method does not guarantee only a global minimum but atleast offers a greater degree of search space.

We also considered alternative global optimisation algorithms that were available in Scipy such as differential evolution (an example of an evolutionary algorithm) and dual annealing (a form of simulated annealing). These generally did not perform well and were too slow for use and required parameter bounds which could be specified arbitrarily but generally unknown if a global optimum would exist in these bounds.

Cost Functions

We also consider the use of alternate cost functions attempting to create a policy-loss function that adequately resembles the solution required. This is very difficult and the thought process to generate such a policy-loss function was to consider the following:

Our primary goal is to keep the pole angle at 0 and make sure the pole velocity is 0. If the cart-velocity is non-zero, then the pole-velocity must be non-zero as no such situation arises wherein the pole does not rotate when the cart is moving and the pole angle is 0. So a condition for cart velocity is not strictly necessary in our cost function. If we now consider that we do not care about the cart location. We come up with a cost function dependent only on the pole angle and pole velocity. Ignoring cart velocity is not most suitable so we can simply reduce the effect of the cartpole velocity. This is only an example of how a cost function can be constructed.

A custom cost function dependent on the 4 state variables cl, cv, pa, pv considered during our development is the following: $L(cl, cv, pa, pv) = 0.1(|cv| + 4(pa)^6 + pv^2)$ This performs comparably well to the default cost function however the unnormalised cost meant that optimisation was very slow due to the high non-linearity and large loss magnitudes in the search space. Hence the default cost function mentioned $L(X) = 1 - e^{-|X-X_0|^2/2\sigma_l^2}$ in **Section 2** was continued to be used instead.

Note: Here we define the loss function for a specific state as the cost function in order to clearly distinguish between the policy-loss function (the use of rollouts to compute the mean total loss) and the loss function.

Policy Basis Vectors

To obtain a good policy, it is crucial to have an appropriate set of basis vectors. If we consider the set of basis vectors as checkpoints the controller must achieve with the weights being the importance/weighting on the loss function relative to that basis state, then we need the pole angles to lie on the arc in a symmetric way and we could consider symmetrical weights although this may not be a good constraint for such a non-linear problem. Selecting cart velocity and pole velocity was much harder since if the velocities are too large, the errors would be highly velocity dependent rather than the pole angle and this would not necessarily be appropriate for solving the problem. If the velocity was zero, then a counterexample to show why this would be bad is that the pole velocity must be non-zero if we need to lift it up to the unstable equilibrium and hence penalising the velocity too much does not help to solve our problem.

Parameter Constraints

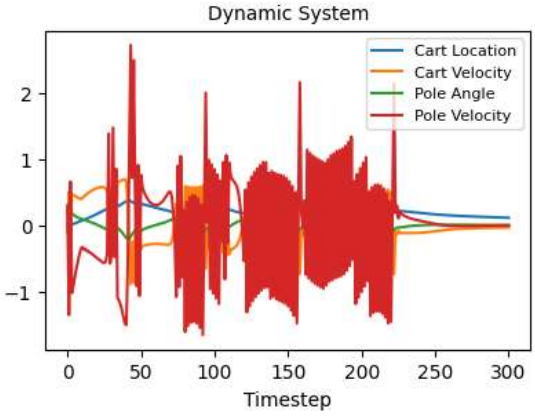
Consider the parameters specifically affecting the cart location (ie. the parameters a,b,c and d) to be 0 did not benefit the non-linear policy unlike the great result in the case of the linear policy. Forcing certain weights to be set constant did not particularly help although a better setting may have been possible.

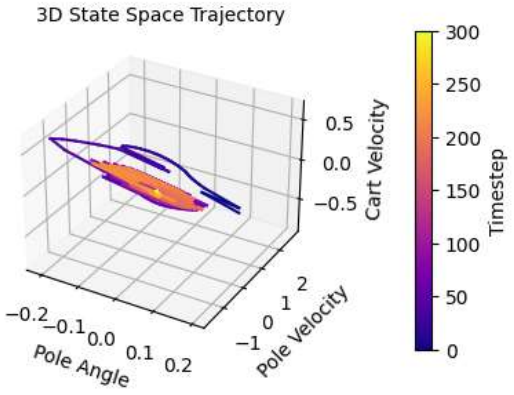
It is also ideally required that the weights matrix is positive definite and this could be done by regularisation but it was unclear as to what value the regularisation parameter should take hence this was omitted after development.

4.2 - Non-Linear Policy

To obtain a good non-linear policy, we considered the aspects mentioned above relevant to optimisation. Next used a policy-loss function that used the mean loss from rollouts to compute the loss of the policy vector. Starting from a small deviation from the stable equilibrium did not work well and hence was reverted to using a small deviation from the unstable equilibrium.

To optimise the policy we then did preliminary runs from the all-ones state and settled to a more optimal policy initialisation. After repeating this process a couple of times, the basin-hopping algorithm was able to find an optimal policy vector that could stabilise the pole although starting from the unstable equilibrium. Upon rollout testing, this same policy could not successfully stabilise the pole from the unstable equilibrium or even for a large deviation from the unstable equilibrium.





Although we were unable to obtain a non-linear policy that stabilises the pole from the stable equilibrium position, we were able to obtain a policy that was able to stabilise the pole starting with a similar deviation from the unstable equilibrium to the linear policy although the actual stabilisation took longer. However it was interesting to see that for most initial conditions, a point exists where both the pole angle and pole velocity are approximately 0 but the cart velocity and cart location are not. The policy basis states hence perform a very good job and does help the convergence of the optimisation algorithm albeit by basin-hopping.

The non-linear policy only works for certain initial conditions specifically the ones that are in the timeseries (although it is possible more such conditions exist). The example non-linear control rollout itself is very non-linear meaning that if our initial condition were to lie on any of the states that coincide with the rollout, then the non-linear policy would work. Such a state can be selected from the timeseries that would converge to the unstable equilibrium but would not work if it was too large a deviation from that state.

In the example rollout, we see that the pole angle only remains a maximum deviation of approximately 0.2 and hence this policy does not work for a deviation from the unstable equilibrium $\theta = \pi$, confirmed by creating a test rollout.

5 - Conclusion

In this project, we explored the cartpole problem with the end goal to develop an optimal controller. By primarily attempting to build a model, we not only are able to build on the model further but we also understand where the model is most likely to go wrong. We considered a linear model and a non-linear model knowing that the linear model would not perform as well but by doing so allowed to see where it performs poorly and why. Creating a non-linear model was not a difficult task but constructing one that is accurate for all initial conditions is very difficult. We built a model that worked consistently but not definitely. Such a model was not very suitable for control but by investigating linear policy iteration on the non-linear model as well as the system, we were able to compare and contrast the capability of the linear policy on both the model and the system. We also found that the performance of the linear policy controller was independent of the amount of data added and primarily dependent on the clarity and accuracy of the non-linear model and although we were able to accomplish a linear controller using the system itself, we were unable to develop a policy using the non-linear model.

We then considered the sensitivity and stability of the models and the linear policy in the presence of noise and found that neither models were robust but the non-linear model could be made robust with the inclusion of more training data. The linear policy simply could not find an adequate solution. By understanding the effect of the noise relative to the model fitting, we find that the robustness of the non-linear model is limited and that a linear policy is insufficient. Hence we then focused our objectives on developing a non-linear controller with the aim to stabilise from the stable equilibrium. This was unfortunately not achieved but a non-linear model that allowed a range of initial conditions deviating from the unstable equilibrium was successfully found by the process of global optimisation, basis tuning and hyperparameter changes.