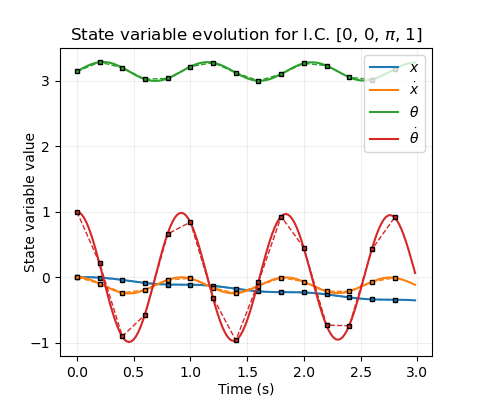
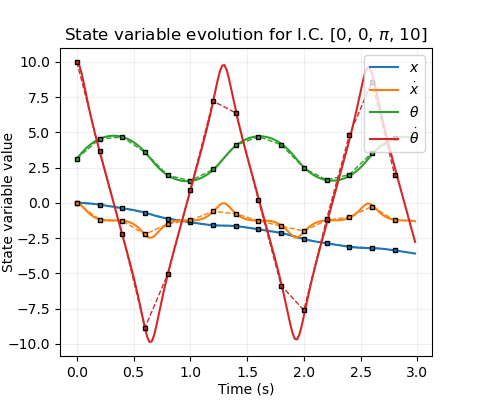
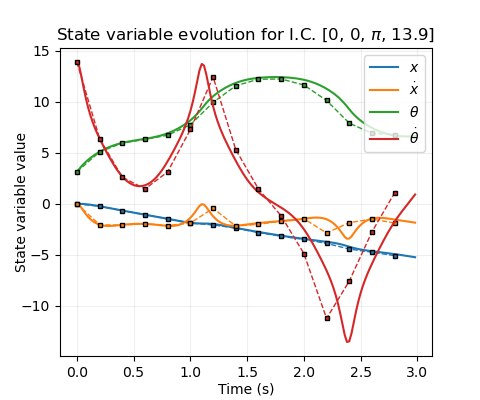
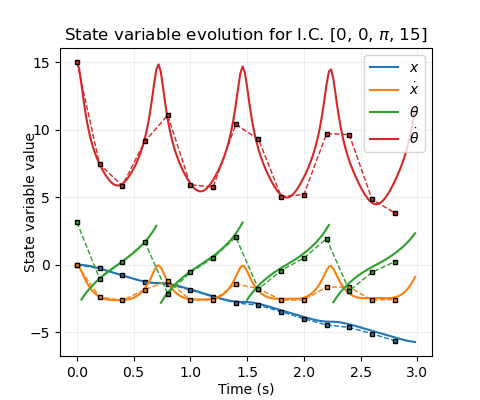
# Introduction

The trajectories and time evolution of a cartpole system are characterised and then modelled linearly. Predictions are then compared to reality.

# General considerations

The model runs 50 Euler steps over a time step of 0.2 seconds. The system has oscillating variables, and so the Nyquist criterion should be considered: if the system is observed at less than half the sampling rate of its maximum oscillation frequency, aliasing artefacts will occur. Such trajectories would be sharply nonlinear, and so would be difficult to predict with any model, linear or nonlinear. Since the sampling rate is fixed, it is therefore best to stick to low frequencies, and assume that useful controllers will keep the system state within this domain.

For the pole angular velocity:



**Fig 1.** Time evolution of system variables with no applied force, starting at the stable equilibrium (, with varying angular velocities. **Solid lines**: Illustrative evolution using a timestep of 0.02 s. **Dashed lines & markers**: Sampled values with the actual timestep (0.2 s). Note that the nature of the Euler scheme causes these pairs to diverge when friction is significant.

So should be satisfactory. Similarly, will be taken. is an angle (centred at the unstable equilibrium) and so . The system dynamics are invariant to shifts in , so can be set initially to zero for all trajectories.

# Rollouts

## Time-trajectories

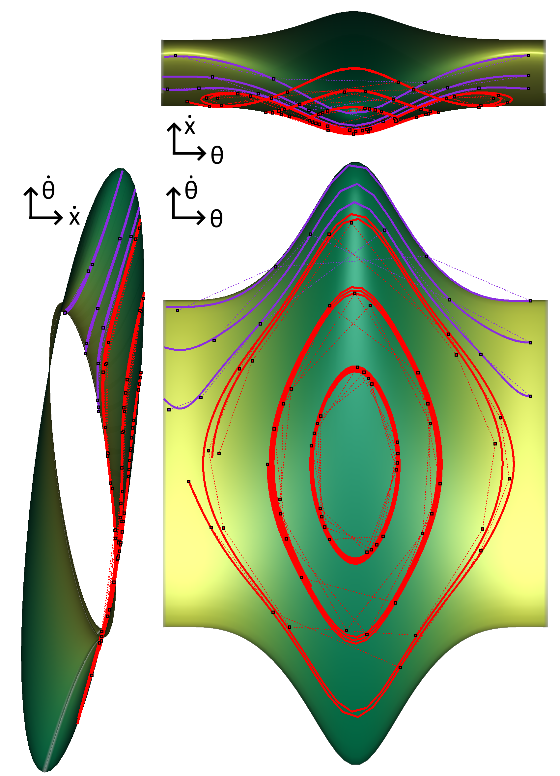
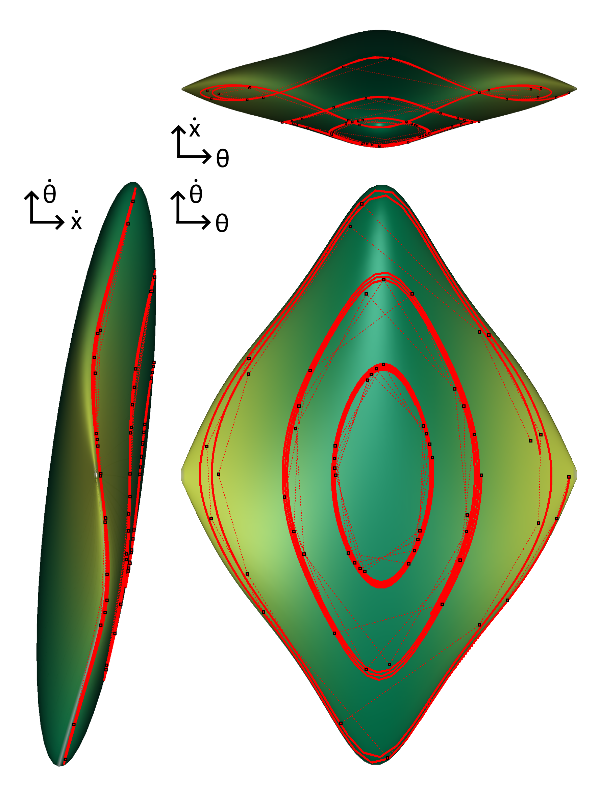
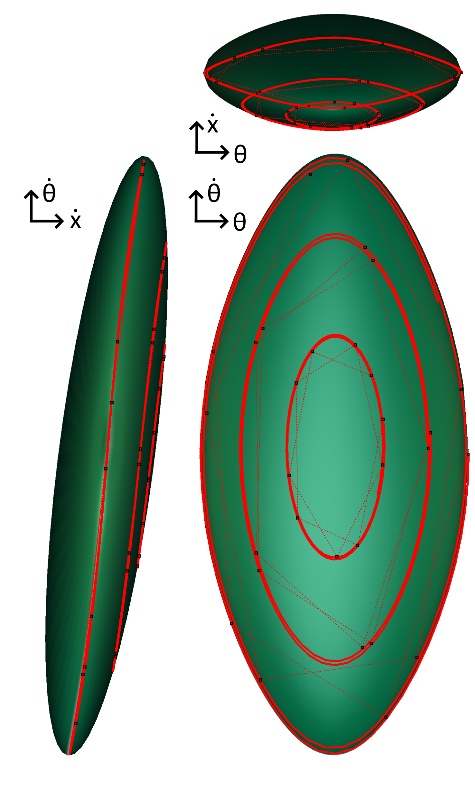
There are two principal trajectories of the system: *bound*and *unbound*. Bound trajectories occur when the pole oscillates with insufficient energy to reach . In the bound, low-energy limit (fig 1a), the pole oscillates sinusoidally about . The cart’s velocity couples to this, oscillating in phase with the but with a much smaller amplitude. However, it remains of an opposite sign to the initial angular velocity, so that the cart position “drifts” monotonically. Conversely, unbound trajectories have sufficiently high energy that the pole repeatedly “flips”, causing to monotonically increase (sans remapping) (fig 1d). Between these extremes, the behaviour stays broadly similar either side of the critical transition energy (, defining zero potential at [A1]), and the monotonic cart “drifting” remains, though the waveforms change shape in a nontrivial way (fig 1b). In all cases, friction gradually causes the energy to diminish and the oscillation amplitudes to reduce, so that bound states decay to zero and unbound states eventually become bound (fig 1c). This effect of friction can be largely decoupled from the broader system dynamics, especially when a controller is present to offset the energy loss.

Adding a nonzero cart velocity leaves the pole dynamics mostly unchanged (see next section), and shifts in initial cart position are trivial.

## Phase portraits

Since the cart position is trivial to deduce given its velocity history, and the timescale of energy loss due to friction is much greater than the timescale of the cart dynamics, the a qualitative understanding of the state-space trajectories of this system can be gained entirely via energy embedded in the 3d space formed by [.

**Fig 2.** Orthographic views of trajectories in [-space with initial energies equal to 1, 0, and -2 for a), b), c) respectively, superimposed onto energy isosurfaces at those values. **Red lines**: Bound trajectories. **Purple lines:** Unbound. **Solid lines:** Illustrative, with 0.02 s timestep. **Dotted lines & markers:** 0.2 s timestep.



**c)**

**a)**

**b)**

This consists of ellipses in the - plane modulated in size and tilt angle by the value of . When , a continuous, periodic tube is formed. It splits into disjoint surfaces at , which shrink into ellipsoids as reduces to its minimal value. Over longer time periods, trajectories migrate onto lower-energy isosurfaces until they become bound and oscillations decay.

# Changes of State

## Formulation

Suppose is the current state of the system. After one step of time evolution (one “action”), let be the next state, and define . The goal of modelling this system is to learn the mapping , or equivalently .

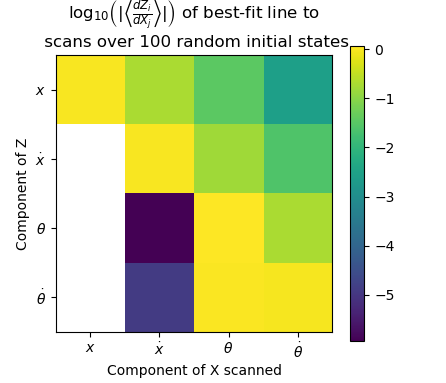
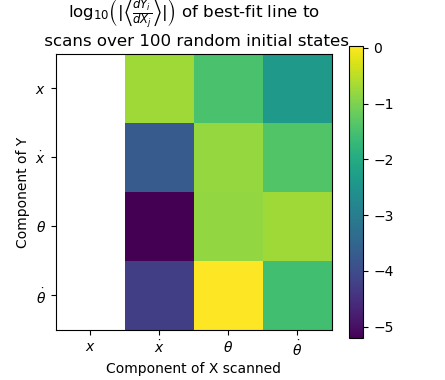
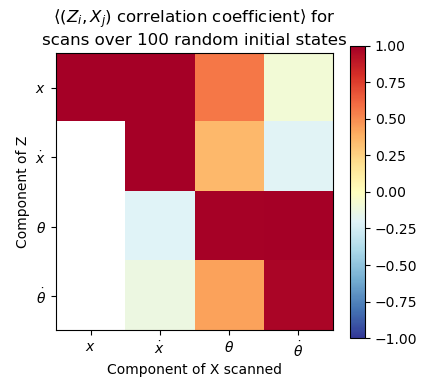
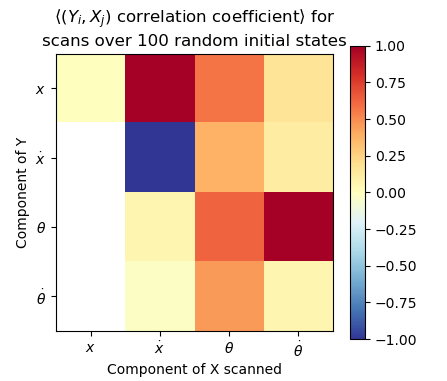
## General Properties of and

Over the relatively small timestep T=0.2 s, variables should generally evolve in a relatively simple way. In fact, many pairs of input-output variables have approximately linear relationships for . This is demonstrable by initialising the system in random states, scanning alternately over the full ranges of state variables, and calculating correlation coefficients with the output state variables. Figure 3a displays the mean of such values over 100 initial states – clearly a number of matrix elements imply linearity, at least for particular settings of other parameters. In particular, since each variable does not change much in a timestep, it is correlated strongly to itself, and the gradient of its best fit line is approximately equal to 1 (fig. 3b).

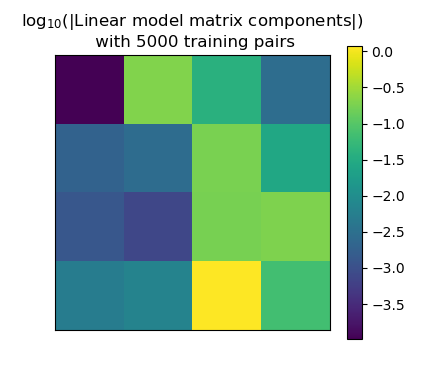
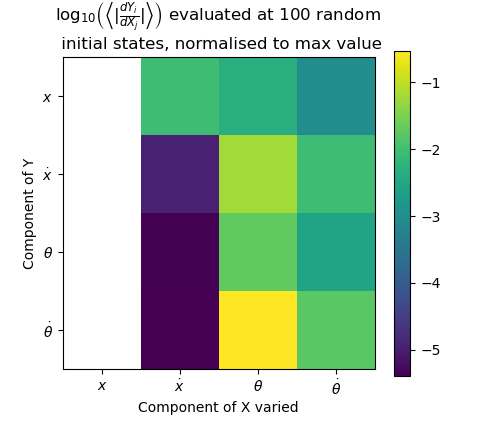
**c)**

**b)**

**a)**



**Fig 3.** Colour maps of: **a), b)** Mean Pearson correlation coefficient between change in function value and initial state scanned over full range for 100 random initial states for and ; **c), d)** Log magnitude of mean fitted gradient for corresponding scans; **e)** Components of the linear model matrix fit to 5000 random (Y, X) data pairs; **f)** Log mean magnitudes of gradient components evaluated at 100 random initial states. *White-coloured values are those with magnitudes < 1e-12, and so considered to be zero.*



Subtracting the initial state gives and removes these diagonal correlations – though the others remain (fig. 3c,d)– as well as removing constant terms. If we fit a linear model to , we can hope to capture these particular relationships, hoping that the gradients do not change significantly over the range of other parameter settings.

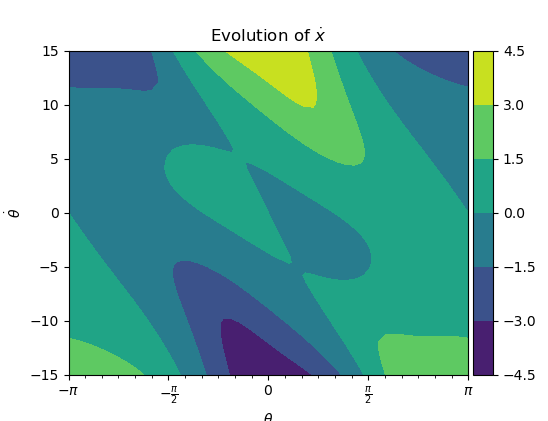
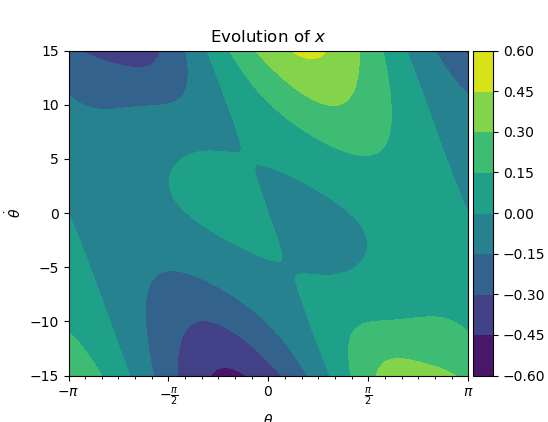
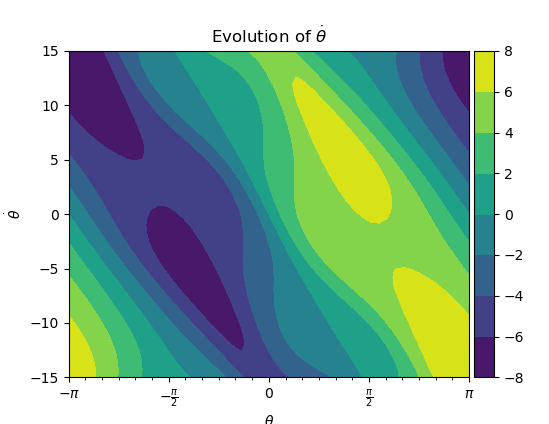
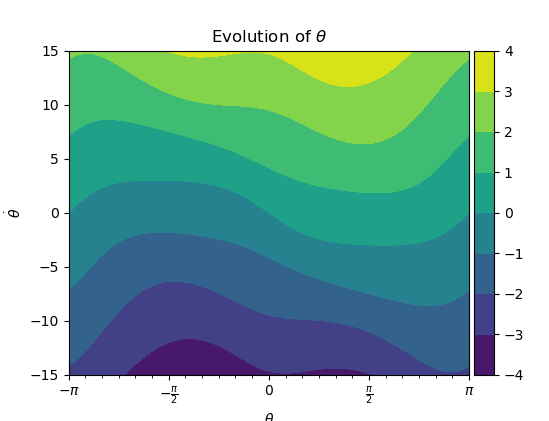
The nonlinear parameter dependencies of which have been obscured by looking for linear relationships can be approximately found by evaluating the gradients at random points, and summing their absolute values over many samples (fig 3e). With this in mind, we can entirely exclude dependence on , and establish that dependence on is indeed weak except of course for . However, a better picture of nonlinear mappings should be obtained first.

**f)**

**e)**

**d)**

## Understanding in Physical Terms

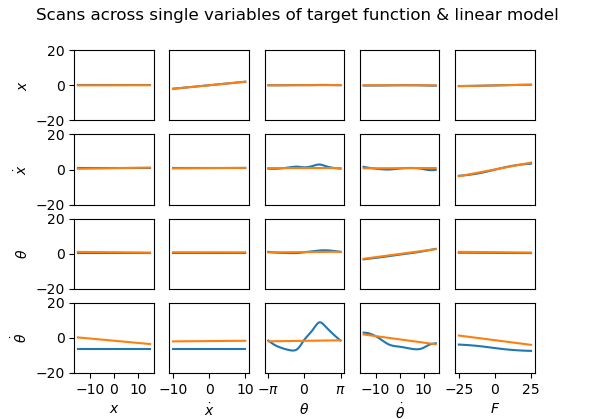


**Fig 5.** Contour plots of state variable evolutions as functions of and .

Each state variable’s evolution depends fairly strongly on and . Variables only have a very weak dependence on , except of course . The variables are all completely independent of .

These traits can be understood by considering the dynamical equations of the system [A2], where causal connections for an infinitesimal timestep are depicted schematically in fig. 4. There is a general flow of information downward through the figure, with only affecting the angular variables and itself through the weak effects of friction. During the 50 Euler iterations per timestep, the information flow is diffused through the network, resulting in the causality matrix of Figure 3e. In this sense, the angular variables completely drive the dynamics, and so it is important to take care to model them well.

## Nonlinear Properties of

Although friction is important on longer time scales, the goal is to actively control the system, and so the shorter-term dynamics are the only important aspects to model. Therefore all of the components of except the first can be characterised in a contour plot against and . For the evolution of cart position, the relationship with will be simply excluding friction, so with this in mind, a contour plot against and with is again sufficient for understanding. These plots are shown in fig. 5. It is clear that for these mappings, a linear function will be insufficient to approximate the entire landscape. For the evolution of , the broad behaviour will be correct (ignoring the “wiggles”), and for the evolution of , a linear model trained from data in the central diagonal region would roughly characterise the local contours. However, the full range of needs to be modelled, so a linear model cannot be expected to model the oscillatory relationship perpendicular to these contours well. For and the function is near zero in the majority of the landscape – which can be reproduced trivially with a linear model – but there are hills and basins at large , which definitely cannot. However, if is kept sufficiently small, an approximately zero function could work moderately well, excluding the shallower central wavelet-like variation which corresponds to the minor couple oscillations in fig. 1.

**Fig 5.** Contour plots of state variable evolutions as functions of and .

**Fig 4.** Lines represent the flow of information and causal changes between state variables over an infinitesimally short timestep. Solid lines indicate that the dependence is important on short timescales; dashed lines indicate that the only dependence is via the weak effects of friction.

# Linear model

## Formulation

For each component of , assume the dependence on is linear. As discussed, the constant term is then zero, and for each ,

Where the coefficients of are to be learned by least-squares minimisation over a large number of pairs. This gives an equation of the form where is a matrix of stacked vectors and is the corresponding values. Each such equation can then be optimised in a least-squares sense by standard algorithms and the resulting vectors stacked in into the matrix to give .

## Convergence

A good number of random samples to generate is (fig. 6). The magnitudes of elements of a matrix generated is shown in fig. 3e, and its elements broadly line up with expectations from fig. 3d. The nonzero coefficients of the cart position can be attributed to the fact the matrix has not fully converged.

## Single-step effectiveness of model

A model was trained with datapoints for this section [A3].

#### Expected vs target next step

To visualise how well the model matches the real system evolution over a single step, predictions can be plotted on the y-axis against the actual evolutions on the x-axis in pairs. The closer the points to the line y=x, the better the prediction. If an initial state is then scanned over, this forms a line which should ideally lie entirely near y=x. Such plots are shown in fig 7. They illustrate several features of the model:

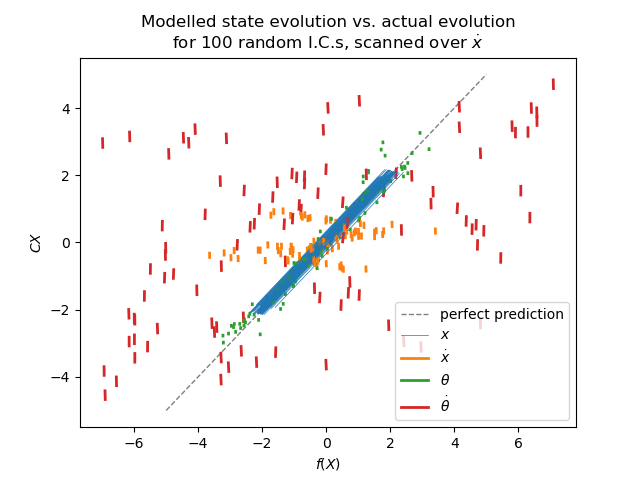
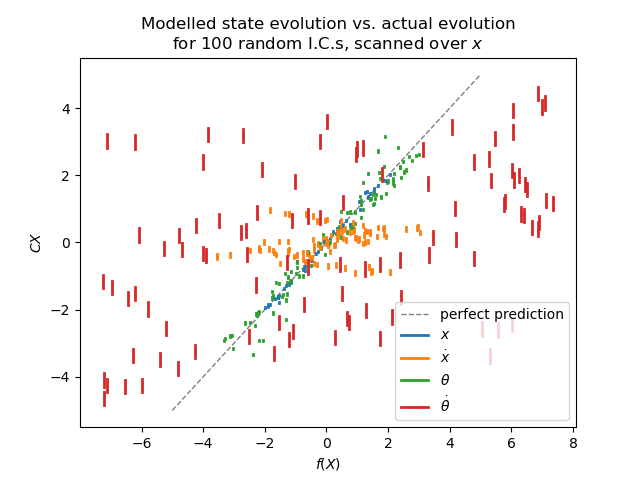
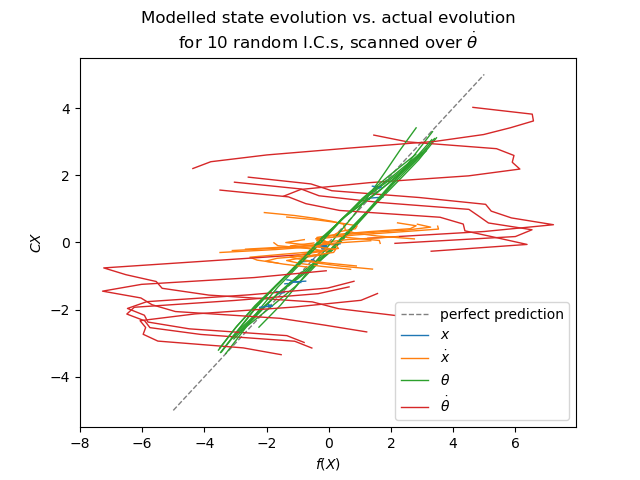
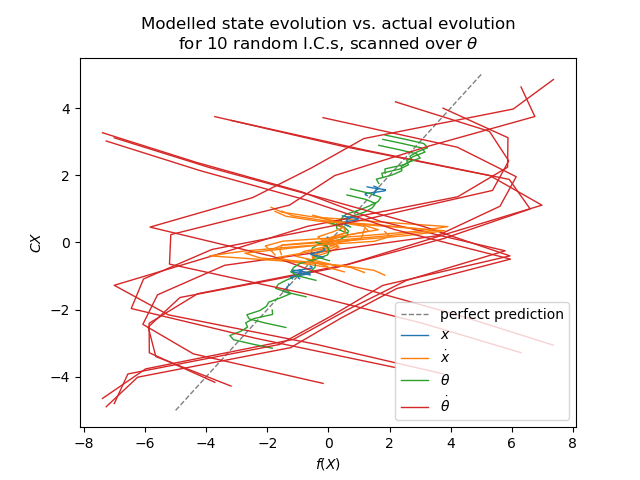
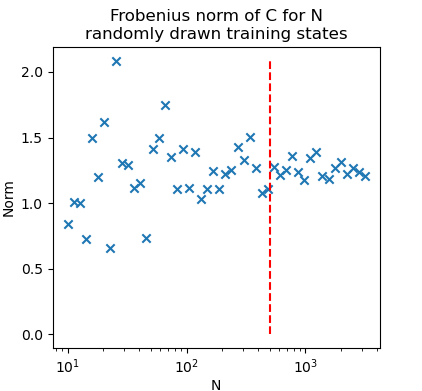
* Predictions of and are in general very good, while predictions of the velocities are much poorer.
* The model correctly demonstrates independence of and on and (hence their lines are points).
* The velocities are slightly more sensitive to and than the positions, but are still fairly independent.
* and correctly have a large effect on state variables, but their predictions are fairly poor.

#### Contour plots

The difference between the predicted and actual evolutions can be plotted as a contour map against and , showing how well the linear model fits to fig. 8. Predictions are poor in general, and get worse as and increase. Unfortunately, since the stable equilibrium is at , the system will always end up in this region of state space.

DO THESE PLOTS WITH RELATIVE

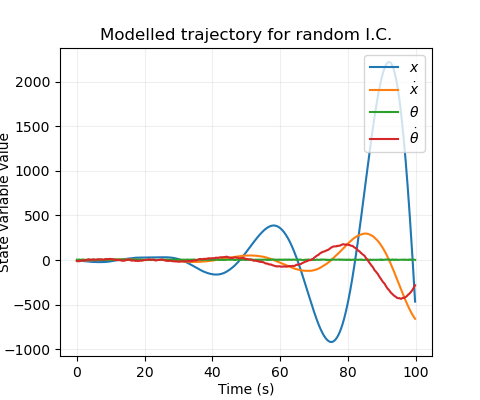
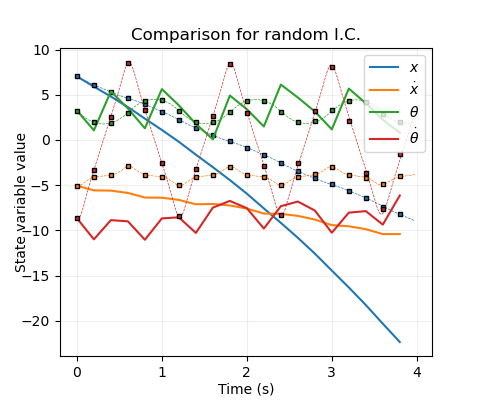
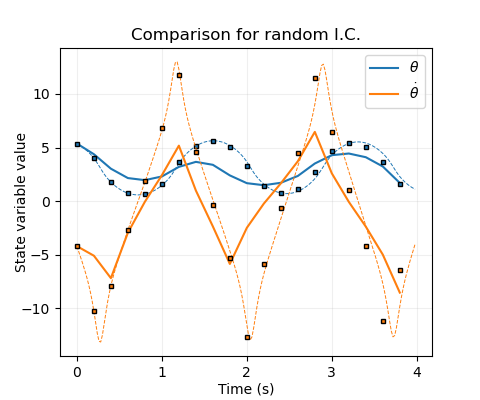
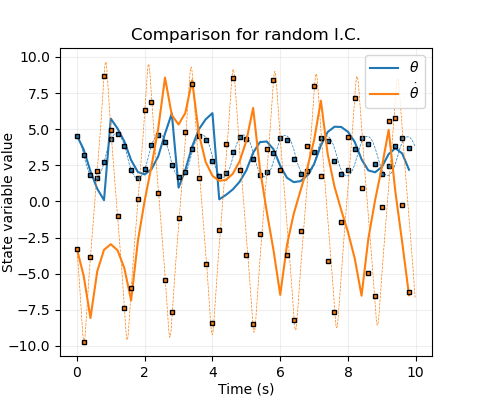
**Fig 6.** Frobenius norm of matrix used as a metric for convergence, plotted for collections of ( pairs of varying sizes (with randomly drawn within the variable ranges). represents a good tradeoff of optimisation speed versus accuracy. The model is expected to perform fairly poorly, so a little noise is not much of a problem.



**Fig 7.** Comparison of state evolutions, initialised with random states, scanned over state variables.

## Trajectories of model

In general, fitted models perform poorly. The state variables diverge from real trajectories within a few timesteps and sometimes immediately. The state variables (excluding the angle) eventually diverge to infinity, driven by the cart velocity, rather than decaying to zero, although this often takes a long time.



**Fig 5.** Contour plots of state variable evolutions as functions of and .

There are some situations when the model performs better. Some matrices, including the previously fitted one with reproduce simple oscillations qualitatively, even if they rapidly become out of phase and often have different frequencies. One particular pathology, which manifests even more strongly if the time step is decreased so that more evolutions are required per cycle (despite the simpler modelling task for a single evolution), is that the model unpredictably alternates between the bound and unbound states, ending up eventually in the bound state, in contradiction to reality.

## Conclusions

A linear model is insufficient to explain the evolution of the cartpole system, due to periodicity-induced nonlinearities in the angular variables which drive the system. A nonlinear model would be required to fit the evolution function better.

# Nonlinear Fit

## Outline

To fit a good regression model to the update function, nonlinear kernel functions are required. For simplicity, diagonal-covariance Gaussians were chosen. Using a subset of training datapoints as kernel centres makes this correspond to a sparse Gaussian process. The solution can be regularised with hyperparameter to aid numerical stability.

Since Gaussians are better for fitting mean-zero fluctuations than linear trends, it’s best to fit the nonlinear model to the residuals of the linear model. (Note: This requires enforcing zero -dependence in the matrix to prevent periodicity artefacts.)

and , the number of training data points and the number of basis function centres, can be separately varied to give a trade-off of model complexity vs. fit, and the six hyperparameters , , , , can be optimised for given data.

## Hyperparameter Optimisation

#### Choosing initial values

First , data points were obtained with a Sobol sequence over the fitting region (same bounds as before).

From the plots of (.), rough length scale variations can be selected as where the first two are already well fit by the linear model so should vary very slowly, if at all. was selected as - it will worsen the fit when it is large by over-smoothing and smaller should be better (for the optimal solution) as long as the inversion is robust. Fit can be evaluated by generating new random data points (not from the Sobol sequence to prevent systematic biases), and calculating root-mean-squared error. Scans from these initial guesses are shown in Fig. 10 (a, b).

#### Numerical optimisation

Starting with the initial guess, the Nelder-Mead algorithm was employed to (ideally) search for a good minimum. The results were:

The fact that there is an optimal setting as above indicates that there are insufficient kernel function centres, so some smoothing is helping the fit. However, these hyperparameters should be good enough to assess fit convergence vs amount of data.

## Fit vs amount of training data

Error should decrease with amount of data, and this is verified. The convergence of the model as M and N increase is shown in fig. 10 (c, d). For fixed M, increasing N leads to a limiting error at N16M.

Fitting time and model evaluation time must be considered against fit accuracy. Since N only comes into training & model evaluation, and due to its lower time complexity (roughly, vs ), N can always be set to 16M (this will be assumed from here). M should then be chosen based on a tradeoff between fit quality and fitting/evaluation time, plotted in fig. 10 (e). M= is good for comparison tasks where the model must be repeatedly fit (~ms to fit); M= is better when a very good fit is required (~ms to evaluate). Re-optimising parameters for each M value, this time keeping the first two elements fixed at , improved fits further, and a set of optimised models were created and stored for M in powers of 2 from to . *(note: 2^13 used parameters from 2^12 because optimisation would have taken days).*

## Assessing single-step fit.

See scatter plots and 2d scans. (figs. 11 and 12).

## Assessing rollout agreement

The nonlinear functions modelled rollouts extremely well in general – impressively so, considering the relative simplicity of the modelling process. Figure 13 shows some illustrative trajectories and typical agreements & disagreements.

#### Mismatch criterion

To quantitatively define agreement, a mathematical statement on at which point a rollout diverges was required. Several were experimented with, and the most reasonable was chosen as follows:

* Look backward for 10 iterations. Find the oscillation ranges of and over that window (for each of the first 10 states, the first 10-step window was used).
* Divergence occurs when the error in either exceeds a fraction of its range, or when the error in exceeds . (15 being half the modelling range).

The fraction was set at default to 0.2 (20%). was ignored because the dynamics of and are so strongly coupled. In practice, the criterion was never violated – errors in the other variables preceded this. The criterion is demonstrated in figs. 13 and 14.

#### Model performance

To evaluate a model’s overall fit, iterations until mismatch were calculated over random ICs (capping energy to remain inside the modelling region). Histograms are shown in Fig. 14a, b. These could be combined into an overall score in a number of ways:

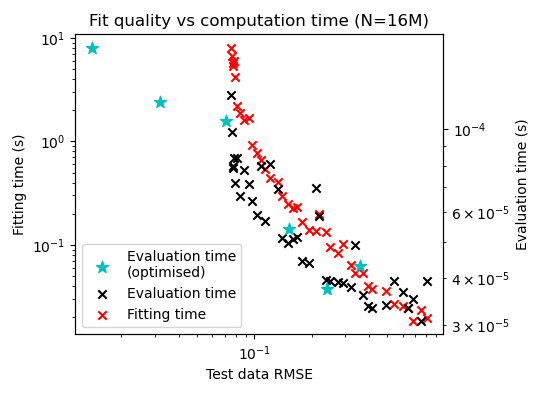
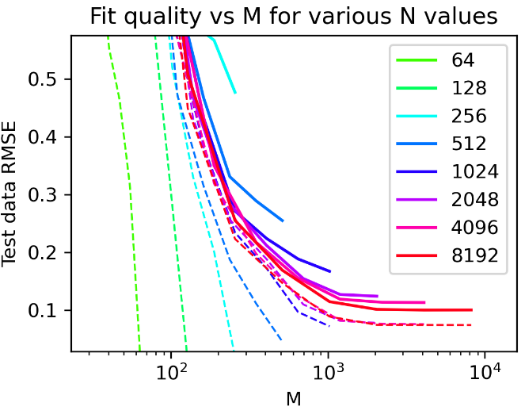
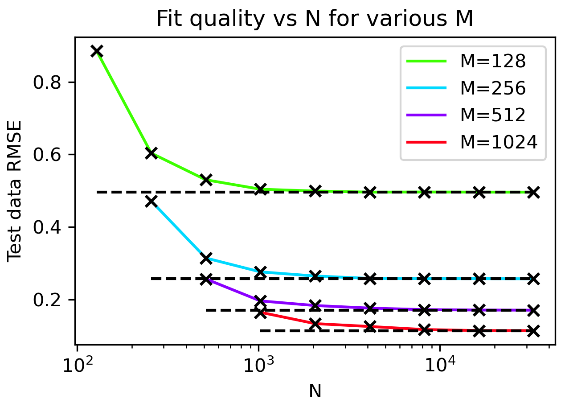
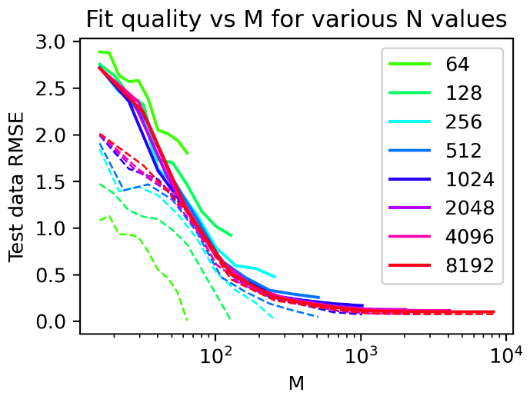
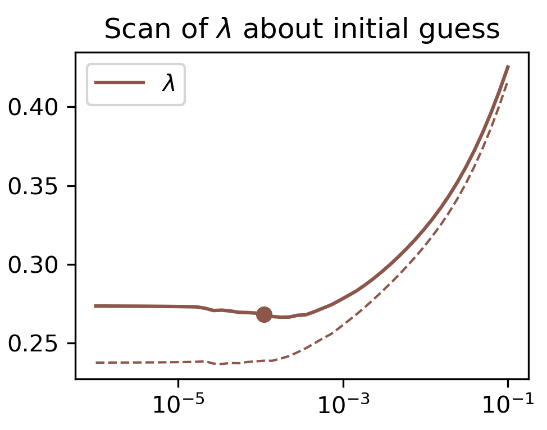
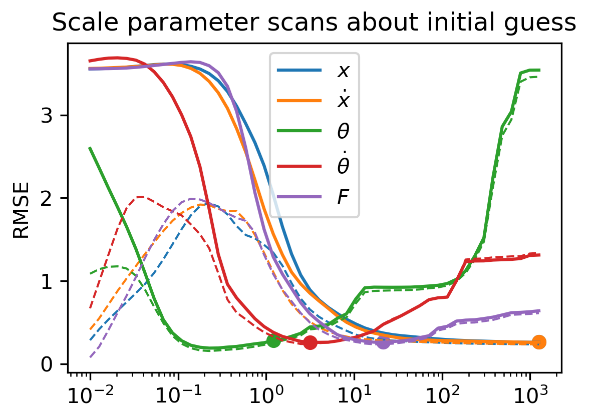
1. Mean number of matching iterations
2. Mean number of matching oscillations
3. Modal number of matching iterations
4. Minimum number of iterations for which 90% of rollouts still match

(1) is problematic because some trajectories match for a long time. Simulating until divergence would be computationally costly, and terminating at a maximum iteration will bias the mean down for these cases. (2) has the same issue. (3) is avoids the mean bias caused by termination, which is attractive. (*2) was implemented in the code as required but the plot wasn’t considered illustrative*

However, (4) is the most relevant measure to the task of control. Ideally, the system will be stabilised within a few cycles, and we need a good and *reliable* match in this time period – it doesn’t matter if the average matching time is 1000 iterations if two thirds of cases diverge after 3. This score provides the added benefits of requiring much less computation (since there is no need to simulate for long time window to see what happens to well-agreeing trajectories) and being insensitive to the precise value of (since most trajectories diverging early diverge by a lot – fig. 13b, 14a). The threshold of 90% was chosen to give a reasonable output spread for model comparison.

From figs. 14c and 10f, was chosen as a reasonable tradeoff of evaluation time (~0.1ms) and score (~6.5 iterations until 10% diverge) for control. For reference, the actual system dynamics function as provided took ~.7ms to evaluate, and a C-accelerated version that was written to aid computation took ~0.01ms.

**Fig 10. a), b):** Hyperparameter scans about initial guess (. **c):** Lines with fixed training in powers of two, varying in powers of two up to . **d):** as c) but zoomed in. **Dotted lines (a-d)**: test RMSE; **solid lines** **(a-d)**: training RMSE. **e):** Lines with fixed , varying in powers of two until asymptote. **f):** Scatter plot of fitting and evaluation times against test RMSE, up to M=2048. Re-optimising parameters for each M value helps break the asymptote, resulting in the trend line of blue stars. *In all cases, .*



**a)**

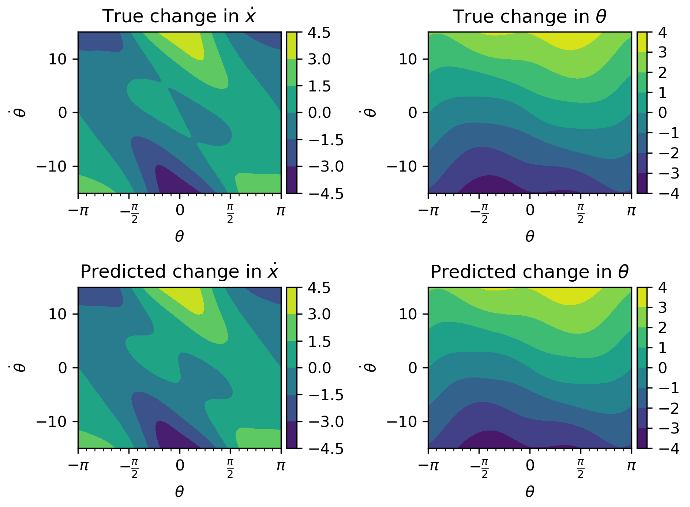
**b)**

**c)**

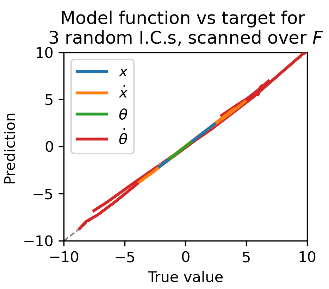
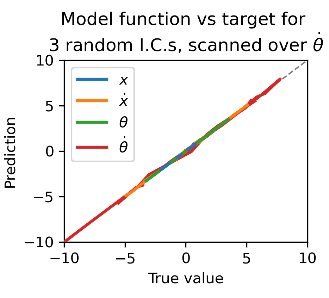
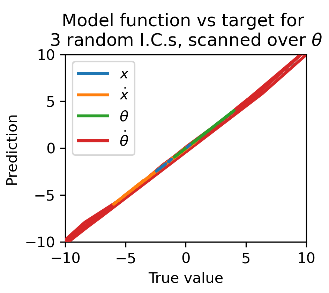
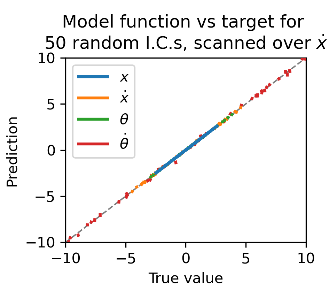
**d)**

**e)**

**f)**



**Fig 12.** 2d scans demonstrating fit of M= model (as required)

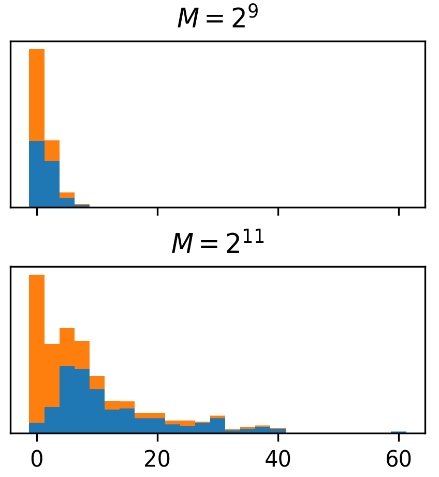
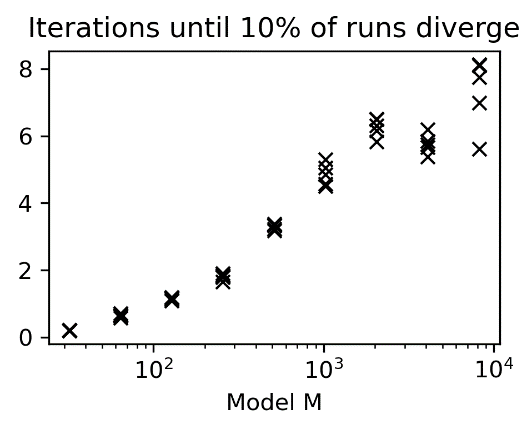
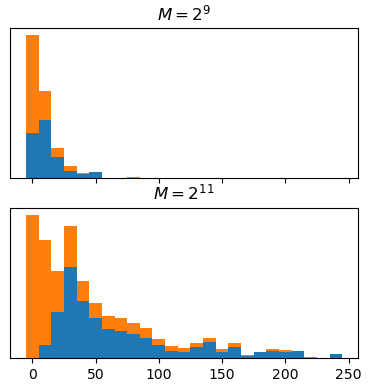


**Fig 11.** Scatterplots demonstrating fit of M= model (as required)

**Fig 12.** 2d scans demonstrating fit of M= model.

**Fig 14. a)** Runs until divergence, , for models with and kernel functions. *Orange:* unbound ICs, exponential-like distribution. *Blue:* bound ICs, peaked distribution. These different shapes are due to the fact that the transition from unbound to bound is the most error-prone. **b)** Same plots, but quantified in number of oscillations *(as required)*. Unbound cycles were considered half-oscillations. **c)** Model matching score for optimised models with M increasing in powers of 2, each run 5 times for 500 trials. Note that one oscillation is approximately 8 iterations.

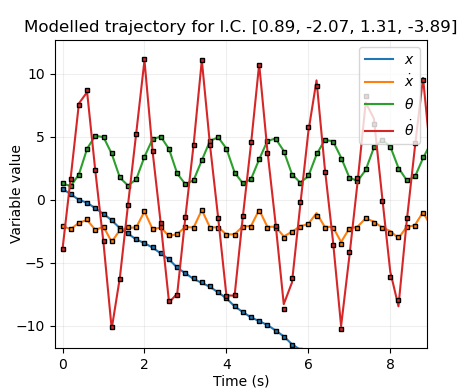
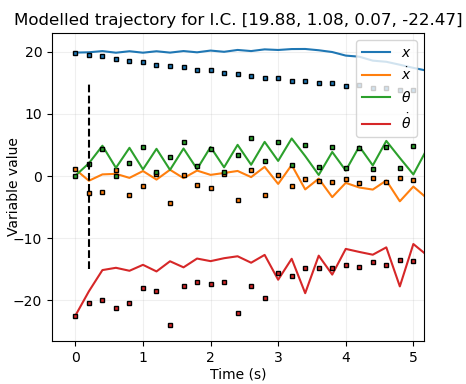
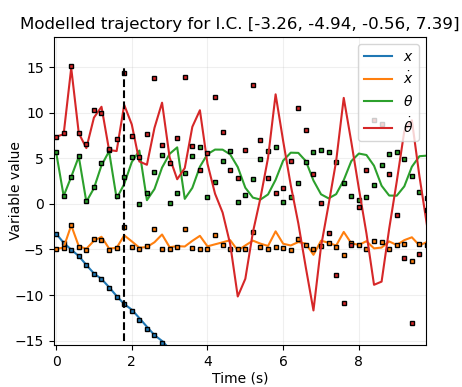
**c)**



**b)**

**a)**

**Fig 13.** Rollouts with illustrative initial conditions, using a model with M=2048. **a)** Bound states were generally modelled extremely well, matching for hundreds of timesteps (the precise mismatch point is then irrelevant for the purposes of control). **b)** Early divergence was most common around decay from unbound to bound states. **c)** The mismatch threshold was set relatively strictly, so that some trajectories remaining qualitatively similar were considered to diverge. **d)** Energies were generally constrained to ensure states remained in the modelled region. When outside, the model performed predictably poorly. *Black dashed line*: Step at which trajectories were considered to divergence; mismatch threshold set at 20%.



**a)**

**b)**

**c)**

**d)**

# Linear Control

## Outline

Adding in force: note that this has been done from the start in this report

#### Problem formulation

The simplest way to control the cart is with a force that is a linear function of the state variables:

To find an effective policy, a score must be given to each set of policy parameters based on how well it stabilises the cart. This is defined as the loss function

With scale parameters for each state vector component, a window of iterations, and target state . Although a linear controller is not expected to stabilise initial states far from the equilibrium, there is no harm in also introducing periodicity to the loss function in the same way as for the nonlinear model.

The policy should ideally be:

1. **Robust:** stabilise over a wide range of initial states.
2. **Fast:** stabilise reasonably quickly.
3. **Bounded:** Avoid visiting extreme states, i.e. states outside the modelling range.

The loss of a *policy* should then be defined as evaluated over a relatively small number of iterations (to ensure goal 2), evaluated as an average value over a number of random perturbed initial states (to ensure goal 1). If the trajectory diverges outside of the modelling range, the loss should return its maximum value of 1 (satisfying goal 3). Since restricting the time allowed for convergence reduces robustness, and considering the matching iteration numbers in Fig. 14c, was set to 20.

#### Loss discount

This formulation of loss function punishes certain types of policy – for example, if the pole starts with and , it may be desirable to let the pole swing around a full revolution before stabilising, but this will pass through a high-loss region. One way of allowing this type of policy is to multiply cumulative loss by a discount factor every timestep, so that early contributions are reduced.

However, this will not be used for the linear controller, on the assumption that it is not complex enough to create this kind of policy while still being good in other regions of parameter space.

#### Strategy

The global minimum basin in -space giving good policies is likely to be very small, possibly hard to find, and one of multiple local minima, even when loss is evaluated using the true system dynamics. When using the modelled dynamics, whose deviations from the true evolution are nontrivial, the loss landscape will be more complex and the global minimum may not coincide with the previous. These problems will reduce in severity as the fit converges (with high numbers of basis functions).

In a real-world situation, the goal is then to find a fit good enough that it can be used to obtain a robust linear policy, while having few enough basis functions that evaluating the function during optimisation is computationally feasible. Here, this will be explored by:

1. Optimising using the true dynamics to find ideal policies
2. Optimising using modelled dynamics for various model sizes
3. Comparing quality and robustness of policies obtained using these models.

Different linear policies be effective starting from different regions of parameter space. Policies which start only at small deviations from the target state will be explored.

## True dynamics optimisation

#### Getting an initial good policy

This is still a hard, multimodal problem with a small region of good policies and large region of terrible policies. To find the former,

1. Loss scales should be as large as possible (to encourage a more gradually-sloping loss landscape, assisting with locating minima) while being small enough to punish bad policies. Initially, scales were chosen to have when each component is at its range limit and the others are zero (). However, policies derived from this tended to have and converge much more slowly than and . To punish this, was set to instead. *Note: the scales could have been explored in more detail with plots and optimisation but were not due to time constraints.*
2. The problem was split into easier problems, the solutions of which were nested supersets of the overall solution: the penalties for and were originally neglected, before being reintroduced (in that order).
3. IC ranges were initially set small, and manipulated as a fraction of the overall parameter ranges .

With , with losses neglected, there was a clear wide global minimum basin in the plane (fig. 17ai). As increased its ‘walls’ rapidly became steeper.

When the loss was introduced, there was still a clear dominant basin, as verified by scanning through one of the angle variables across the first basin while taking 2d contour plots of the other two (fig. 17aii, iii). A line was fitted through the deepest valley in the plane, loss was introduced, and was scanned through while making contour plots of the combined velocity variable and , revealing that a clear dominant basin once again remained (fig. 17aiv). The overall volume of parameter space which contained minimally useful policies was approximately

#### Testing robustness

A good policy should stabilise a wide range of ICs. To check the robustness of a policy, the following algorithm was used:

1. For each state variable , set the random IC range to zero for every other variable, then gradually increase between 0 and 1.
2. For each value, run 100 random IC tests, and calculate the *success rate*, evaluated by whether the run spends at least 10 iterations with mean state magnitudes less than , allowing a total of 50 iterations for this to happen. *(note: this cannot be itself used as a loss function, as there are a discrete number of runs, and it would be computationally infeasible to have enough that numerical optimisation is possible). – footnote?*
3. Record the first value at which the success rate is not 100%,
4. Once this is done for each state variable, carry out the same procedure, but using , a fractional value that multiplies the vector of parameter thresholds found in step 3.
5. The percentage of modelled state space which is stabilised is then approximately .

#### Finding robust policies

Two strategies were used to search for robust policies.

Firstly, the manually-located minimum gave a good starting point for optimisation within a simple basin, using the Nelder-Mead method. Noise due to random ICs hindered this by creating many tiny local minima, so the set random seed was used for each evaluation point in a given optimisation run, meaning that the same set of (50x) ICs were used over the course of that run (leading to a small potential bias at convergence).

Once the method converged to a minimum, slices were made about that point in all six planes to get a rough picture of the loss landscape around it (fig 17c). The robustness of this policy was probed via the above strategy.

It is possible that a model optimised on a wider IC range might be more robust to ICs outside of this training range. To test this, was increased by factors of , the loss function re-optimised (starting from the previous minimum), and these plots and robustness tests were repeated until (fig 18a).

Secondly, optimisation was attempted at the same set of values, but with random initial vectors where each element . 25 attempts were made for each , and the lowest loss function result taken.

#### Conclusions

The most robust policies had the optimisation set to approximately. However, these policies were very hard to find: The random initialisation optimisation attempts found policies that matched the performances in fig 18a with a probability of 1/6 for small , and runs which didn’t find these optima gave policies that performed terribly. Furthermore, this probability reduced as increased to the point where that no successful policies were located for any .

One way to subvert this problem was to set or smaller, perform optimisation with random ICs to find the minimum, and use this result as a starting point for optimisation on . Another way was to use the simplified, encompassing problems to search for a starting point.

## Model dynamics optimisation

#### Landscape visualisation

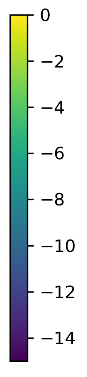
For the modelled dynamics, the loss landscape is likely more complicated and multimodal than for the true dynamics. However, plotting graphs for the simplified problem for as low as (in the same manner as before) reveals a very similar structure (fig. 17b). This means an optimisation start point is easy to find even if the landscape becomes more complex. T

Optimisation was performed for with , using the previously obtained rough manual start point. Contour plots in the six possible planes of two variables were made about this point, and superimposed on those of the original model. (fig 17c,d) Once again, although the basins were more complicated, they broadly aligned in behaviour, even though they didn’t entirely coincide.

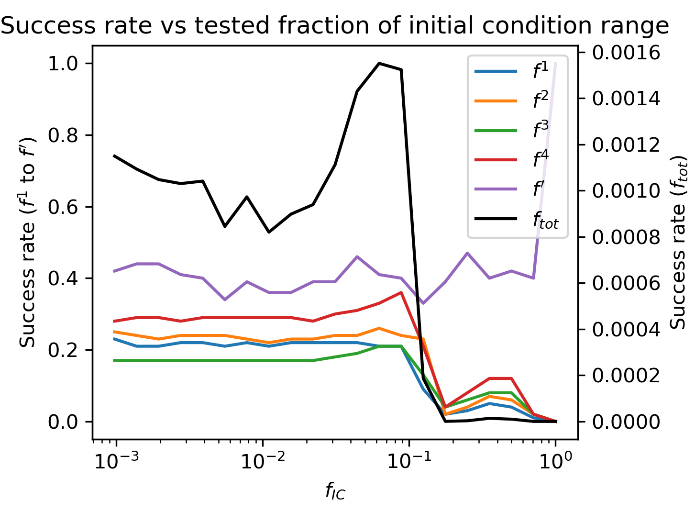
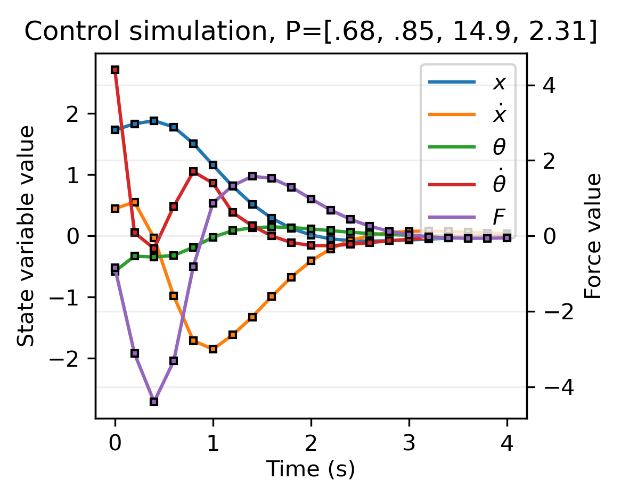
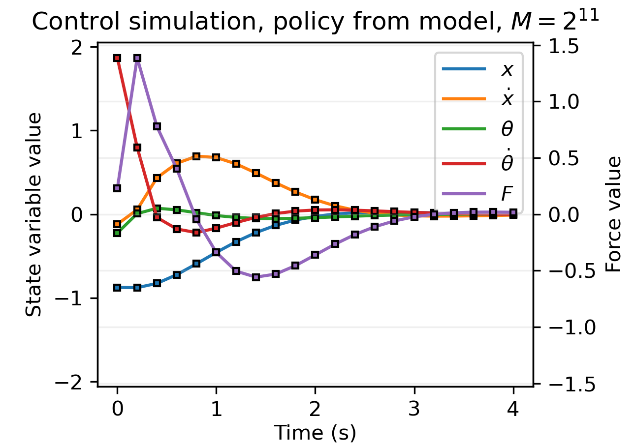
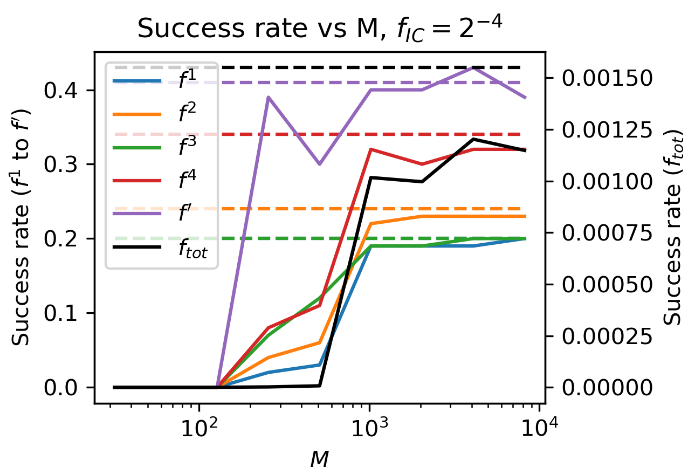
#### Performance vs M

For each M value, optimisation was repeated with , and the robustness of the optimisation results probed as before. (fig 18c)

For , controllers were unstable. All the other controllers were able to stabilise the cartpole from some initial states, and performance approached the ideal case from , showing that model-predictive control is extremely feasible for this system. (fig 18d). By inspecting cases where the models failed, it was clear that either oscillations were not killed fast enough, or the system went unstable, rather than (for instance) nonzero final cart position.



**Fig 18. Robustness and rollouts of optimised policies. C) has 2^-4 values superimposed (dotted).**

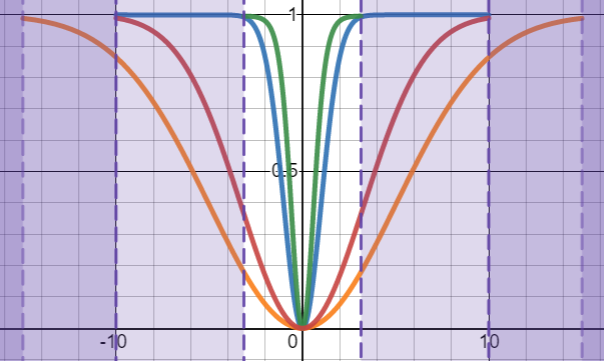


**a)**

**b)**

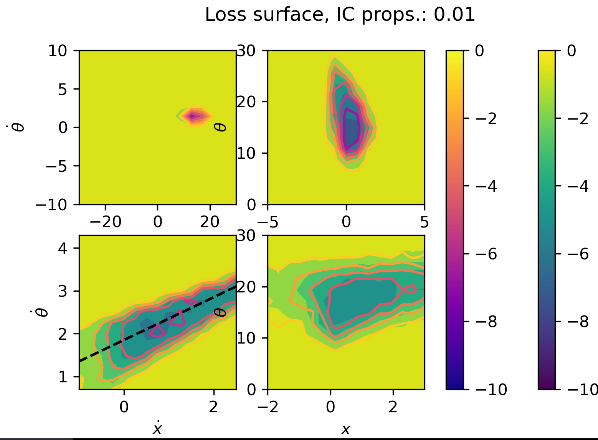
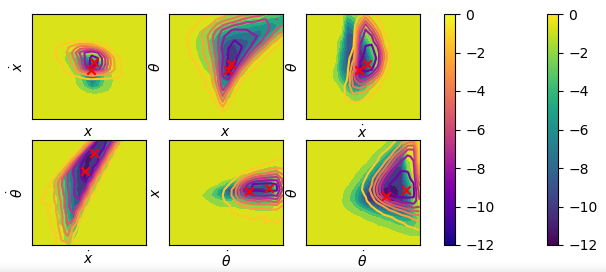
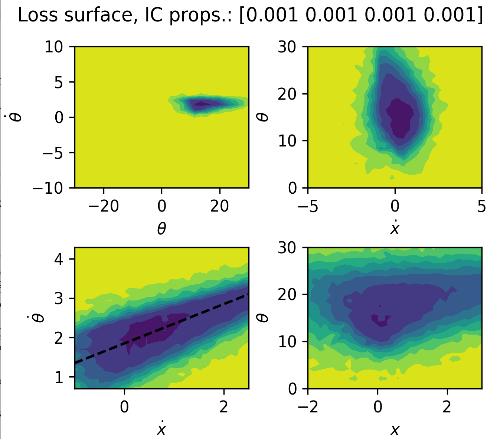
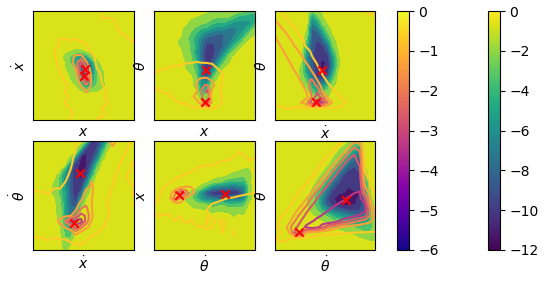
**c)**

**d)**



**Fig 16.** 2d scans demonstrating fit of M= model (as required)

**Fig 19. Contour plots of… a) refining problem to be harder and harder. Noise is present (particularly in this subplot; only 20 runs, others have 50) and hinders optimisation due to different ICs. b) M=2^7 superimposed onto corresponding plots (different f\_IC). c) M=2^7 scan plans around optimum, superimposed onto corresponding for real system. D) same as c for M=2^13.**



**a)**

**b)**

**c)**

**d)**

# Effects of noise

## Outline

In a real-world scenario where model-predictive control is to be used to stabilise the cartpole, noise will be present, which will degrade both the model and the policy. It is important to understand the effects of this noise to be confident in the reliability of a controller.

Two principal types of noise will be investigated:

1. Noise in observations of the model *(for example, due to a camera’s finite precision, or an inertial measurement unit’s inherent measurement noise)*
2. Disturbance to the dynamics of the model itself *(for example, due to wind or vibrations applying random forces).*

For (1), this is often due to a large number of small random factors and can be well-modelled by an additive i.i.d. Gaussian term in observations. The standard deviation of noise will be controlled with the single term , the fraction of the total modelled range for each parameter, in the sensible range [0, 0.5]?. There could also be an unknown bias, but due to time constraints this will not be explored. (including noise in application of F)

For (2), there will an additive term to the forces in the equations of motion, which will have a nontrivial power spectrum. This will be again modelled as i.i.d Gaussian (as a first approximation) added at each Euler iteration. A Gaussian with the same variance will be added to both the and update equations for simplicity, in the range [0, 10?]. Once again, a constant term could be explored but won’t be due to time constraints.

## General ideas of plots:

**A: Degradation in performance as increases.**

**B: Convergence on old version as N increases.**

## Observation noise

#### Effects on linear model

As increases, the fit unsurprisingly worsens – however, not by much (fig 20). Increasing the number of training points allows the model to perform almost as well as it did in the noiseless case – with (very achievable) noise all the way up to 20% of the modelled ranges can be almost entirely rejected, while a modest rejects noise well up to . As the model becomes buried in large amounts of noise, it is evident that no reasonable amount of data can recreate the original model – however noise this high is unreasonable in practice and would make the system impossible to control.

The effects of noise on modelled trajectories will not be discussed fully, since they are already terrible in the noiseless case – however, it is clear this performance can be recreated easily with sufficiently large .

#### Effects on nonlinear model

Effects on , starting with previously optimised parameters, -ish.

Note effects on scale parameter optimisation (presumably spreads out basins, increases optimum in general, not as important).

Convergence as increases for fixed .

Hopefully, conclusion that can associate increase in with decrease in with a fixed SNR value which affects fit quality, and all future plots can be done for fixed , varying SNR.

Effects on matching score (convergence as increases for fixed )?

#### Effects on linear policy

Start once again with actual system and optimise policy in noisy conditions.

##### Effects on previous best policy

For small amounts of noise, the previously obtained best policy still worked. At equilibrium, the system effectively became a linear filter, where the controller amplifies the input noise, resulting in relatively larger perturbations to the system (fig 23 a). Noise gains in steady state, calculated as , over many stable values, were found to consistently be

*(the refers to standard deviation in values)*.

Therefore convergence was defined as when ,over the last 25 observations and averaged across the four states, was less than .

Note that in this case the noise is amplified, rather than attenuated! This is because the controller doesn’t know the true position of the system at any timestep and so applies a slightly suboptimal force.

Policy robustness (as computed before) is plotted against noise quantity for this existing controller in fig 23 b.

In a real-world situation, the policy optimisation process would have to be done in the presence of noise. This should result in policies which better reject the noise.

##### Optimisation in the presence of noise

*Use same (initially graphical) process to optimise the new policy and check robustness.*

To plot: For noise values in range:

* Optimise policy parameters w/noise.
* Plot robustness vs noise

Then… **same shit again** with noise in dynamics.

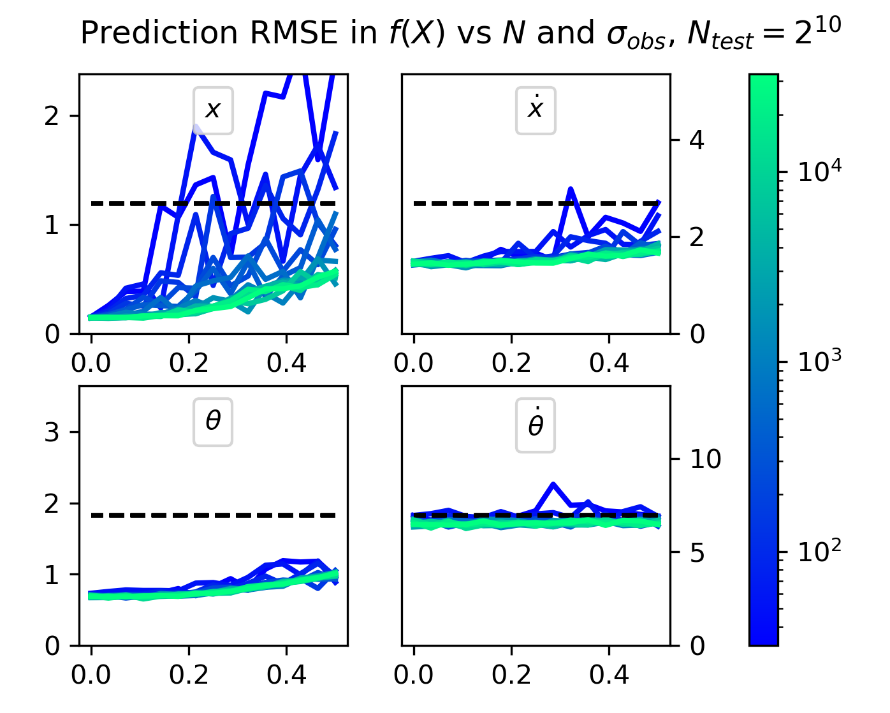
## Dynamics noise

## Conclusions

## Note noise is OK in training data

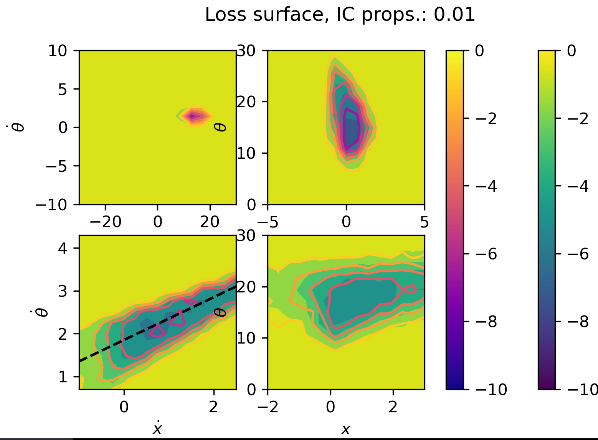
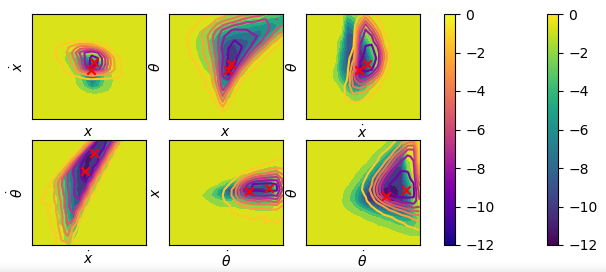
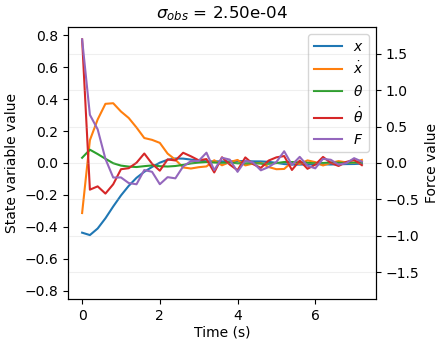
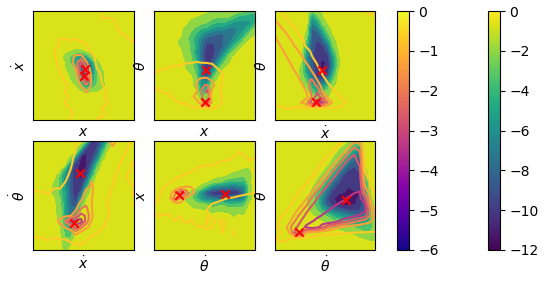
## But screws up policy

## Policies with smaller forces might be betters as more ‘noisy data’ per run



**Fig 20.** Linear model convergence with noise. Needs axis labels.

**Fig 23. a) a run using previously optimised parameters**



**a)**

**b)**

**c)**

**d)**

# Nonlinear Controller.

## Brainstorm…

3-phase control:

* Drive to the correct energy contour,
  + Some energy in the cart though
  + But can use energy \_in pole\_ as guide/or roughly use energy?
* Stabilise about the pole,
* Migrate back to the start position.
* Assuming IC range is below E=+small amount, no need for third step here.

Simple -> hard problem:

* Start without caring about cart position or velocity loss
* Then introduce cart velocity loss
* Finally introduce position loss

Superpose on linear controller?

* It takes over when E close to zero and near-upright. Could have some handing-over function (linear controller) + (nonlinear controller). Where =current value of loss function or similar.

# Appendix

NOTE – updated equations still missing a factor of L given to gabor. Corrected here, I think.

## A1: Energy definition

Let be kinetic energy, be potential energy with zero reference at , and be total energy.

and , so

## A2: Equations of motion

Defining the equations of motion are

## 13: Correlation matrix with N=5000

[[ 4.08391096e-04 2.00806862e-01 4.18320139e-02 2.62501158e-03]

[-3.52547831e-03 6.42016490e-03 2.06380111e-01 2.61166600e-02]

[-1.28462041e-03 1.28783926e-03 1.80439712e-01 1.97702879e-01]

[-1.32082079e-02 1.29174134e-02 1.25855381e+00 -6.06674820e-02]]

## Nonlinear model parameters

## Linear policy parameters