# Predictive Analysis, Uncertainty Modelling of Hall Thruster Propulsion
# Learning of Parametric Dynamical Systems
# Honors Capstone WN '22

Tejas Kadambi
Mentor: Professor Alex Gorodetsky

Apr 25, 2022

# Contents

# List of Figures

# 1 Motivation

The Plasmadynamics and Electric Propulsion Laboratory at the University of Michigan has generated some experimental data from a Hall-Effect Ion thruster. Coupled with that, they have derived a rough model that governs the data, containing some unknown parameters including normalization constants and other coefficients. The general form of an unknown model is shown in Equation 1, where $\theta$ represents the unknown parameters in the model that we wish to learn, and $\xi$ represents some noise disturbance.

$$y = f(x, \theta) + \xi \tag{1}$$

The generated data models the inputs and outputs of the thruster over time to form a time-series dataset and create what is known as a dynamical system. The goal of this capstone project is to learn the dynamics of that system, and these dynamics are governed by the parameters of the system. We can characterize the posterior of the parameters (or an updated belief) Bayesian Inference, specifically a Markov Chain Monte Carlo variation: Adaptive Metropolis Hastings. Once the posterior distribution has been generated, it can be analyzed to identify which points in the distribution are most likely to occur, or in other words, have the highest density to predict and simulate, with uncertainty, different conditions and trajectories for the model under different operation conditions. The uncertainty ranges can be defined based on the entire distribution as a whole or using credible and confidence intervals.

# 2 Bayesian Inference

Bayesian Inference is an advanced statistical inference method, using Bayes Rule (Equation 2) to update some hypothesis or initial belief using conditional probability relationships as more data and evidence becomes visible.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2}$$

Bayesian Inference begins with some prior understanding of the unknown parameters. This could be a range of values or an assumption as to what distribution the samples will fall in. Before beginning the sampling process, an expression needs to be created describing the likelihood. This represents the likelihood of the sampled values matching the data given our prior understanding. Next, the evidence/data is considered and matched against the prior using the likelihood. The last step involves updating our current belief after seeing the data using the current samples, which generates the posterior.

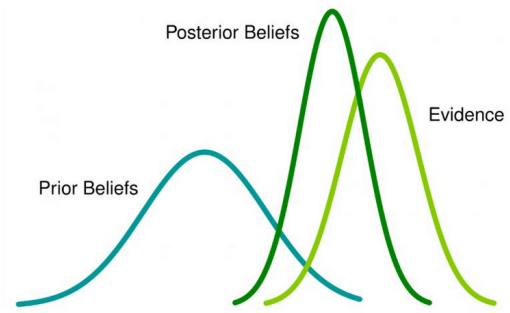


Figure 1: Bayesian Inference Visualization [1]

Figure 1 above shows an evolution of belief. Initial belief could begin anywhere with the prior and then shifts towards the evidence. It will never fully match the evidence because it is a hypothesis update, not a hypothesis replacement. The log of the posterior, mathematically, is equal to the sum of the log prior and log likelihood. It is characterized after several samples are put into place based on how close the generated trajectories are to the actual data.

## 2.1 Metropolis Hastings Methodology

The particular Bayesian Inference technique that was employed for this project was Adaptive Metropolis Hastings Markov Chain Monte Carlo (AMHMCMC). The standard Metropolis Hastings algorithm begins by defining the probabilistic model which represents the posterior of the parameters. In situations where the relationship that governs the data is known, the model is just the governing equation with the parameters unknown. If the equation itself is not known, the model can be defined as a neural network, where the weights are unknown. For non-linear models, the log likelihood is evaluated as a squared error between the actual evidence/data and the current estimate, normalized by the variance of the noise (Equation 3), where $y_i$ represents the data, $\sigma$ is the noise disturbance, and $M(x_i, \theta)$ represents the model output given the current sample.

$$log(\mathcal{L}(\theta)) = -\frac{1}{2}\frac{(y_i - M(x_i, \theta))^2}{\sigma^2} \tag{3}$$

The sampling process begins with an initial sample and an initial covariance. This initial sample represents a preliminary estimate of the unknown parameters. For instance, if 3 parameters are unknown in the probabilistic model, then every sample would be represented as a vector of length 3, with each of the values representing an unknown parameter. There are a few techniques which can be used to optimize the initial sample, which will be discussed later. Centered around the current sample, a new sample is proposed, generated according to the standard normal, or a Gaussian distribution, with the standard deviation being the square root of the provided covariance matrix. This can be computed using Singular Value decomposition or a Cholesky decomposition. Now, we have two points. One representing the current sample, and the other representing the newly proposed sample, which will be in the same region. The samples are inputted into the probabilistic model to generate a time-series trajectory which can be used in the log likelihood to determine the squared error difference between the truth (unknown) and the current sample. Each time the AM algorithm generates a sample, it holds the current estimate of the parameters ($c_{norm}$ and $k_{norm}$ in this case). This estimation is plugged into the model to generate $M(x_i, \theta)$, which is subtracted from our noisy data (the truth $y_i$) and squared to get a mean squared error which is normalized by the effect of the noise. If there the sample is exactly the same as the truth, there will be no mean square error and the log likelihood will be 0. Larger an larger deviations will return larger negative numbers. In the event that solve_ivp returns early or returns nonsense, the log likelihood returns -$\infty$ (equivalent to the largest of deviations). If the newly proposed sample has a density greater than the current sample, it is probabilistically favorable to move to that new location.

$$a(x, y) = min(1, \frac{\pi(y)}{\pi(x)}) \tag{4}$$

The probabilistic densities are weighed against each other and the proposal to move from the current sample to the new sample is either accepted outright or accepted with some conditional probability depending on the ratio of the densities. This acceptance probability (Equation 4) can be compared to a randomly, uniformly generated number between 0 and 1 to determine whether the proposal is

accepted or rejected. If accepted, the current sample moves to the proposed sample location. If not, the current sample stays where it is and the density increases. The sampling process is repeated several times (10000+) until an entire distribution of samples has been generated which can be used to analyze the parameter posteriors.

## 2.2 Metropolis Hastings Variations

Additional variations of the standard MHMCMC include the employed method, Adaptive Metropolis [2][3], as well as Delayed Rejection[4] and Delayed Rejection Adaptive Metropolis[5]. The AMHMCMC keeps track of all the samples as they are accepted and rejected to create a rolling sample mean and rolling sample covariance. These adapting mean and covariance values are used to propose new samples, essentially sampling from the center point of the distribution instead of the current sample. This proves to be optimal in case the current sample is in a low probability area and avoids the problem of getting stuck in that low density region by proposing additional low probability samples. The adaptation equations are available in Equations 5 and 6.

$$\bar{x}_k = \frac{1}{k+1}(x_{(k)} + k\bar{x}_{k-1}) \tag{5}$$

$$S_{k+1} = \frac{k-1}{k}S_k + \frac{s_d}{k}[\xi I + k\bar{x}_{k-1}\bar{x}_{k-1}^T - (k+1)\bar{x}\bar{x}^T + x_{(k)}x_{(k)}^T] \tag{6}$$

The Delayed Rejection method delays when a particular sample is rejected. Once the acceptance probabilities (Equation 7) are generated, if a proposed sample is rejected, instead of increasing the density at the current location and proposing a new point, the algorithm tightens the covariance matrix and resamples from that point to generate a sample closer to the current sample.

$$a_2(x,y) = min(1, \frac{\pi(y_2)q_1(y_2,y_1)[1-a_1(y_2,y_1)]}{\pi(x)q_1(x,y_1)[1-a_2(x,y_1)]}) \tag{7}$$

This can be done as many times as required. This effectively increases the acceptance ratio of samples, making the final posterior distribution much more densely populated if some previously generated distributions are too sparse. The Delayed Rejection Adaptive Metropolis combines the mean, covariance adaptation and the rejection delay elements and implements them both.

## 2.3 Initial Sample Optimization

Optimizing the initial sample and covariance can be key to generate distributions that effectively represent the posterior of the parameters. A standard implementation involves generating some random initial sample with identity covariance, but this can prove to be quite undesirable.

### 2.3.1 Burn-In

A commonly implemented tactic is Burn-In. Once the sampling process is complete, a small percentage (2%, 5%, 7%) of the entire distribution is discarded. These samples would be the first few samples accepted while generating the distribution, and this would eliminate the possibility of the initial sample being in a low probability area and skewing the overall distribution.

### 2.3.2 Laplace Approximation

Another implemented tactic is known as the Laplace Approximation. This algorithm attempts to generate its own distribution, with a Gaussian fit, that is centered around the MAP point and also returns a covariance matrix that closely matches the curvature of the target density. The MAP point, also known as the maximum a-posteriori, represents the mode of the full posterior distribution. With the Laplace Approximation, a reliable starting point can be utilized to avoid initially getting stuck in a low probability region. The outputted MAP point and target covariance are simply approximations and could need to be hand-tuned. Based on the final sample acceptance ratio, the covariance matrix can be scaled such that between 30% and 45% of proposed samples end up being accepted. The derivation (Equation 8) involves linearizing the log posterior through a Taylor expansion around a given point ($x^*$), such that $x = x^* + \delta x$.

$$log f_x(x) = log f_x(x^* + \delta x) = log f_x(x^*) + \nabla log f_x(x)|_{x=x^*}\delta x + \frac{1}{2}\nabla^2 log f_x(x)|_{x=x^*}\delta x^2 + ... \quad (8)$$

## 2.4 Analyses

The generated sample distribution can be analyzed in several ways. Visualizing the distribution of the samples yields a general idea as to whether it matches a target distribution, is aligned with prior beliefs, and what a singular best guess for what the truth is. A quick sanity check can be performed to ensure that the samples are in fact being generated randomly. Instead of visualizing each of the samples relative to the distribution, if they're shown as a function of time, truly randomly generated samples will mimic white noise, confirming good mixing. Essentially, there's minimal correlation between one sample and the next and without any discernable patterns or trends, it's fair to confidently state that the samples were in fact generated randomly. Since a newly proposed sample is going to be generated with relation to the previous sample and a rejected sample involves storing the current sample again, there will be some correlation between one sample and the next. The autocorrelation between the initial samples and successive samples can be visualized by monitoring the covariance between samples some distance apart. As the lag (distance between sample numbers) increases, the autocorrelation should rapidly drop to 0.

# 3 Mass-Spring-Damper Implementation

This algorithm was initially implemented on a known target distribution just to ensure the target distribution matches a known outcome. After computing the log posterior, the truth was plotted along with the sampled 1D distribution after applying the AMHMCMC algorithm.

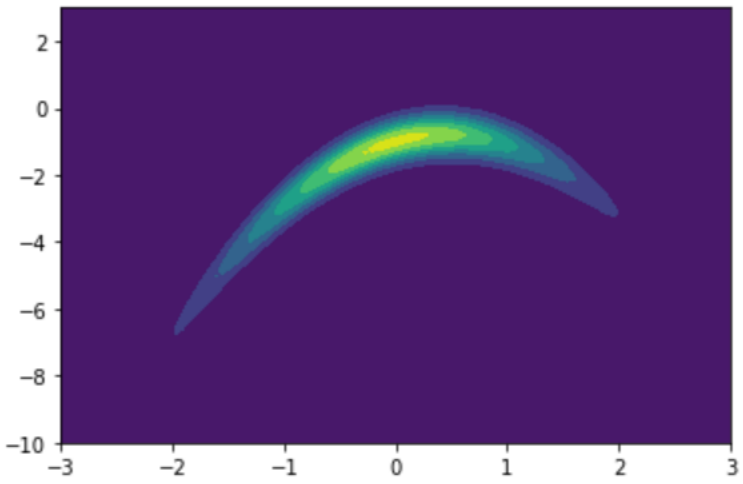


Figure 2: Banana Plot Truth Visualization

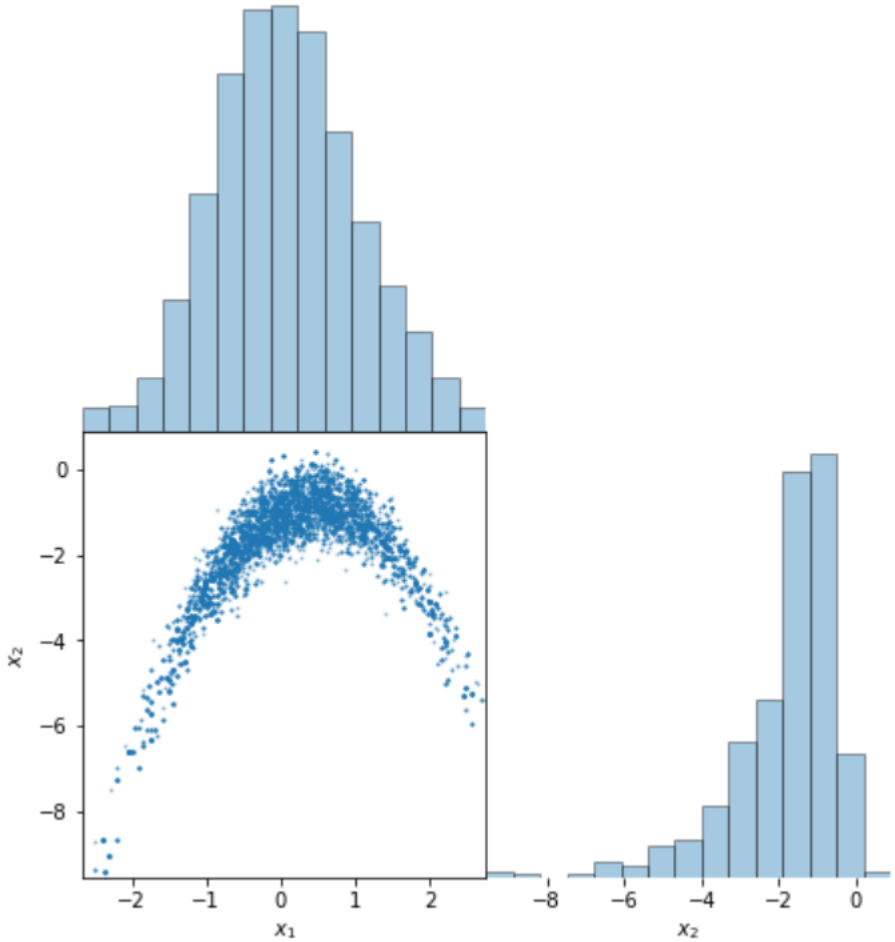


Figure 3: Banana Plot 1D Marginal Visualization

In this case, the visualization shows that the truth and the sampled 1D and 2D marginals match.

## 3.1 System Definition

Next, the algorithm was implemented on a known probabilistic model, where certain parameters were treated as unknowns. A standard damped oscillator model without a forcing term, which can be modelled in circuitry, but also in physics and mechanical engineering as a mass-spring-damper system (Equation 9). The states of the dynamical system are the position and velocity of the rigid body, and they are governed by the spring and damper, specifically the spring constant and damping constant and mass of the rigid body.

$$\ddot{x} = -\frac{c}{m}\dot{x} - \frac{k}{m}x \tag{9}$$

The ordinary differential equation (ODE) was defined to be the probabilistic model with unknown parameters being the mass-normalized spring and damping constants.

$$\ddot{x} = c_{norm}\dot{x} + k_{norm}x \tag{10}$$

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\end{bmatrix} = \begin{bmatrix}0 & 1\\ c_{norm} & k_{norm}\end{bmatrix}\begin{bmatrix}x\\ \dot{x}\end{bmatrix} \tag{11}$$

In this case, since the model is known and this exercise is being conducted simply to verify and debug the Adaptive Metropolis model, data was synthesized and then perturbed with some noise.

## 3.2 Parametric Case

This analysis is useful for analyzing a single dataset to determine what the parameters are. If the parameters are change and multiple datasets are synthesized or generated, the analysis can be run several times to generate distributions and predictions for all the different parameter combinations. However, if the relationship between the parameters and the states are not known, but the actual parameter values are, the parametric dynamical system can still be learned using the above method. In the standard mass-spring-damper example, we know that the dashpot coefficient is the velocity's coefficient and the spring constant is the position's coefficient. In the scenario where multiple trajectories were available with different parameter combinations, the relationship tying the coefficients to their respective states could be learned as well. The new log likelihood is the computed as the sum of the likelihoods of all the individual trajectories as shown in Equation 12.

$$log(\mathcal{L}(\theta)) = -\frac{1}{2}\sum \frac{(y_i - M(x_i, \theta))^2}{\sigma^2} \tag{12}$$

## 3.3 Results

Several datasets were generated using different parameter combinations and the results are shown below, for both the single trajectory and parametric case. The ODE was run to generate the truth for our AM algorithm and it was then disturbed with some noise (std = 1 in this example). The decaying oscillation on the left in blue represents the position of the mass, and the decaying oscillation on the right in orange represents the velocity of the mass. The scattered blue dots show the distorted data points.

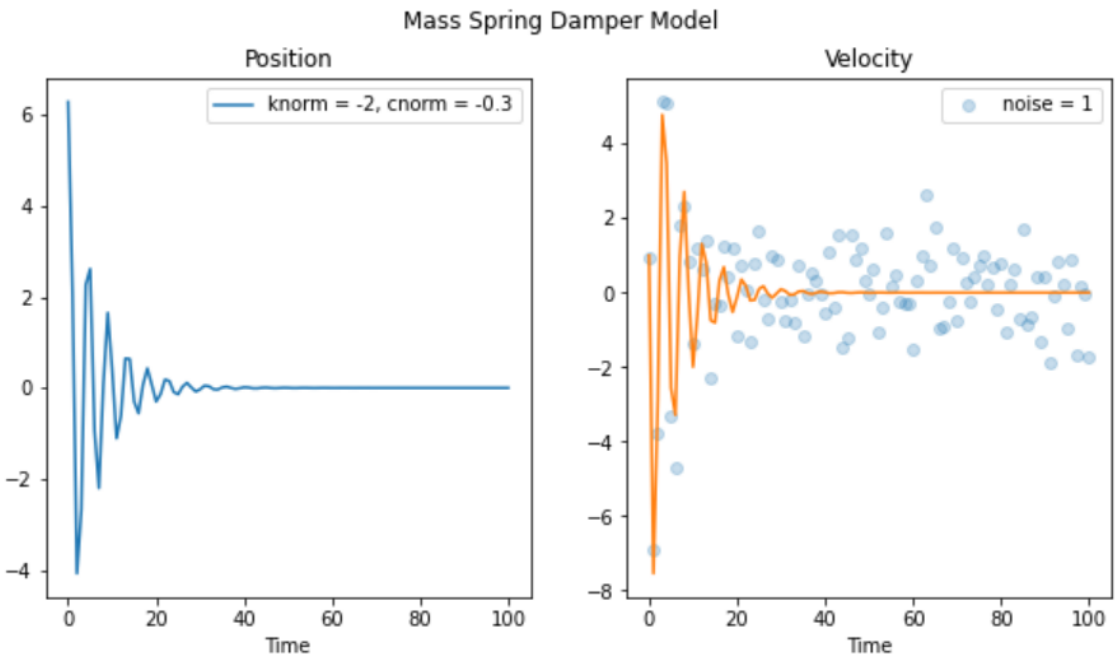


Figure 4: MSD Truth Generation

The log likelihood is calculated in Equation 3. In this case, $\theta$ is $c_{norm}, k_{norm}$.

The algorithm begins with a computation of the MAP point and some approximation of the covariance matrix with is computed using a Laplace approximation. The generated MAP point for this example was [-2.01993501, -0.31139041], which is very close to the truth of [-2, -0.3]. The AM algorithm is run for 10000 samples with a $k_0$ threshold of 100, where $k_0$ represents the number of samples before the adaptation process beginst to take effect. Acceptance rate for this trial hovered around 56%. Plots are shown below.

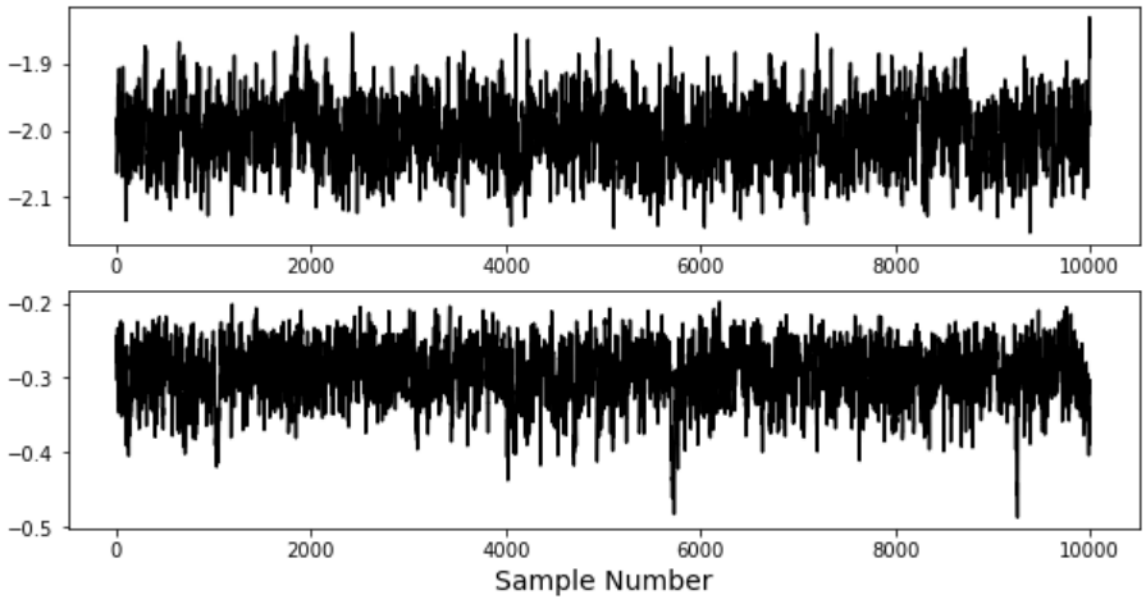


Figure 5: Sample Trace

The sample trace looks like white noise, which is good. The samples are being generated randomly, which is what we want.

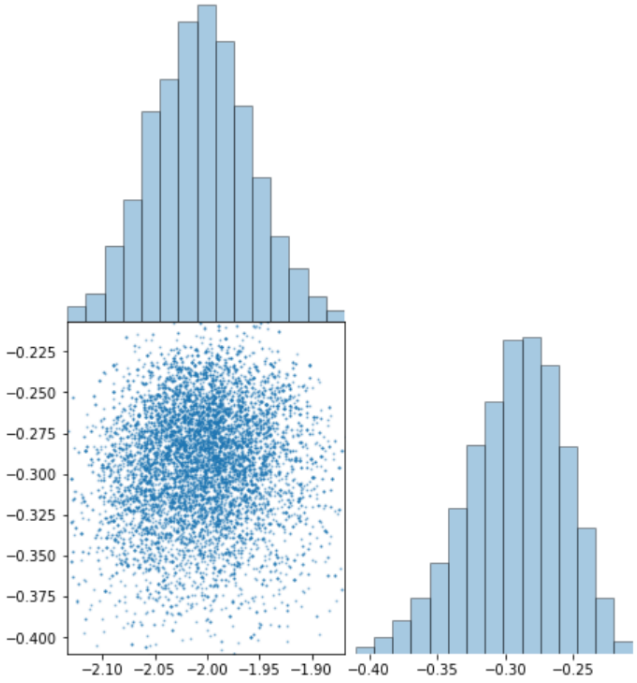


Figure 6: 1D Marginal

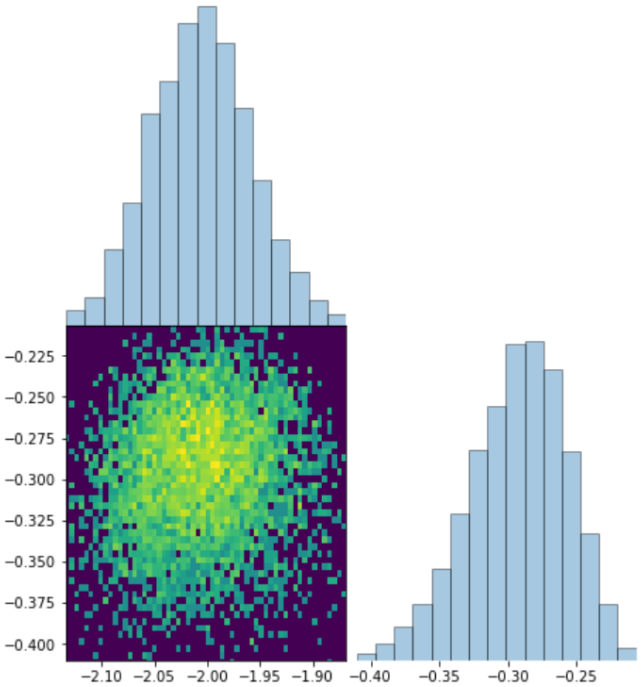


Figure 7: 2D Marginal

These samples are all closely clustered around the expected parameter value with very little

deviation, which is very good (and what is required).

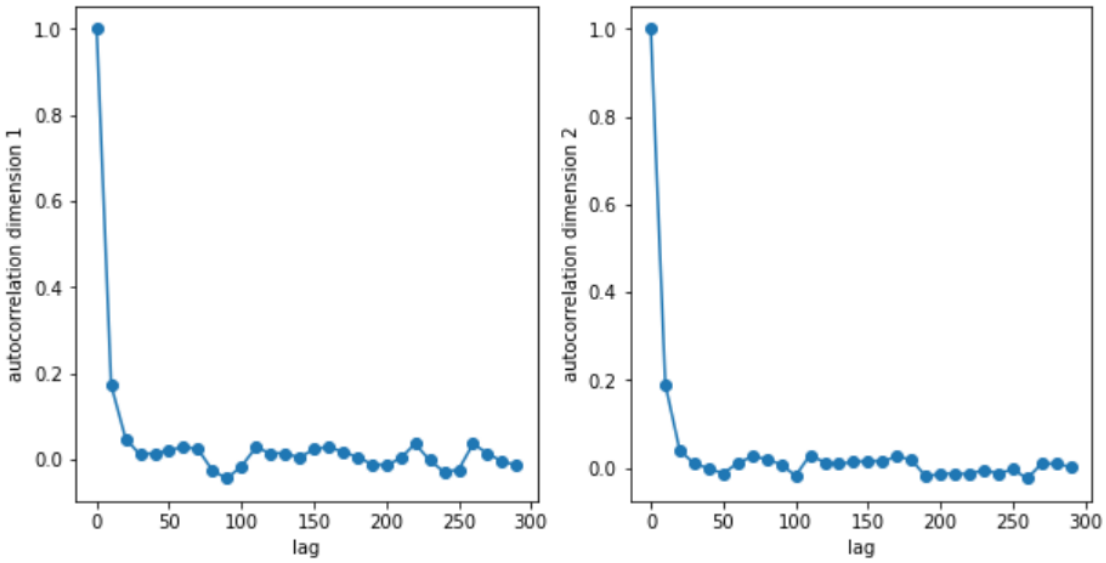


Figure 8: Autocorrelation

Autocorrelation drops to 0 at a lag of around 25-30 samples.

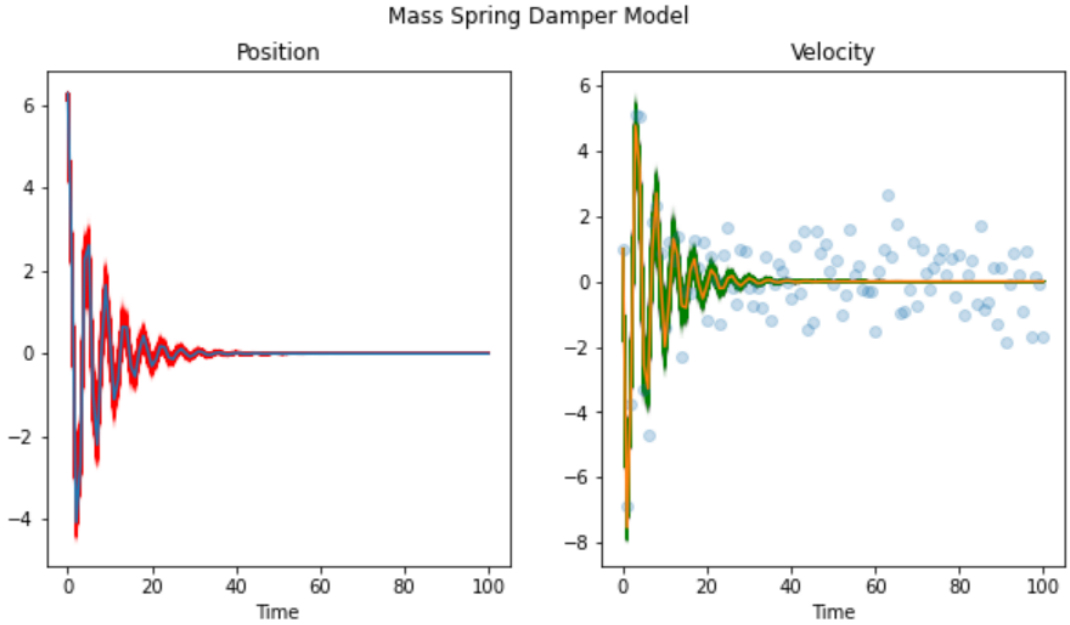


Figure 9: Sample Overlay

All 10000 samples are plotted with an opacity of 0.01 behind the original model. As expected from the marginals and sample traces, they match the curve very tightly. This took 2min, 48sec to generate trajectories and plot all 10000 samples.

All the initial truth generations and the sample overlays are shown below for the remainder of the iterations. All the sample traces look like white noise and cluster around the expected value. The 1D and 2D marginals all look Gaussian and cluster around the expected value. If a coefficient

is very close to 0 (-.075, for instance), the Gaussian holds its shape, but an outlier sample here or there is very prominent. It does little to affect anything, but just stands out because of how small the numbers are relatively.

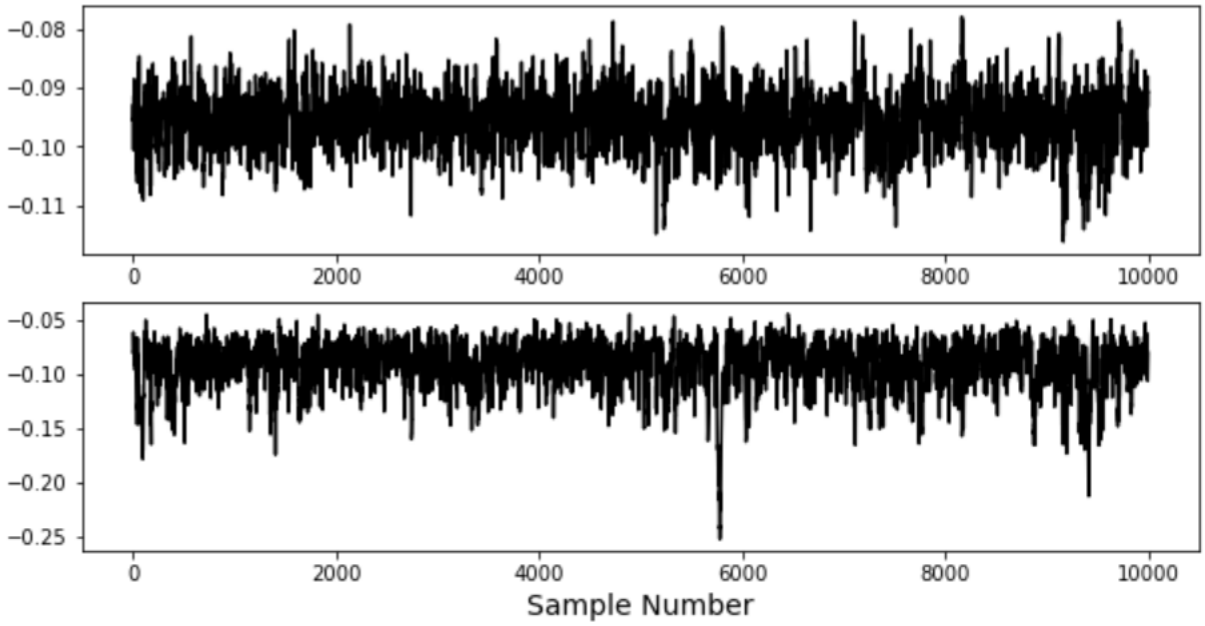


Figure 10: Outlier Emphasis Example

Again, this isn't really any cause for concern because the marginals for this particular example show a tight cluster, and it's just one bad sample.
The IAC reaches 0 around 25-30 samples for all the iterations.

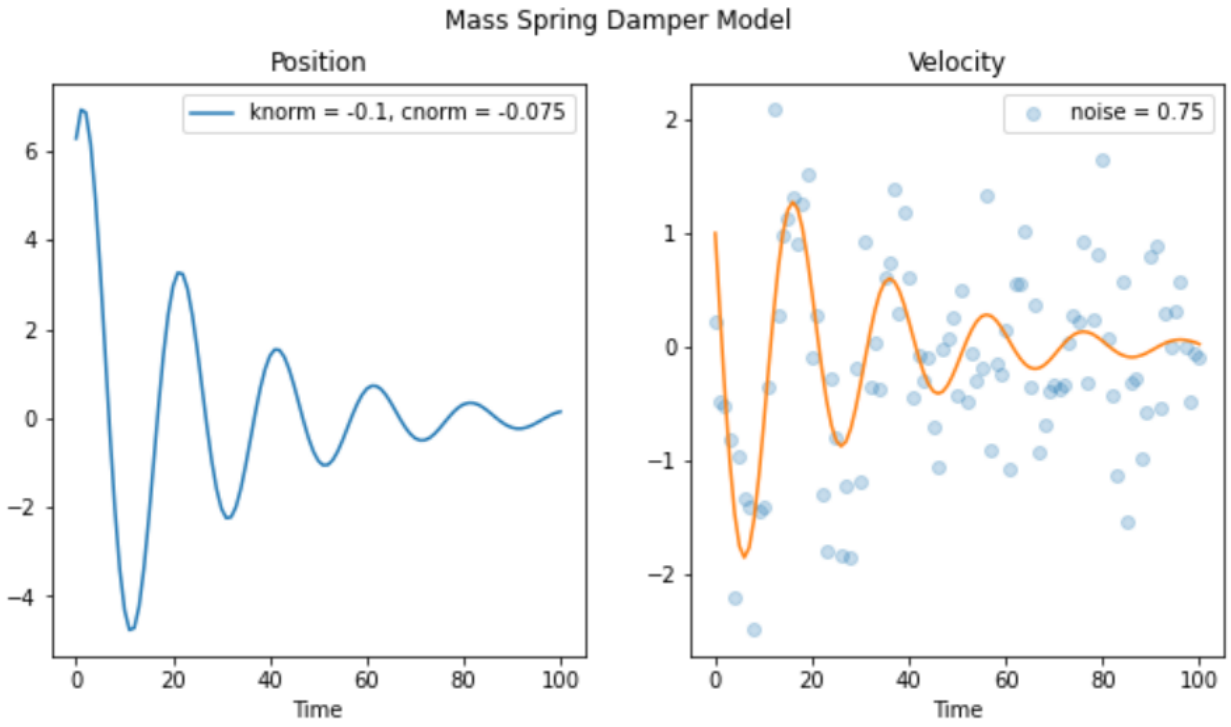


Figure 11: Iteration 2 Truth Generation

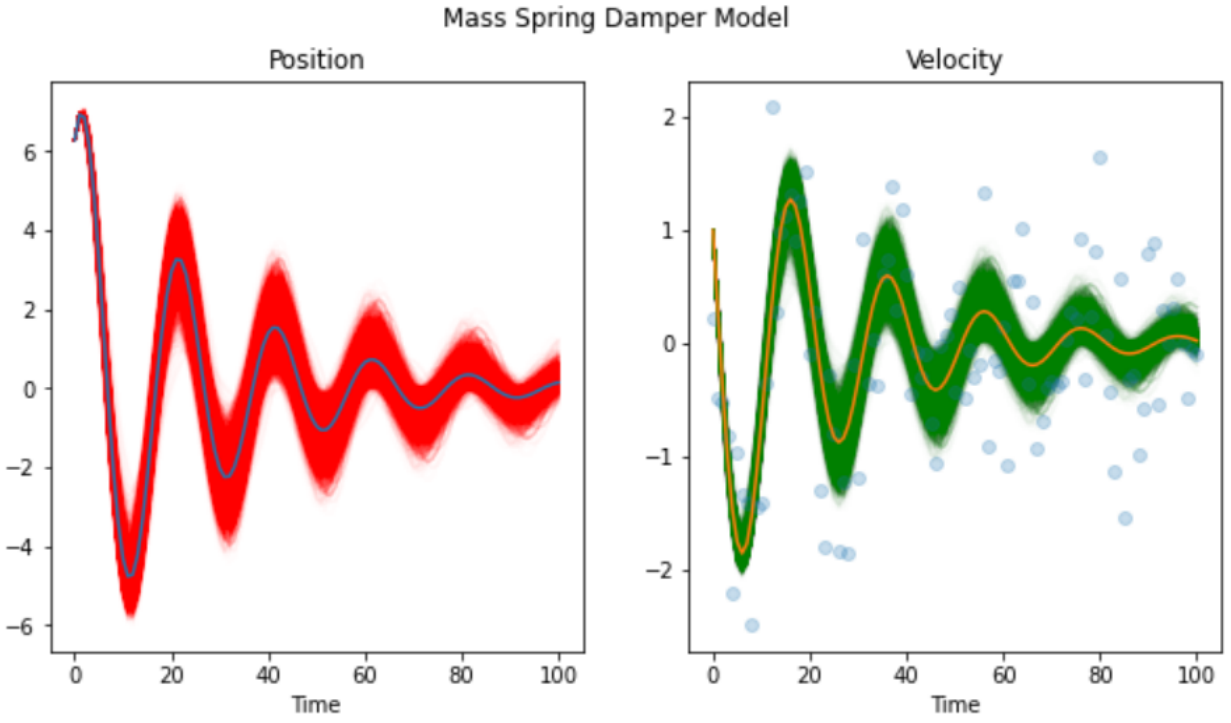


Figure 12: Iteration 2 Sample Overlay

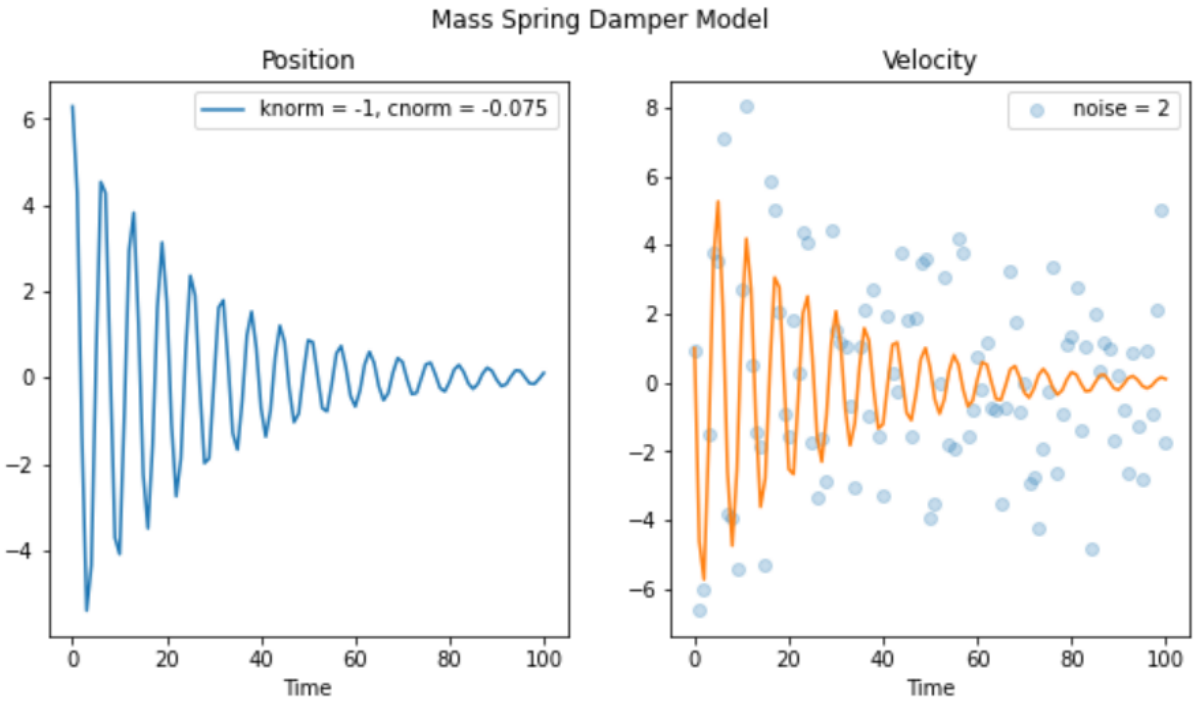


Figure 13: Iteration 3 Truth Generation

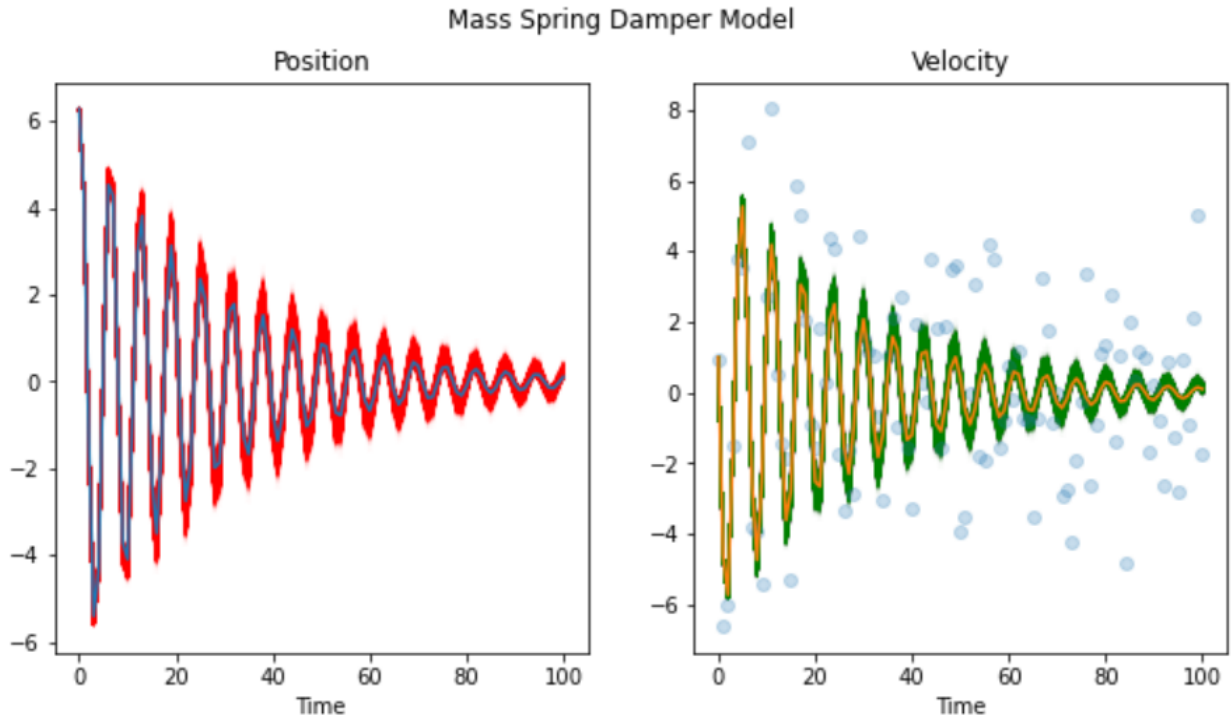


Figure 14: Iteration 3 Sample Overlay

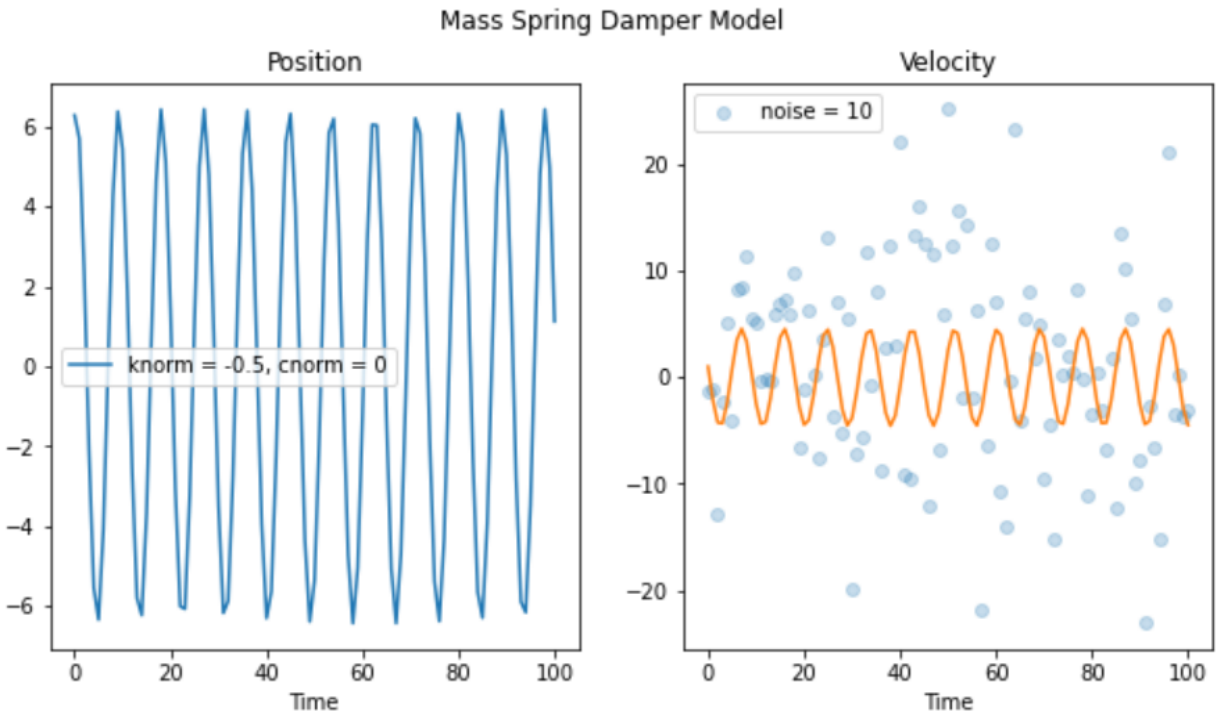


Figure 15: Iteration 4 Truth Generation

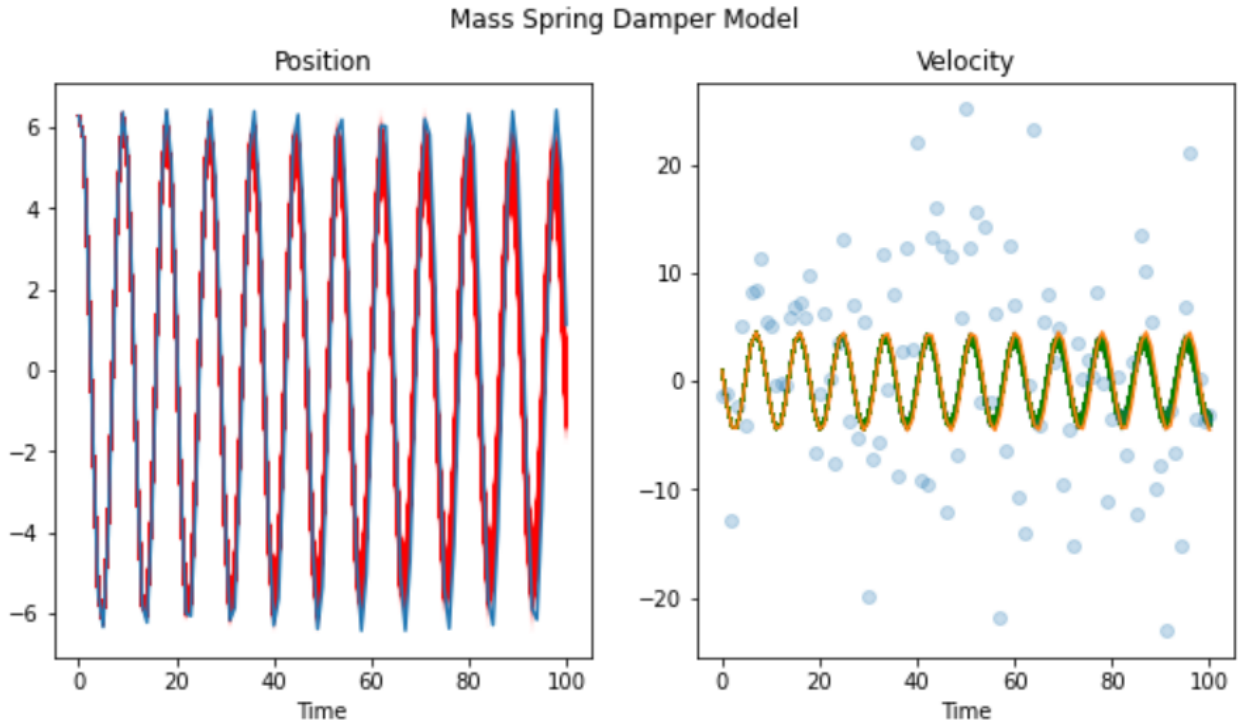


Figure 16: Iteration 4 Sample Overlay

One point of concern is that the samples are clustering very closely around the initial guess. That means that if the MAP point estimate is not good, the resulting trajectories are not as accurate,

however the initial errors are solved through Burn-In once the target density is found. This makes large noise (std ¿25) problematic because the Laplace approximation has trouble deciphering the data and cannot generate a proper MAP point estimate. However, noise levels this high are an entirely different problem to solve because the model is largely unrecognizable with such high levels of disturbance.

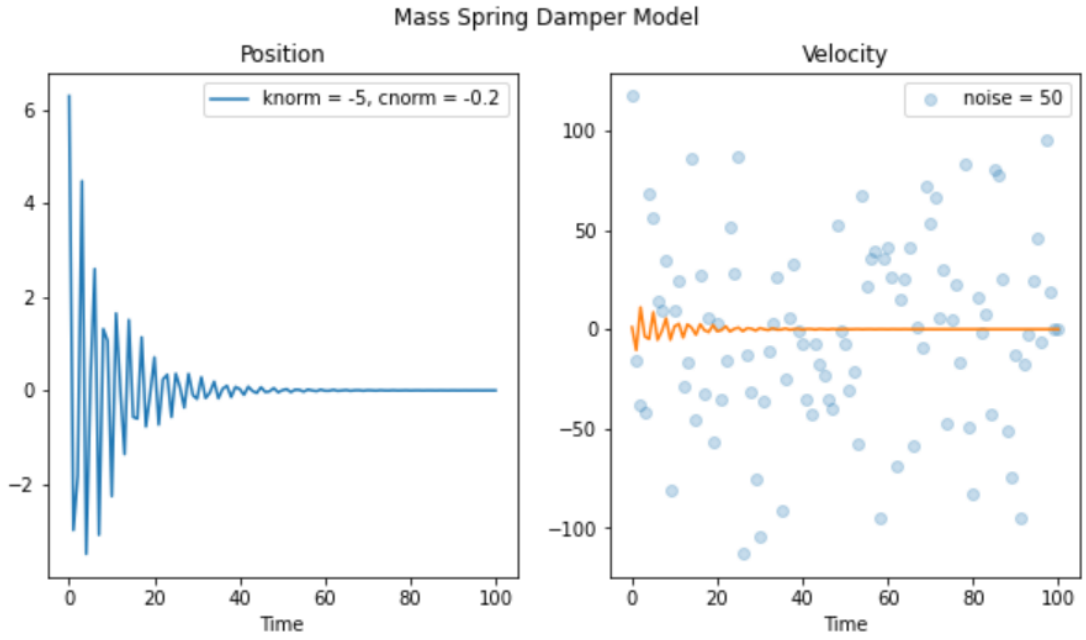


Figure 17: Iteration 5 Truth Generation (high noise)

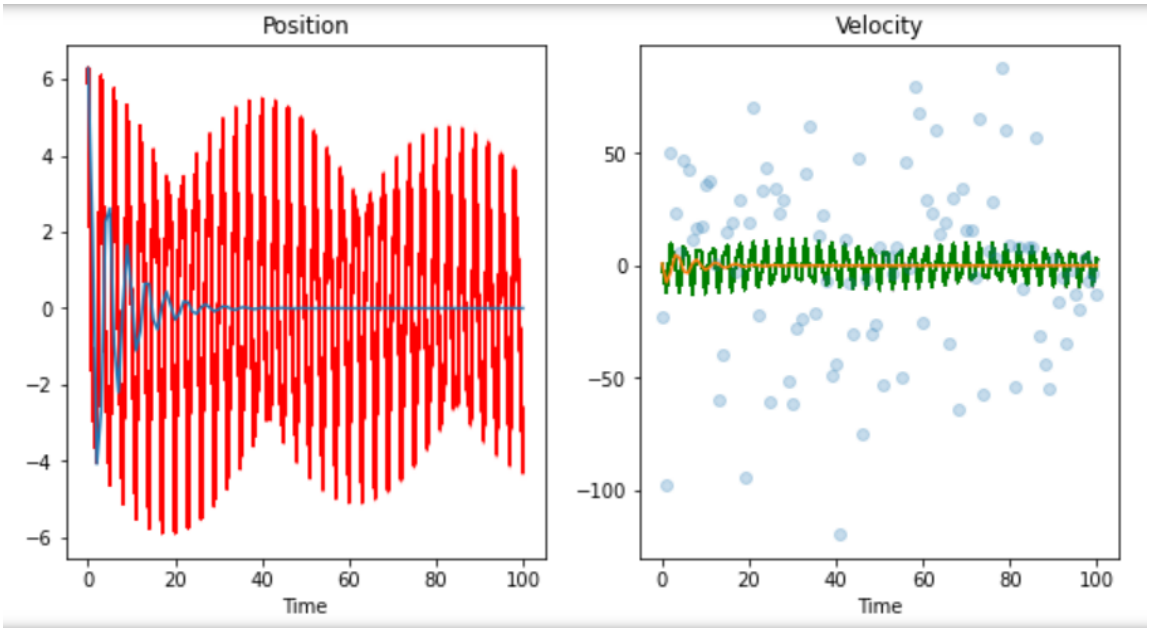


Figure 18: Iteration 5 Sample Overlay (bad posterior estimate)

### 3.4 Thruster Data Implementation

Now that the AMHMCMC implementation has been thoroughly tested and verified, training began on PEPL's datasets. The provided governing equations for the dynamical systems tracked the dynamics of the densities of the ions and neutrals in the channel. The provided datasets tracked the evolution the currents and voltages of the cathodes and anodes with an input/output relationship that was not present in the model. Additionally, the provided equations are extremely rough and did not converge to a steady state. As a result, a new probabilistic model was generated. Initially a polynomial approximation was used, attempting to find the coefficients and the relationship between the inputs and the outputs. This was essentially a brute force method because the polynomial order was unknown and there could have been exponential or logarithmic terms that this model left out. To improve the probabilistic model, a feed-forward neural network was created. The model first began with one hidden layer and then moved to two and then eventually included a time-delay state. The parameter estimation eventually partially characterized the posterior, achieving a very loose parameter approximation. Unfortunately, PEPL has asked that I don't share the data in too much detail, so the axes labels and context have been left out. Shown in blue is the actual data and shown in orange is the model estimation at the average location of the generated samples after running AMHMCMC.

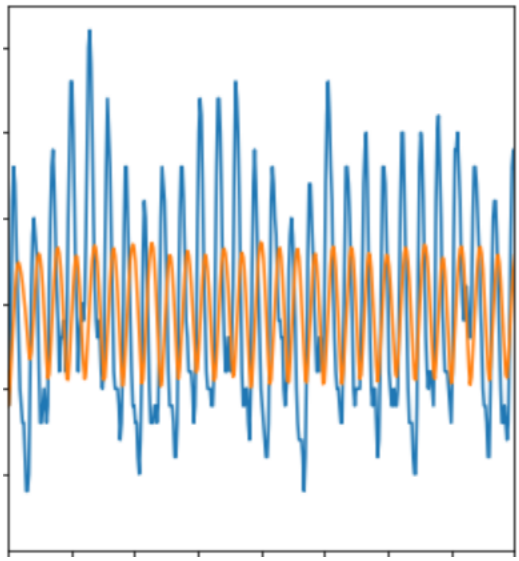

Figure 19: Parameter Approximation Achieved

As shown in Figure **??**, the model approximation was not able to fully model the characteristics of the data. A rough fit was managed with some subtle fluctuations in the amplitudes, but this requires more fine-tuning. This particular approach employed a state-delay double hidden layer neural network, learning 56 parameters representing the unknown weights.

# 4 Future Work

Attempts to characterize the posterior and determine the parameters achieved a partial characterization and a rough approximation of the parameters, which leaves some room to continue working.

### 4.1 Improve Neural Net

Some continuations of this project involve optimizing the neural network model to more accurately match the peaks and troughs of the oscillations in the original dataset. This will have to be done experimentally by finding the optimal number of hidden layers and nodes per layer. Increasing the number of unknown parameters in the model through use of more layers and nodes will significantly increase the computation time; as a result, this will require increased processing power and a more efficient way to characterize the probabilistic model.

### 4.2 Noise Learning

Currently, all the simulations have been run with a Gaussian prior. It assumes that the posterior distribution will model a Gaussian fit, which requires the noise in the dataset to be known. Implementing a noise learning approach to identify what the noise that the data is perturbed by requires assuming what is known as an Inverse Gamma prior. The log prior and log likelihood equations change as follows, shown in Equations 13 and 14:

$$log(f_\theta(\sigma^2, \alpha, \beta)) = -(\alpha + 1)log(\sigma^2) - \frac{\beta}{\sigma^2} \tag{13}$$

$$log(\mathcal{L}(\theta) = -\frac{1}{2}[log|\Gamma| + (y - M(x, \theta))\Gamma^{-1}(y - M(x, \theta)^T)] \tag{14}$$

### 4.3 MHMCMC Variations

Lastly, delayed rejection can be experimented with to improve the acceptance ratio. This will naturally increase the lag between samples because samples that get accepted after being rejected several times will have been generated with an extremely tight covariance matrix, making them extremely similar to the current sample. It will also, however, increase the acceptance ratio and fill out the working distribution. This would prove desirable as one of the current issues is that several samples are being rejected because the model approximation does not properly characterize the desired posterior.

## 5 Acknowledgements

I would like to extend my gratitude to my mentor, Professor Alex Gorodetsky, for all of his patience and advice[6] throughout this capstone experience. I would also like to thank Professor Benjamin Jorns and PEPL for countless resources[7] and providing me the opportunity to analyze their data. Lastly, I would like to thank the Honors department, faculty, and organization for this amazing opportunity and experience.

# 6 References

# References


[1] Bayesian statistics.
https://www.analyticsvidhya.com/blog/2016/06/bayesian-statistics-beginners-simple-english/.

[2] Johanna Tamminen Heikki Haario, Eero Saksman. An adaptive metropolis algorithm, 2001.

[3] Adaptive metropolis demo.
https://chi-feng.github.io/mcmc-demo/app.html?algorithm=AdaptiveMHtarget=banana.

[4] Antonietta Mira. On metropolis-hastings algorithms with delayed rejection, 2001.

[5] Antonietta Mira Eero Saksman Heikki Haario, Marko Laine. Dram efficient adaptive mcmc, 2006.

[6] A. A. Gorodetsky. Ae 740: Statistical learning, inference, and estimation, Fall 2019.

[7] Ira Katz Dan M. Goebel. Fundamentals of electric propulsion: Ion and hall thrusters, 2008.