## 1 Introduction

In this project we attempt to estimate parameters $m(j)$ (the masses or collectively, the medium) for the following system of differential equations

$$m(j)\ddot{u}(j,t) = \begin{cases} u(j+1,t) - u(j,t) + F(\omega,t) & j = 1 \\ u(j+1,t) - 2u(j,t) + u(j+1,t) & 1 < j < 10 \\ u(j-1,t) - u(j,t) & j = 10. \end{cases} \tag{1}$$

The system in meant to model the interactions of 10 masses in a line, connected together via springs, see Figure 1. In this case, an external force is being applied to the left most mass, and we can imagine vibrations travelling like a wave through to the other masses. We do not have a specific example in mind, but one could imagine that one may want to predict features of such a system by measuring dynamics of one of the masses.

Without a specific example in mind, one can get lost in the numerous model assumptions which can greatly impact the performance of the parameter estimation. For example, it improves performance if one has a priori knowledge of the range of possible values for the $m(j)$. In what follows, we assume arbitrarily that the masses $m(j) \in [1,10]$. Furthermore the frequency $\omega$ of the forcing function can also play a role in the performance of the estimation. A small $\omega$ produces longer waves which would be less affected by the heterogeneity of the medium through which they travel, whereas waves which oscillate too rapidly could localize and therefore not probe into the medium far enough to retrieve information. There is also the question of what observations of $u$ one would have access to in practice. Therefore, we are going to make a lot of assumptions that could resemble something one may encounter in a physical scenario.

First, we assume the source has some fixed frequency $\omega = 2\pi$ and that its amplitude initially increases and then decays. Specifically

$$F(2\pi,t) = \operatorname{sech}(t)\sin(2\pi t). \tag{2}$$

Next, we assume that we can make observations of $u(3,t)$, that is we are pretending like our measuring device occupies an arbitrary spot in the medium, although the distance from the source can affect performance as well. The idea for such a setup is partly inspired by [5], where scientist used measurements from seismic waves which eventually became sound waves to estimate average global ocean temperatures at different times in the past. [1] They essentially looked at discrepancies in arrival times for these sound waves, since the water temperature is a significant factor in controlling their speed. To do this however, they had to actually measure discrepancies in so called “repeaters" since the location of an earthquake cannot be pinpointed precisely enough. They state, "the generation and propagation of seismic waves is sufficiently similar in repeaters that the source and structure of the uncertainties cancel out". In our case, we are pretending we know the form of the source a priori.

Finally, we should note the form of any model itself is an assumption. In our case, we have assumed constant harmonic (linear) forces between the $u(j)$ and that their dynamics are governed by Newton’s second law. Having access to the right model for a particular situation, even without knowing all the parameters is important because it allows one to generate data by simulating the model for many parameter values. These simulations can be used to train machine

[1]The connection between the system we are using above and water waves is complicated. The system above is meant to be a small linearized Fermi-Pasta-Ulam-Tsingou chain, which is known to have dynamics approximated by a famous partial differential equation called the Korteweg-de Vries (KdV) equation. KdV possesses travelling wave solutions and was intended to model water waves.

learning models that can then in turn be used to make predictions about the real world. Our situation is completely made up so we will have made-up real world (validation) data as well.

## 2 Methods

Throughout, let $m^*$ denote true parameters for some given set of data and $m$ some estimate. Let $t_i$ be a series of observation times. Recall that we are making observation of $u(3,t_i)$ and note that, although not explicitly indicated, $u$ is a function of the parameters $m$ as well. So let us write $u(j,t_i,m^*)$ to denote our observation. One may also assume that one does not observe $u(j,t_i,m^*)$ directly but instead observes $u(j,t_i,m^*)+\eta_i$ where $\eta_i$ can be thought of as independent observational noise. If we assume these are independent and with variance $r_i^2$ so the noise is heteroscedastic, one route to estimating $m^*$, outlined in [3], is by trying to minimize the sum of squares

$$m^{\text{est}} = \text{arg min}_m \sum_{i=1}^{n} r_i^2(u(3,t_i,m)-u(3,t_i,m^*))^2. \tag{3}$$

However, there is not necessarily a unique minimizer in (3) so one typically tries to include a regularization term such that

$$m^{\text{est}} = \text{arg min}_m \sum_{i=1}^{n} r_i^2(u(3,t_i,m)-u(3,t_i,m*))^2+\alpha_i(m_i-\mu_i)^2, \tag{4}$$

where $\mu$ is the mean of some prior distribution of $m^*$. The $\alpha_i$ can be thought of as our confidence in the prior. Minimizing (4) is slated to be difficult due to the high dimensionality (10 dimensions) of the parameter space. We do employ gradient descent below, but is it probably worth training some regression models first. Doing so can not only give us a place to start our minimization search but can also serve as a prior.

### 2.1 Regression

One aspect of our problem to be aware of is that even though (1) is a linear system, the solution $u$ does not necessarily depend linearly on the parameters. This is typical. For example in the equation $\theta x = 7$ where we seek the solution $x$, $x$ does not depend linearly on parameter $\theta$ even though the equation is linear in $x$. Therefore, we need to use non-linear regression methods. In the data analysis below we try both ridge regression and support vector machine regression (SVM) which can use the so called “kernel trick" to fit non-linear data.

In a linear regression algorithms, we try to fit the linear model (without intercept)$y=\theta^T x$ by minimizing some objective function. The kernel trick is a way of simplifying the implementation of a complicated feature map. We would use a feature map $\phi$ to transform data $X$ so that way it is easier to fit with a linear regression algorithm. For example if we had suspicion that the feature $y$ was quadratic in $x$, we could let $\phi(x)=x^2$ and fit the linear model using $\phi(X)$ as the data instead. Most of the time, a complicated feature map is needed, meaning it could be time consuming to calculate or even infinite dimensional. (Note that $\phi$ can lift $x$ to a higher dimensional space, the dimension of the sought parameters change correspondingly.) However, one can notice that in certain linear regression algorithms one never needs to calculate $\phi(x_i)$ by itself and instead needs to calculate values (inner products) such as $\phi(x_i)^T\phi(x_j)=k(x_i,x_j)$. This $k$ is often easier to compute. An example from [4] is as follows; suppose a promising feature map is $\phi$ where $\phi(x)$ returns a vector with its first $x$ entries as 1, and its remaining entry as 0. Such a feature map returns a vector which is infinite dimensional so it would be tricky to store in computer to say the least. However, $\phi(x_i)^T\phi(x_j) = \min\{x_i, x_j\}$, which is simple to compute. The feature map corresponding to the RBF kernel also turns out to be infinite dimensional. A discussion in [2] also gives some insights into why kernels can be useful. First, they give us another way of looking at our data besides through the lens of a feature map. That is kernel can be thought of as a function which measures the similarity between two samples in some specified way. Second, kernels can often be designed in a way that allows them to be computed faster, and one can design kernels without referencing a feature map.

In addition to being non-linear, our targets, $m$ (which we usually call $y$), have multiple outputs i.e. 10. In Scikit Learn, regression and classifier algorithms do not support multiple outputs natively. One must either use the MultiOutputRegressor or ChainRegressor to extend regression algorithms to have multiple outputs. These take estimators like RidgeRegression or SVM as parameters. MultiOutputRegressor fits an estimator to each target independently i.e. each mass is predicted independently. This can be easily parallelized using the *n_jobs* parameter. ChainRegressor fits targets one at a time in an order specified by parameter *order*. It makes predictions based upon the input features as well as the target predictions made earlier in the order. For example, in our problem, because the source inputs at the first mass, we might expect that knowing its value first would be helpful in learning the values of the other masses.

## 2.2 Numerical Gradient Descent

After fitting a regression model, we attempt to minimize (4) using numerical gradient descent. Let

$$f(m) = \sum_{i=1}^{n} r_i^2(u(3,t_i,m) - u(3,t_i,m^*))^2 + \alpha_i(m_i - \mu_i)^2. \tag{5}$$

It might not be obvious at first that such a function is differentiable in $m$, but note

$$\frac{\partial f}{\partial m_j} = 2\alpha_i(m_i - \mu_i) + \sum_{i=1}^{n} 2r_i^2(u(3,t_i,m) - u(3,t_i,m^*))\frac{\partial u(3,t_i,m)}{\partial m_j}. \tag{6}$$

Therefore, the differentiability of $f$ depends on the existence of partials $\frac{\partial u(3,t_i,m)}{\partial m_j}$, but it turns out this partial exists and is even continuous. Briefly, if we have a differential equation $\dot{y} = F(y,\gamma,t)$ where $\gamma$ is a vector of parameters and $F$ is $k$ times continuously differentiable with respect $y,\gamma$ and $t$, then $y$ is $k$ times continuously differentiable with respect to $\gamma$[1] .[2] However, since there is not nice way to calculate $u$, other than numerically, probably there is not a nice way to analytically find its partial with respect to $m$. Therefore, we approximate the gradient of $f$ at the current guess of $m$ using finite differences in order to determine what our next guess should be. That is we pick $h$ sufficiently small and find

$$g_i = h^{-1}[f(m_1, m_{i-1}, m_i + h, m_{i+1}, \ldots, m_{10}) - f(m_1, \ldots, m_{10})]. \tag{7}$$

We then update our guess $m$ by moving in the direction of the gradient with respect to some learning rate, $\varepsilon$.

$$m_{\text{new}} = m_{\text{old}} - \varepsilon g. \tag{8}$$

Smaller $\varepsilon$ allows the descent trajectory to settle more closely to some local minimum because with small $\varepsilon$ it is unlikely to overshoot. But since smaller $\varepsilon$ makes the descent take longer since

[2]There are a lot of technical details that I am skipping for brevity, but the upshot is if $f$ is smooth in every possible way, then so is the solution.

it requires more steps, it is unwise to keep $\varepsilon$ fixed or to manually try different values. Instead, one should perform a line search, that is, at each iteration let the computer test various values of $\varepsilon$ until one is found that maximizes descent the most.

It is possible to also use second order considerations to obtain a good value for $\varepsilon$. Note that, by Taylor expanding,

$$f(m_{\text{new}}) \approx f(m_{\text{old}}) - \frac{1}{2}\varepsilon g^T g + \varepsilon^2 g^T H g. \tag{9}$$

Here $H$ is a the hessian i.e. the matrix of second derivatives so the term $g^T H g$ is essentially the concavity along direction of the gradient. If $\varepsilon g^T H g < g^T g$, then $f(m_{\text{new}})$ may be larger than $f(m_{\text{old}})$, but picking $\varepsilon = g^T H g$ minimizes (9) in this case. In two or more dimensions, there is even more reason to use information about the concavity of the function. In Figure 2, a figure from [4] is recreated to show how always moving in the direction of greatest descent is not necessarily the most efficient method.

One last issue we need to contend with is the fact that the masses can only come from the interval $[1,10]$, and therefore our guesses should be constrained to that region. Probably the most naive method and the one we employ is to simply project any guess that falls outside of the constraint region back into the region. That is if the mass is greater than 10, we force its value to be 10 and proceed. But there other ways to handle the situation. We could parameterize $m_i$ in terms of an unconstrained variables, that is, we could let $m_i = 4.5 * \sin(\theta_i) + 5.5$ and now find the $\theta$ that minimizes (4) or we could minimize a so called “generalized Lagrangian" [4].

## 3 Data Analysis

We supposed we were given some real data $u(3, t_i, m^*)$ and asked to determine as best we could the true values $m^*$. (We have pretended there is no observational noise, since the problem is challenging enough.) We implemented the following strategy. We simulated (1), 10,000 times for 30 seconds and sampled at 1 second increments with different choices of masses to create training data. Since we are allowed to create the training data, there may be better strategies for choosing the masses, i.e. there could be a back and forth between training and creating training data, but in this case we chose masses randomly.

Once we had the training data, we scaled it and performed dimension reduction. We then split it so we had some testing data, which we used to fine tune the hyper-parameters of SVM and Ridge regressors. After some testing, it was determined that Ridge-regression performed better and faster. Compare Figures 3 and 4. A grid-search yielded the best value of the hyper-parameter $\gamma$, which is actually a parameter of the RBF kernel, and it turned out that chain regression did not yield any improvement to the model.

The regressor model could be used to predict any newly simulated validation data. We found using the validation data that the model was fairly good at predicting $m(1)$ (recall, we are measuring at $m(3)$) and steadily got worse until $m(6)$, after which, a guess of 5.5 proved just as good as the regressors's prediction. This trend was likely due to the fact that the source was located at $m(1)$ and measurements were taken at $m(3)$ and masses too far away were contributing little to the dynamics of the system. At this point, we could predict $m^*$ and be fairly confident that we likely had a descent approximation for the first couple of masses.

To make the approximation better, we used numerical gradient descent. The predictions made by the regression model now became the priors for the objective function (4). In principle, we would want to use the training data to find values for $\alpha$ that yield the best results or we could use our accuracy on the testing data as guidelines. In practice, we took $\alpha_i = 0$ for all $i$ because the algorithm proved too be slow to run on many samples, since it had to numerically compute a gradient by solving (1) at each iteration. Still, the guess we obtained from the regressor proved to be a better initial condition for the descent trajectory than some random initial condition. See figure 6 which shows why it is helpful to seed the descent algorithm with some educated guess. Most often, gradient descent yielded some improvement in accuracy of the priors, see Figure 5. On occasion, its prediction was stunning but probably lucky. Still, it seemed significant enough that it should be used to try to predict $m^*$.

Satisfied with our predicting capabilities (or rather, out of time), we apply our methods to the "real world" data. Our Ridge regression model predicted the masses to be (rounded) $[0.8, 3.1, 3.0, 5.0, 4.9, 5.8, 5.7, 6.0, 5.9, 6.3]$. Using this prediction as a seed, gradient descent predicted $[1.0, 2.0, 3.0, 4.1, 4.8, 5.9, 5.8, 6.0, 5.9, 6.3]$. For comparison, here is how gradient descent did with random initial state of 5.5 for all the masses: $[1.0, 8.1, 8.6, 5.7, 5.7, 5.0, 6.1, 5.2, 5.4, 5.5]$. Soon after, (fictional) scientist found a way to observe the masses directly and reported the true values as being $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$. It seems our prediction was fairly descent for masses closer to the source.

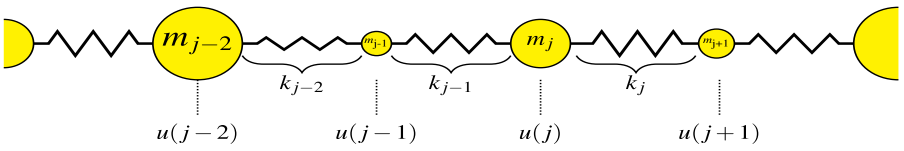

Figure 1: This figure shows how masses of various size can be connected via springs. In our specific setup, there are 10 masses and the left mass is subject to some external forces.

## References

[1] C. Chicone (2010), *Ordinary Differential Equations with Applications*. Texts in Applied Mathematics, Springer Science+Business Media, Inc.

[2] M.P. Deisenroth,A. A. Faisal, and C.S. Ong, (2020) *Mathematics for Machine Learning*. Cambridge University Press.

[3] B. G. Fitzpatrick (1991), Bayesian analysis in inverse problems. *Inverse Problems* **7**(675) 675-702. `https://doi.org/10.1088/0266-5611/7/5/003`

[4] I. Goodfellow, Y. Bengio, and A. Courville (2016), *Deep Learning*. The MIT Press.

[5] W. Wu, Z. Zhan, S. Peng, S. Ni, and J. Callies (2020), Seismic ocean thermometry. *Science* **369**(6510) 1510-1515. `https://www.science.org/doi/10.1126/science.abb9519`

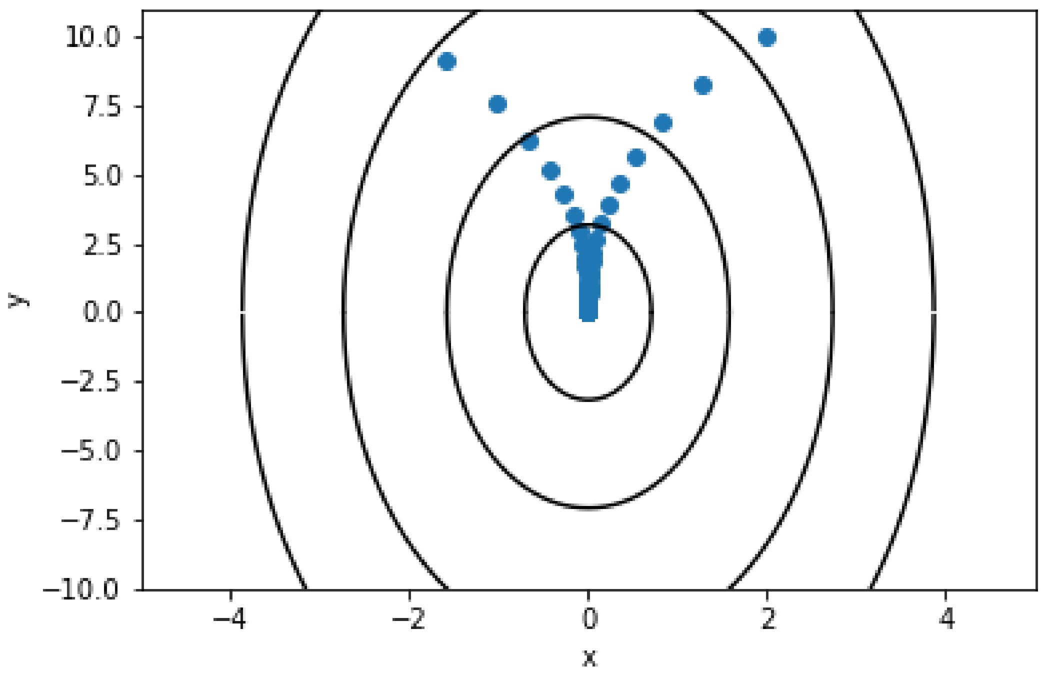

Figure 2: Gradient descent is not always efficient

I have attempted to replicated Figure 4.6 in [4]. The plot shows the descent trajectory and how it is less efficient without second order information since it bounces across the quadratic potential instead of taking a more direct path.

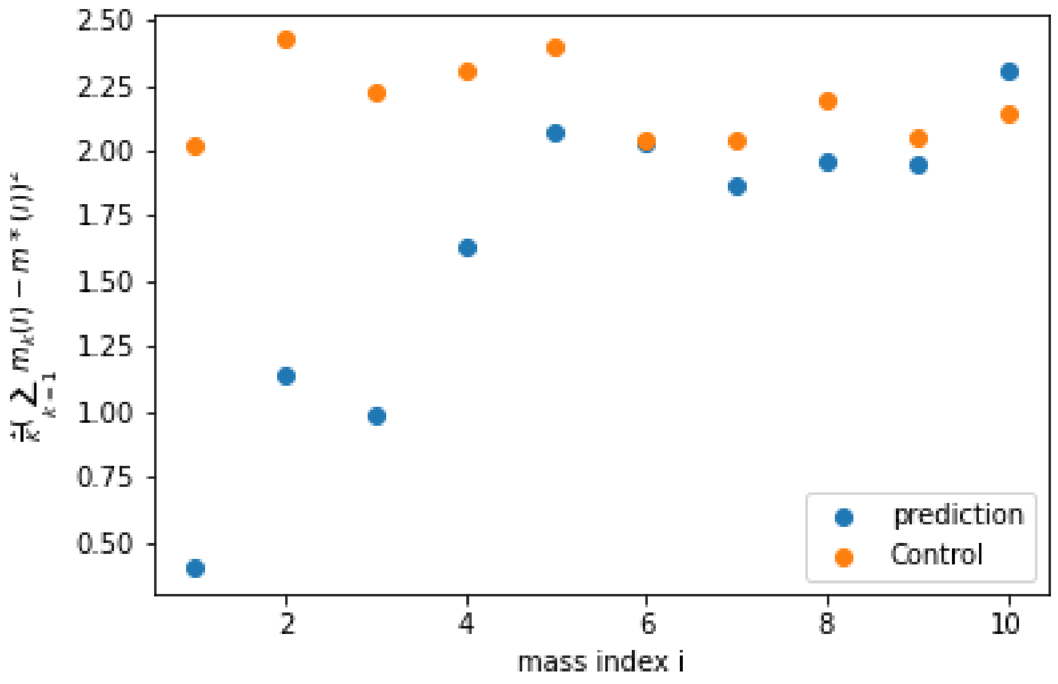

Figure 3: SVM accuracy

This plot depicts the accuracy of SVM on the testing data. SVM was trained on 1000 simulations. For the testing data with $K = 100$ samples, we have calculated $\frac{1}{K}\sqrt{\sum_{k=1}^{K}(\hat{m}_k(i) - m_k(i))^2}$ for each mass index $i$ and plotted this on the y-axis. Here, $\hat{m}$ are the predicted masses while $m$ are the true masses for that sample. The control replaces $\hat{m}$ with $11/2$ for all values, which is the mean i.e. best for $m$ with no information.

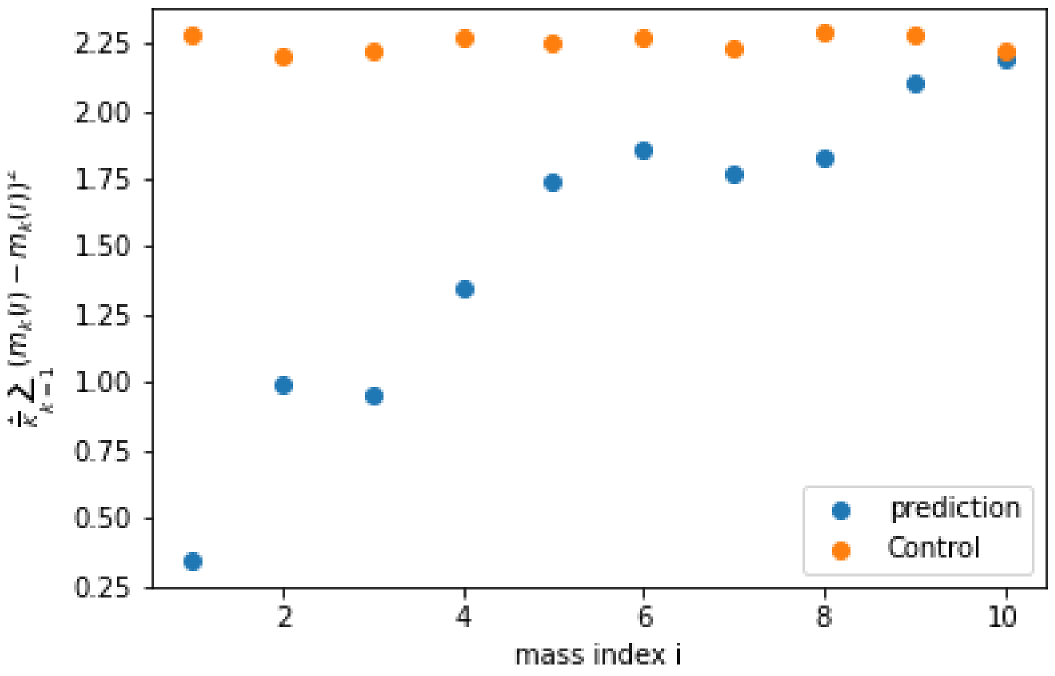

Figure 4: Ridge accuracy

This plot depicts the accuracy of Ridge regressor on the testing data. The setup is the same as in Figure 3, but we could train and test the Ridge regressor on more data because it was faster than SVM. That is why the control looks smoother, and it also explains to some degree why the Ridge regressor performs better.

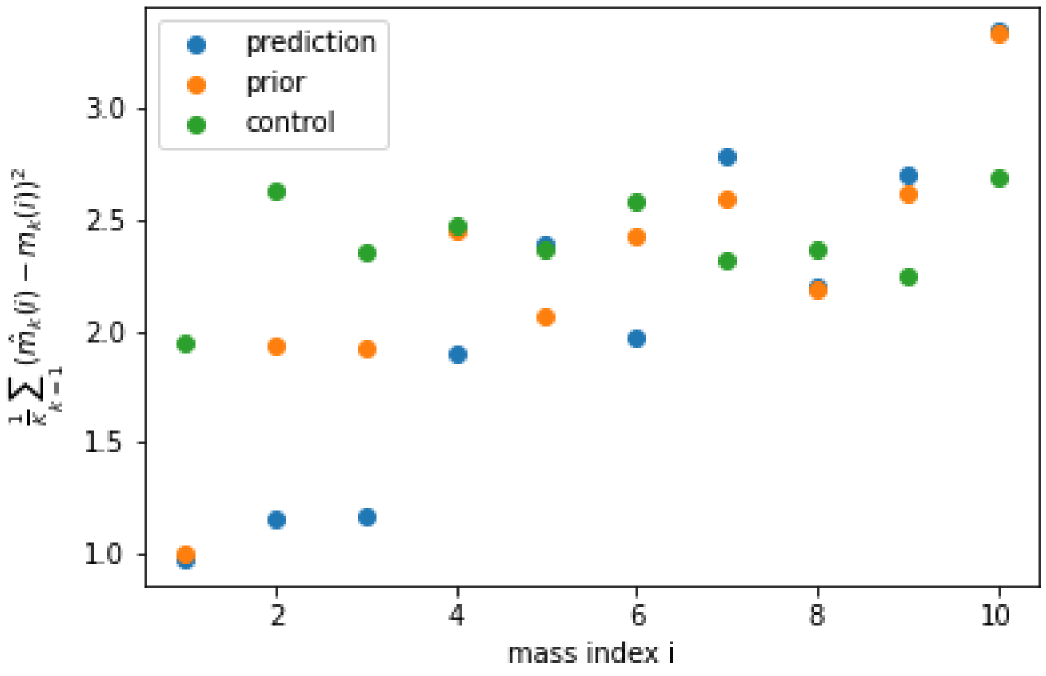

Figure 5: Gradient descent Accuracy

This plot depicts the accuracy of Ridge regressor followed by gradient descent on 10 validation samples. The y-axis is the same as that in Figure 3 and Figure 4. The ridge regressor (prior) does not quite perform as well has it had done, probably partly because we are using new validation data and partly due to chance. What is noteworthy is that the gradient descent (prediction) does improve the accuracy of the prediction for some of the masses.

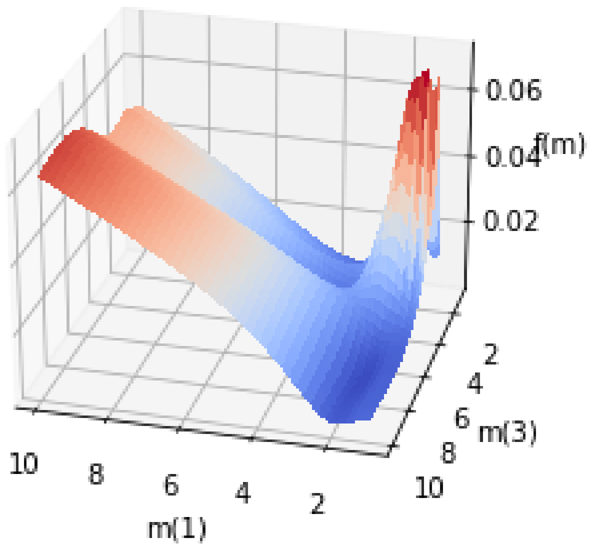

Figure 6: The objective function contains multiple local minimum

Here, we can see the objective function, $f$, plotted against the masses $m(1)$ and $m(3)$. We can see that their are probably at least two local minimums even just restricted to these two dimensions. In order for gradient descent to work, we need the starting positions to be within the correct watershed so to speak. This is why using the values predicted by the regression model is useful. In this case, the correct values are approximately $m(2) = 2$ and $m(3) = 8$.