02610 Optimization and Data Fitting
Assignment 2

Ramiro Mata

December 10 2016

1 Data Fitting with Linear Models

The set of data points we are given in this problem can be described by the following linear data fitting model:

$$y_i = \alpha t_i + \beta + e_i \tag{1}$$
$$e_i \sim N(0, \sigma^2)$$

Where e_i corresponds to the residual associated with the data point i. We shall write equation 1 in matrix form:

$$\mathbf{b} = \mathbf{A}\mathbf{x} + \mathbf{r} \tag{2}$$

And

$$\mathbf{A} = \begin{bmatrix} \mathbf{1} & \mathbf{t} \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \quad \mathbf{b} = \mathbf{y}$$

Where $\mathbf{t}$ and $\mathbf{b}$ are vectors containing the coordinates of the data points, $\mathbf{r}$ is the vector of residuals, and $\mathbf{1}$ is a vector of ones which is the same length as $\mathbf{t}$.

We are going to study four different ways of estimating the parameters α and β of our approximating model, and analyze their performance in the presence of outliers. When applicable, the objective function will be reformulated in a way that allows us to solve the function using MATLAB optimization functions linprog and quadprog (code snippets can be found in Appendix C).

Finally, statistical analyses are provided throughout to compare the different methods and their respective advantages/ disadvantages. Mathematical formulas implemented for our statistical results are included in Appendix A.

1.1 Introduction

Figure 1 shows the measured data (from Problem1Data.mat) along with the true model. Three data points deviate substantially from the rest of the data and will be regarded as outliers. These outliers could result from errors in the measuring device that records the data, or could be considered data points coming from a different statistical distribution. In this specific data set, the outlier case is obvious. However, there are many instances where classifying outliers is less clear-cut decision and therefore more uncertain from the modeller's perspective. Because of this, ideally we would want an approximating model that can incorporate outlier data without negatively affecting its performance. This would lead to a robust estimator or, said differently, a robust regressor. With this goal in mind, the outliers in the plot below will be included in our models. To avoid statistical biases however, the outliers will be discarded during our statistical tests (e.g. confidence intervals, std. deviation of the noise, etc).

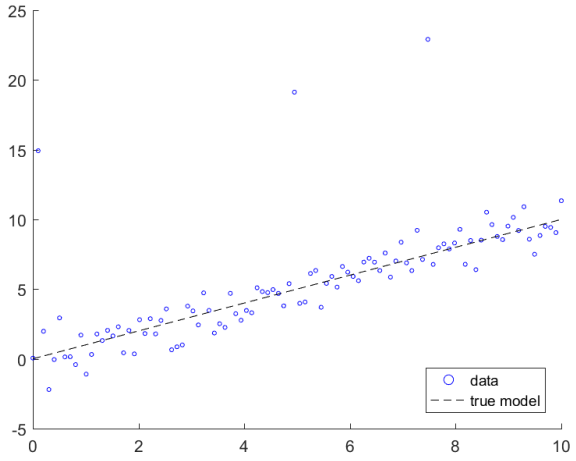

Figure 1: Scatter plot of the data

1.2 Least Squares Estimation

The least squares estimation is by far the most common method used for determining the parameters of a data fitting model. The objective function is solved, as the name implies, by finding the parameters that minimize the sum of the squared residuals. The $\frac{1}{2}$ is introduced for mathematical computational ease and does not affect the parameter estimates.

$$\min_{x \in \mathbb{R}^n} F(x) = \frac{1}{2} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 \tag{3}$$

Equation 3 can be rewritten as:

$$\begin{aligned} F(x) &= \frac{1}{2}\|\mathbf{Ax}-\mathbf{b}\|_2^2 = \frac{1}{2}(\mathbf{Ax}-\mathbf{b})'(\mathbf{Ax}-\mathbf{b}) \\ &= \frac{1}{2}\mathbf{x}'\mathbf{A}'\mathbf{Ax} + (-\mathbf{A}'\mathbf{b})'\mathbf{x} + \frac{1}{2}\mathbf{b}'\mathbf{b} \\ &= \frac{1}{2}\mathbf{x}'\mathbf{Hx} + \mathbf{g}'\mathbf{x} + \gamma \end{aligned} \tag{4}$$

Where:

$$\mathbf{H} = \mathbf{A}'\mathbf{A} \qquad \mathbf{g} = -\mathbf{A}'\mathbf{b} \qquad \gamma = \frac{1}{2}\mathbf{b}'\mathbf{b}$$

The minimum of the quadratic problem in equation 4 can be obtained in MATLAB using the optimization function quadprog.

Figure 2 shows the model obtained with the least squares method. Note that the outliers are incorporated in the model. The L2 model closely resembles the true model except for a minor offset in the y-intercept (β estimate). The reader is referred to table 1 in section 1.6 for further detailed results and discussion.

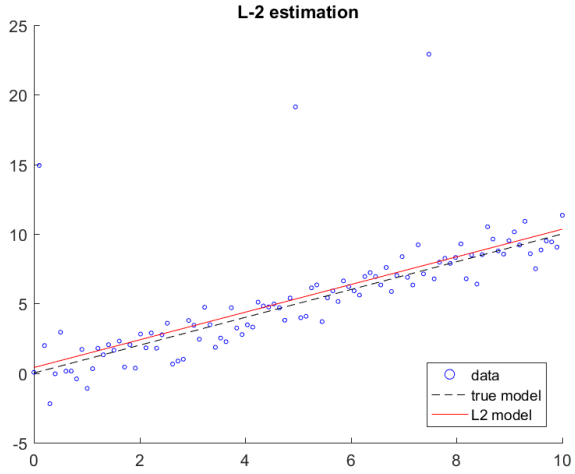

Figure 2: L_2 estimation

The upper-half of Figure 3 shows a histogram of the residuals when including the outliers; in the bottom-half histogram, the outliers have been removed resulting in the familiar shape of the gaussian distribution. As such, we consider the residuals to be "white noise." The standard deviation of the noise in the L2 model is 1.1228 compared to 1.0293, which is the true noise of the data (see Table 1).

The 95% confidence interval of both the parameters and the predictions of the L2 model are shown in Figure 4. Notice that the L2 model overestimates the data. That is, most of the data points lie beneath the model. This is due to the L2 model overestimating the β parameter presumably due to the upward influence of the three large positive outliers. The small offset in the estimated β can be seen in the figure's y-axis - recall that the true β equals 0.

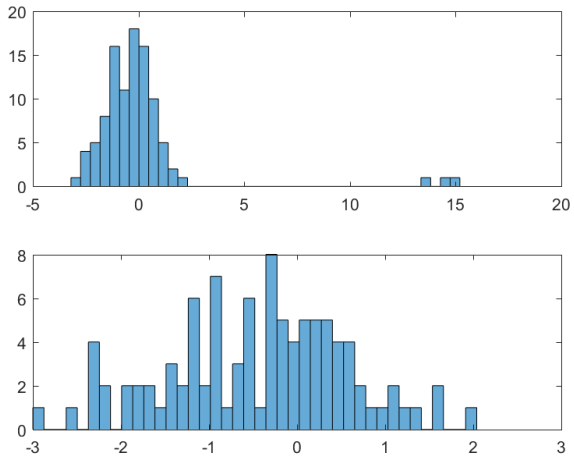

Figure 3: L_2 histogram of the errors

The room between the two green lines in figure 4 represents the confidence interval, and it is the space where 95% of the time the expected value of a prediction will lie. This should not be confused with another type of interval called prediction interval, which outlines the bounds where a single new observation will lie 95% of the time. The latter accounts for both, the uncertainty of the mean value and the variability of the data. An example of a prediction interval can be seen in figure 22 in Appendix B.

On the other hand, the brown dashed lines represent the amount of variation one could expect from the parameters if the model was estimated from a different sample population. In other words, since the estimates would not be identical depending on the sample pool they were estimated from, the parameter confidence intervals tell us how much the estimates could vary from sample to sample. That is, if the experiment was to be carried out 100 times, we would expect the parameters to lie within these intervals 95% of the time. The numerical values of the parameter estimates and their confidence intervals for each model are listed in Table 3.

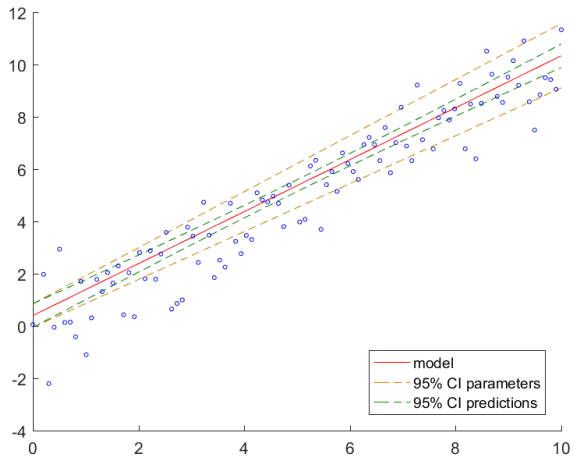

Figure 4: L_2 confidence intervals

1.3 L_1 Estimation

The L_1 regression is now obtained by minimizing the sum of the residuals' absolute value. The absolute value function, however, is not smooth. That is, the function is not differentiable at every point, thus the problem must be reformulated.

A technique that circumvents this problem involves defining a smooth function that envelopes the kinks of the absolute value function. These functions are merged resulting in a kinkless absolute value function.

$$\min_{x\in\mathbb{R}^n} F(x) = \|Ax-b\|_1 = \sum_{i=1}^{m} |Ax-b| \tag{5}$$

The above equation can be expressed as:

$$\min_{x\in\mathbb{R}^n} F(x) = \sum_{i=1}^{m} t_i \tag{6}$$

$$t_i \geq |r_i(x)| \qquad i = 1, 2, ..., m$$

As can be seen, the problem becomes a linear constrained optimization problem which can be solved using MATLAB optimization tool linprog. Before calling the function we must write the problem in matrix form and thus, equation 6 becomes:

$$F(x,t) = \sum_{i=1}^{m} t_i = \begin{bmatrix}\mathbf{0}\\ \mathbf{1}\end{bmatrix}' \begin{bmatrix}\mathbf{x}\\ \mathbf{t}\end{bmatrix} \tag{7}$$

Where $\mathbf{0}$ is a vector of zeros which is the same length as $\mathbf{x}$, and $\mathbf{1}$ is again a vector of ones of length m equal to the number of data points. The vector $\mathbf{t}$ contains m new parameters.

$$\mathbf{t} \geq |\mathbf{Ax}-\mathbf{b}| \leftrightarrow -\mathbf{t} \leq \mathbf{Ax}-\mathbf{b} \leq \mathbf{t}$$

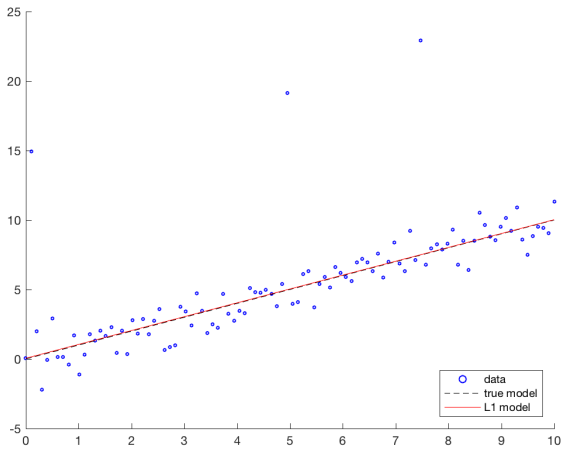

Figure 5: L_1 estimation

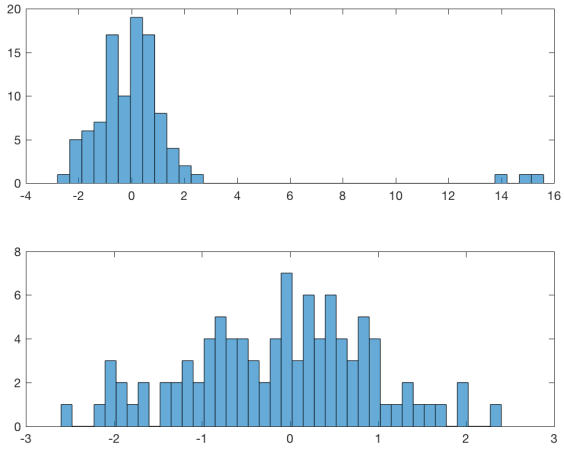

Figure 6: L_1 histogram of the residuals

We reformulate in matrix notation as:

$$\begin{bmatrix} \mathbf{A} & \mathbf{I} \\ -\mathbf{A} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{t} \end{bmatrix} \geq \begin{bmatrix} \mathbf{b} \\ -\mathbf{b} \end{bmatrix}$$

where $\mathbf{I}$ is the $m \times m$ identity matrix. Note that given how the problem is formulated, the linprog function finds estimates for both $\mathbf{x}$ and $\mathbf{t}$ vectors even though we are only interested in $\mathbf{x} = \begin{bmatrix} \alpha & \beta \end{bmatrix}'$. We discard $\mathbf{t}$ in our analyses.

Notice that the linprog function requires the inequality to have opposite signs from how we have formulated above. We can simply change the signs to both sides of the inequality and pass the arguments to the function.

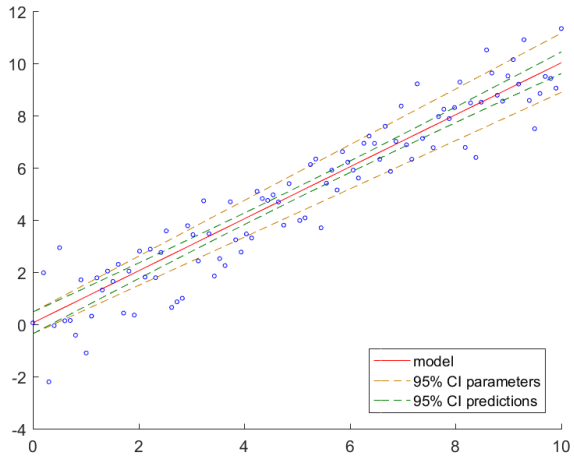

Figure 7: L_1 confidence intervals

Figure 5 shows how the L1 model performs relative to the true model. It matches it very closely and we can conclude that the L1-estimation is very robust, at least in our data sample, even in the presence of outliers.

Since the estimated parameters are approximately equal to the true parameters, the residual vector is almost identical to the pure data noise. That is why the histogram in figure 6 looks like a normal distribution and thus, the residuals, as in the L_2 case, can be considered white noise.The certainty of the predictions and the estimated parameters is shown in figure 7.

1.4 L_∞ Estimation

The L_∞ model minimizes the largest of the residuals.

$$\min_{x\in\mathbb{R}^n} F(x) = \|Ax-b\|_\infty = \max_{i\in 1,2,...,m} |Ax-b| \tag{8}$$

As in the L_1 case, the function is not smooth so we must modify equation 8 in order to compute a solution. To do this,we include a new parameter t and set bounds to our problem.

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} F(x,t) = t \\ &t \geq |r_i(x)| \qquad i = 1,2,...,m \end{aligned} \tag{9}$$

We end up with a constrained linear system that can be expressed in matrix form as:

$$F(x,t) = \sum_{i=1}^{m} t_i = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}' \begin{bmatrix} \mathbf{x} \\ t \end{bmatrix} \tag{10}$$

where $\mathbf{0}$ is again a vector of zeros of length equal to the number of data points; the number 1 in the matrix is just a single value.

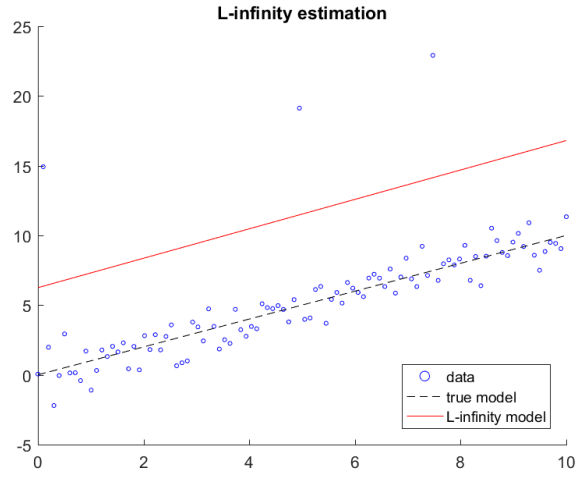

Figure 8: L_∞ estimation

From equation number 9 we see that the constant t must be larger than the largest of the residuals, and therefore we shall formulated as:

$$t[\mathbf{1}] \geq |\mathbf{Ax} - \mathbf{b}| \leftrightarrow -t[\mathbf{1}] \leq \mathbf{Ax} - \mathbf{b} \leq t[\mathbf{1}]$$

With $\mathbf{1}$ being a length m vector of ones.

$$\begin{bmatrix} \mathbf{A} & \mathbf{1} \\ -\mathbf{A} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ t \end{bmatrix} \geq \begin{bmatrix} \mathbf{b} \\ -\mathbf{b} \end{bmatrix}$$

As in L_1, if we want to use linprog we must change the sign of the inequality. The function will then yield a vector containing the parameters of our model plus the constant t. The red line in figure 8 represents our estimated model which in this case seems to be very much affected by the outliers.

Using the formulas in Appendix we computed the statistical values for the noise and the estimated parameters. We see in table 3 that the values are now significantly distant to those of the true model. We shall then conclude that the L_∞ model is not a good estimator as it fails to capture the behavior of the data.

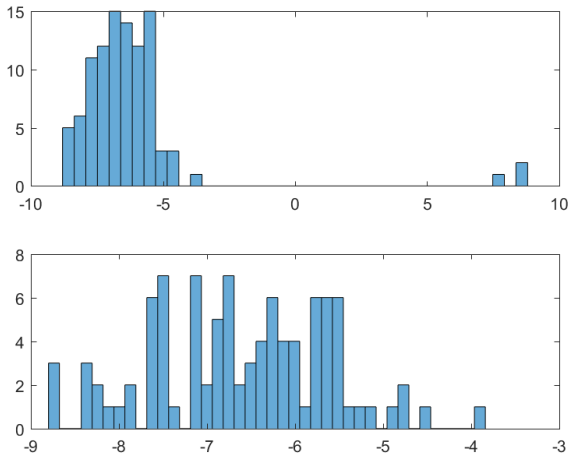

Figure 9: L_∞ histogram of the errors

The histogram displayed in figure 9 confirms this. The residuals are completely different to to those of the true model, and the standard deviation of the noise wildly differs from that of the true model. In fact, it is nearly seven times greater (see Table 1).The size of the residuals makes the model considerably uncertain. Indeed, figure 10 shows the extremely wide confidence intervals of both the parameters and the predictions.

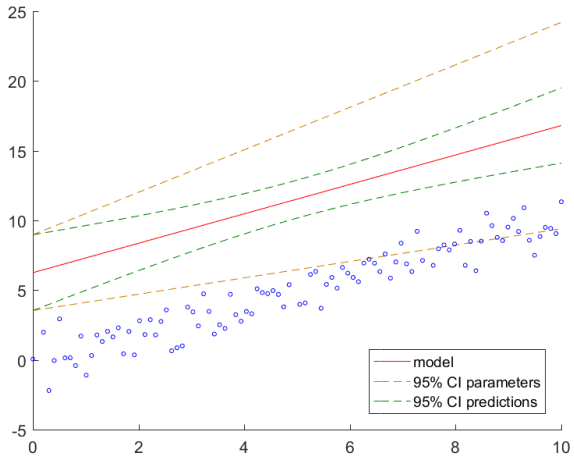

Figure 10: L_∞ confidence intervals

1.5 Huber Estimation

So far we have studied models that are fed residuals directly. The Huber method, in contrast, is fed a residual function ($\varrho(t)$) that when treating the residuals shifts between two functions depending on the size of the residual. The discriminant variable τ is specified by the user, and its value should resemble the standard deviation of the data's noise. In our case, the data's noise standard deviation is 1.0293, and so we set $\tau = 1.0293$. We define the Huber function as follows:

$$\rho(t) = \begin{cases} \frac{1}{2}t^2, & |t| \leq \tau \\ \tau|t| - \frac{1}{2}\tau^2, & |t| > \tau \end{cases} \tag{11}$$

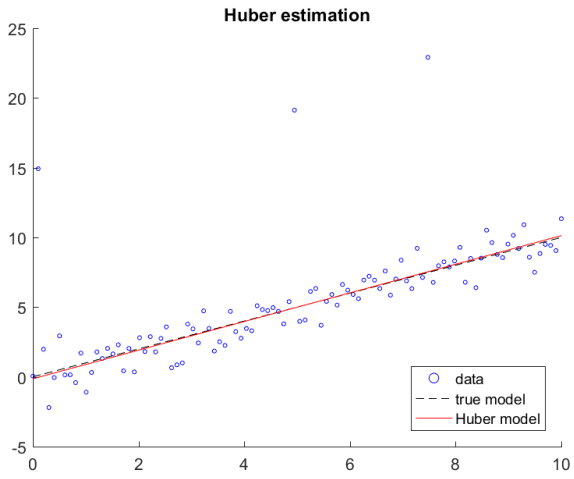

Figure 11: Huber estimation

We can write the minimization problem as:

$$\min_{x \in \mathbb{R}^n} \phi = \sum_{i=1}^{m} \rho([Ax - b]_i) \tag{12}$$

To find a solution, we have to provide three new vectors y, z and w to our set of parameters. Equation 12 takes then the form:

$$\min_{x,y,z,w} \phi = \frac{1}{2}\mathbf{w}'\mathbf{w} + \tau[\mathbf{1}](\mathbf{b}+\mathbf{z}) \tag{13}$$
$$\mathbf{w} - \mathbf{A}\mathbf{x} + \mathbf{b} - \mathbf{b} + \mathbf{z} = 0$$
$$\mathbf{b} \geq 0 \qquad \mathbf{z} \geq 0$$

Which can be transformed into a quadratic problem:

$$\frac{1}{2}\mathbf{x}'\mathbf{H}\mathbf{x} + \mathbf{g}'\mathbf{x} + \gamma \qquad \mathbf{A}\mathbf{x} = \mathbf{b} \qquad l \leq x \leq u \tag{14}$$

Where:

$$\mathbf{H} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \qquad \mathbf{g} = \tau \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{0} \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} -\mathbf{A} & -\mathbf{I} & \mathbf{I} & \mathbf{I} \end{bmatrix} \qquad \mathbf{b} = -\mathbf{b}$$

$$\begin{bmatrix} -\infty & \mathbf{0} & \mathbf{0} & -\infty \end{bmatrix}' \leq \mathbf{x} \leq \begin{bmatrix} \infty & \infty & \infty & \infty \end{bmatrix}'$$

The vectors $\mathbf{b}$, $\mathbf{b}$ and $\mathbf{z}$ are all length m (number of data points), thus, $\mathbf{0}$ are square matrices of zeros whose dimensions correspond to the length of the vectors they multiply, and I is the identity matrix of dimension $m \times m$.

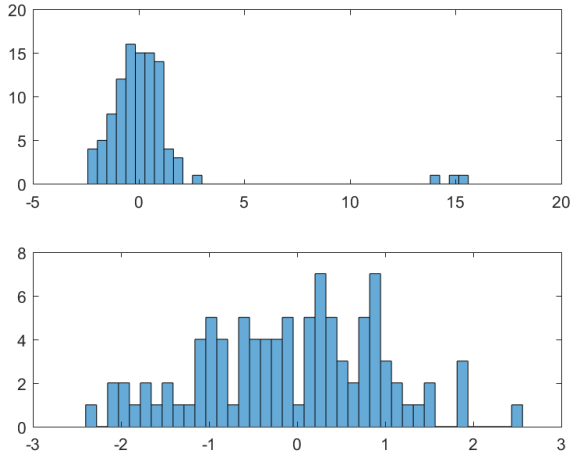

Figure 12: Huber histogram of the errors

Finally, we input these matrices to the linprog MATLAB function to obtain our parameter estimates. We observe in figure 11 that the model closely resembles the true model, and therefore is not sensitive to the three large positive outliers. From equation 12, we can

see that the Huber estimator obtains robustness by alternating between the L_2 estimator and a hybrid of the L_1.

Since the estimated parameters are very close to the true parameter values, the histogram plot of the residuals in figure 12 shows again characteristics of a normal distribution.

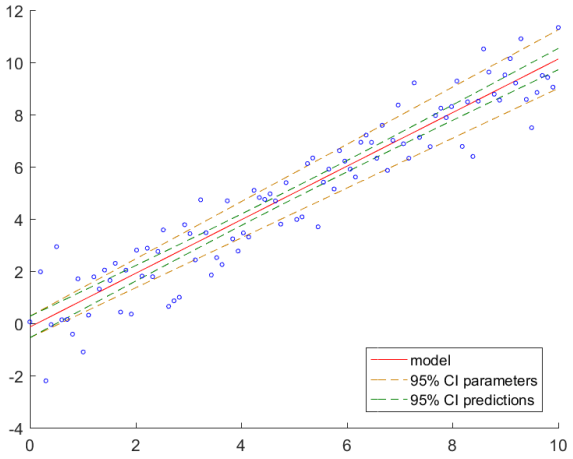

Figure 13: Huber confidence intervals

Figure 13 tells us that on aggregate the expected value of new observations would lie within that interval 95% of the time. The brown dashed lines, on the other hand, explain how much our model could vary if we used a different data sample from the same distribution.

1.6 Method Comparison

As mentioned in the introduction, a robust regression method could be described as one which incorporates potential outliers without being too sensitive to them. To that effect, we can test the quality of the methods studied in this report and assess their robustness. Table 3 lists the parameter estimates and the uncertainty of those estimates for each method.

Evidently, the best performer is L_1 followed closely by the Huber model in regards to parameter estimation. The L_1 model estimates the α and β parameters more closely than all other methods while the uncertainty of the estimates (std. dev.) is relatively similar across all methods except for the L_∞ model, which clearly is the underperformer of the group. The L_2 or LSQ method is sensitive to outliers as discussed in section 1.2.

Standard Deviation of Noise for Different Models				
True Model	L_2	L_1	L_∞	Huber
1.0293	1.1228	1.0339	6.9085	1.025

C

Table 1

Ultimately, the reason behind this is that the loss function of the LSQ ($0.5r^2$) model grows faster (than that of Huber and L_1) as the magnitude of the residuals increase (see Figure 14). Since the L_2 minimizes the sum of the squared residuals, outliers greater than one will have a bigger influence (e^2 -vs- e) than in the L_1 model. Consequently, the L_2 is

much more vulnerable to outliers as the model has to adjust to minimize these larger values. Under this light, it is not surprising to see that the L_∞ performs so poorly: since it minimizes the maximum value, it is extremely sensitive to outliers.

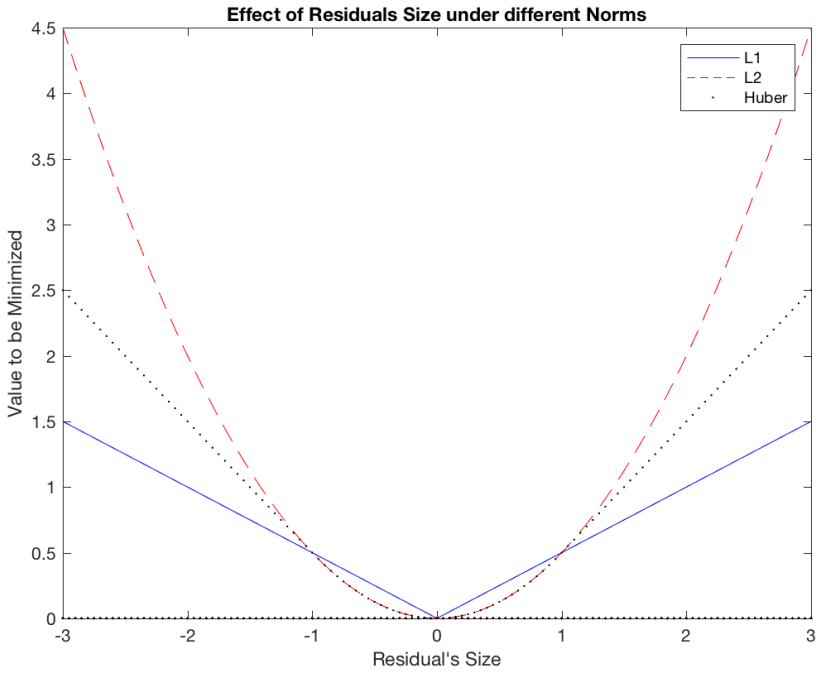

Figure 14: The effect of the residuals' magnitude on the minimization task is shown above for each of the estimators. Above, *tau* is set equal to 1.023 which mirrors the standard deviation of data's noise.

For very small residual values, both the Huber and the L_2 loss functions grow slower than the L_1 (see Figure 14). However, outside this small range as the residuals' magnitude increases, the L_2 loss function grows faster (quadratically). In contrast, the loss function of L_1 grows linearly and consequently can better handle outliers.

In terms of computational efficiency, since L_2 is differentiable and smooth, a unique analytical solution can be found. The L_1 holds no such advantage and is therefore more computationally expensive (see Table 2).

Computational Efficiency (in seconds)			
L_2	L_1	L_∞	Huber
0.0026	0.0621	0.0599	0.0141

Table 2: The elapsed time is computed as an average of 100 iterations.

The advantage of the Huber formulation is that it incorporates the smoothness of the LSQ with the robustness of L_1. By combining these two features, it is able to handle outliers (like the L_1) while also finding a unique, stable solution with computational efficiency (like the L_2). Furthermore, another benefit of Huber is that the user can specify what exactly constitutes a large or small residual (to target outliers) by manipulating the value of *tau*. Therefore, the user's knowledge of the data and the problem at hand can be leveraged by assigning an appropriate τ value, which should reflect the standard deviation of the data's noise.

In conclusion, the L_1 provides robustness but it's not computationally efficient, relatively speaking, while the L_2 has the reverse trade-off. Therefore, the L_1 estimator is better suited for applications where large outliers are likely and can be justifiably ignored, such as seismic-inverse problems. For applications where the information provided by outliers is important the L_2 estimator is more apt. Finally, the Huber method captures the advantages of both worlds.

L_2 Model						
Parameter	Estimate	Std. Dev.	Lower Bound	Upper Bound	Covariance	
					α	β
α	0.9943	0.0392	0.9178	1.0707	0.0015	
β	0.3967	0.2277	-0.0456	0.8390	-0.0077	0.0518
L_1 Model						
Parameter	Estimate	Std. Dev.	Lower bound	Upper bound	Covariance	
					α	β
α	0.9971	0.0361	0.9254	1.0688	0.0013	
β	0.0530	0.2096	-0.3632	0.4692	-0.0066	0.0439
L_∞ Model						
Parameter	Estimate	Std. Dev.	Lower bound	Upper bound	Covariance	
					α	β
α	1.0556	0.2335	0.5921	1.5191	0.0545	
β	6.2502	1.3517	3.5677	8.9327	-0.2727	1.8272
Huber Model						
Parameter	Estimate	Std. Dev.	Lower bound	Upper bound	Covariance	
					α	β
α	1.0278	0.0358	0.9567	1.0988	0.0013	
β	-0.1256	0.2078	-0.5382	0.2870	-0.0064	0.0432

Table 3: The bounds on the parameter's estimates are computed as 95% confidence intervals.

2 Enzyme Catalyzed Reaction

2.1 Linear Least Squares Estimation

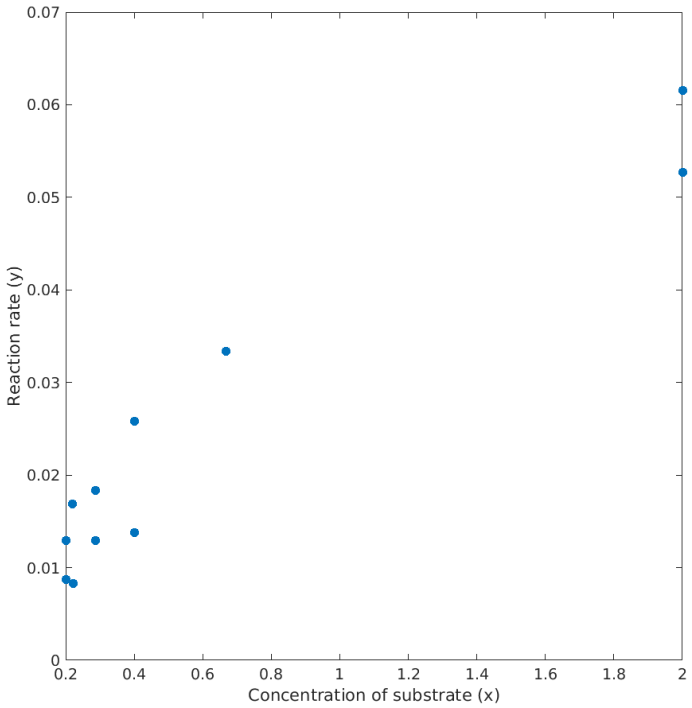

Figure 15

The given data can be seen in figure 15. In the following, Michaelis-Menten kinetics will be used as a model for the data:

$$y = \frac{\theta_1 x}{\theta 2 + x} \tag{15}$$

where y is the reaction rate and x is the concentration of the substrate. It should be noted that as

$$\lim_{x \to \infty} y = \theta_1 \tag{16}$$

and that for $x = \theta_2$

$$y = \frac{\theta_1 \theta_2}{\theta_2 + \theta_2} = \frac{\theta_1}{2} \tag{17}$$

This means that the maximum reaction rate is θ_1, and that half of the maximum reaction rate is reached for $x = \theta_2$. We use this insight to set our starting guess for the optimal θ_1 and θ_2. We do this by letting the initial θ_1 be equal to the largest y value in the observed data, and letting the initial θ_2 be equal to the x value at which half of the highest reaction rate is achieved. This puts the initial guess at

$$\theta_0 = \begin{pmatrix} 0.0615 \\ 0.6670 \end{pmatrix} \tag{18}$$

It should be expected, that these values are too low, as it does not seem from the raw data, that any plateau is reached for the highest concentration. Also following from this, it should be noted that both θ_1 and θ_2 are positive, as negative values could result in a negative reaction rate, which is not sensible as the model is based on an irreversible reaction transforming substrate to product. This is used as bounds for the algorithms, such that $\binom{0}{0} \leq \binom{\theta_1}{\theta_2}$

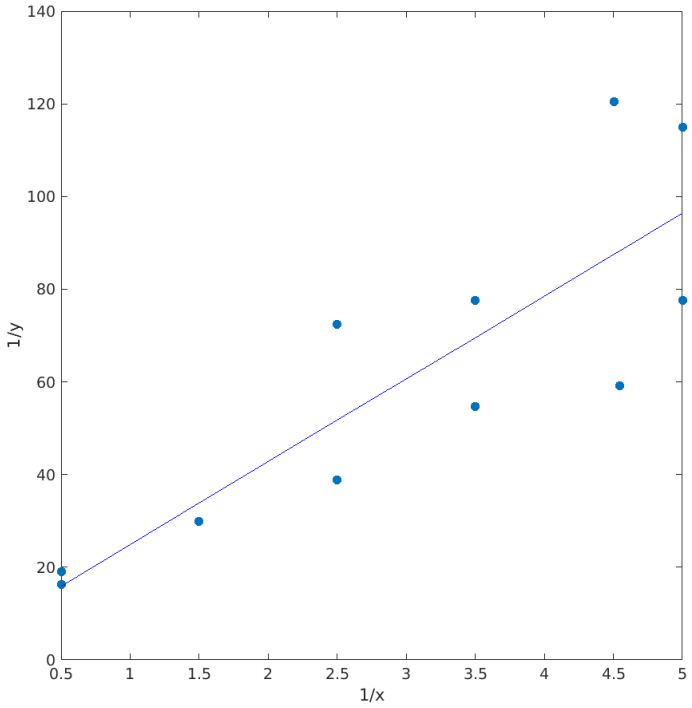

Figure 16: This plot shows the fitted linear least squares model, θ^*_{LS}, and its predictions against the observational data. Note that the axes are based on the reciprocal of the data.

Linear estimation To transform the problem into a linear least squares problem, two variables are introduced as

$$a = \frac{1}{x} \quad \text{and} \quad b = \frac{1}{y}$$

with these inserted the model can be written as

$$b = \frac{\theta_2 + x}{\theta_1 x} = \frac{\theta_2 + \frac{1}{a}}{\theta_1} a = \frac{\theta_2}{\theta_1} a + \frac{1}{\theta_1}$$

such that a line can be estimated with slope $\alpha = \frac{\theta_2}{\theta_1}$ and intercept $\beta = \frac{1}{\theta_1}$.

By letting $\mathbf{a}$ be a vector of the reciprocal of each observed substrate concentration, and $\mathbf{b}$ be a vector of the reciprocal of the observed reaction rate, the minimization problem can be written as a least squares problem

$$\min_{\mathbf{p} \in \mathbb{R}^2} \|\mathbf{Ap} - \mathbf{b}\|_2^2$$

where

$$\mathbf{A} = \begin{pmatrix} \mathbf{a} & \mathbf{1} \end{pmatrix} \quad \text{and} \quad \mathbf{p} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

This is a quadratic problem, as

$$\begin{aligned} \|\mathbf{Ap} - \mathbf{b}\|_2^2 &= (\mathbf{Ap} - \mathbf{b})^T(\mathbf{Ap} - \mathbf{b}) && (19) \\ &= \frac{1}{2}\mathbf{p}^T(2\mathbf{A}^T\mathbf{A})\mathbf{p} + (-2\mathbf{A}^T b)^T\mathbf{p} + \mathbf{b}^T\mathbf{b} && (20) \\ &= \frac{1}{2}\mathbf{p}^T\mathbf{H}\mathbf{p} + \mathbf{g}^T\mathbf{p} + \mathbf{b}^T\mathbf{b} && (21) \end{aligned}$$

This is solved in MATLAB using `quadprog` giving parameters $\alpha = 17.8738$ and $\beta = 7.0247$. The reciprocal data can be seen with the linear model in figure 16. Backtransforming the parameters gives

$$\theta_{LS}^* = \{\theta_1 = \frac{1}{\beta} = 0.1424 \quad \text{and} \quad \theta_2 = \alpha\theta_1 = 2.5444\}$$

where we denote θ_{LS}^* our linear least squares estimate. Figure 16 below shows the least squares model with the reciprocal data.

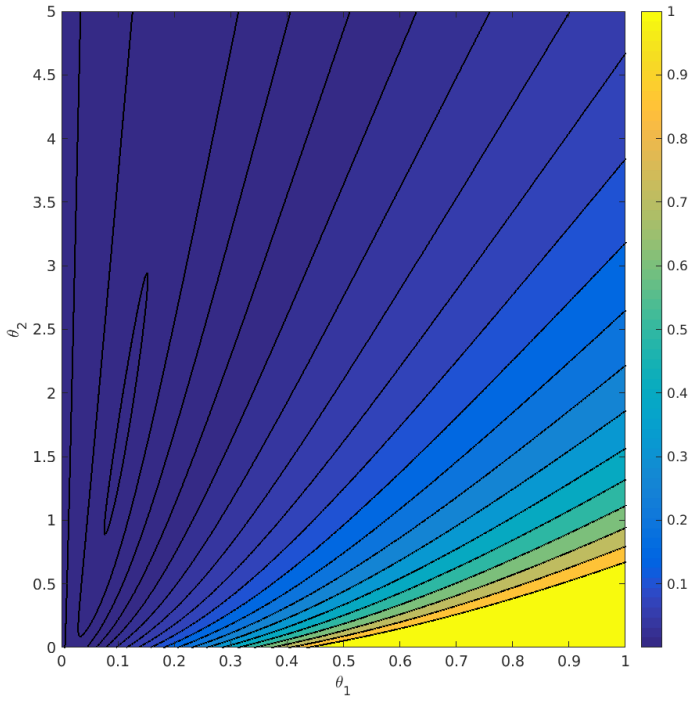

Figure 17: This plot shows the magnitude of the residuals as contours resulting from different combinations of θ_1 and θ_2. Deep blue colors represent low residual values where optimal theta estimates can be found. Note that only the positive quadrant is shown as the solution is to be positive.

2.2 Nonlinear Optimization

Nonlinear estimation When doing the transformation to a linear problem, it is still assumed that the reciprocals are normally distributed afterwards when fitting the model. If the reciprocals are in fact normally distributed before the transformation, they won't be afterwards (the reciprocal of a normal distribution will be a bimodal distribution). To avoid this, a model can be fitted without transforming it by using nonlinear optimization

$$\min_{\theta \in \mathbb{R}^2} \phi(\theta) = \frac{1}{2} \sum_{i=1}^{m} \|y_i - f(\theta, x_i)\|_2^2 \tag{22}$$

where $\theta = \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix}$ and $f(\theta, x) = \frac{\theta_1 x}{\theta_2 + x}$.

Algorithm	ϕ	Time [ms]	Iterations	Funciton calls
`quadprog`	1.348e-04	4.4	3	-
`nonlinsq`	9.539e-05	15.8	3	12
`nonlinsq` with Jacobian	9.539e-05	11.3	3	4

Table 4: The time is calculated as an average over 100 repetitions. The `quadprog` algorithm does not return a number of function calls.

A section of the contours of $\phi(\theta)$ can be seen in figure 17. Note that we only show the positive quadrant as the solution has to be positive. It is seen that the lowest values of $\phi(\theta)$ are in the area of $\theta_1 \in [0.07; 0.15]$ and $\theta_2 \in [0.9; 3]$ (the ellipsoid contour line). This is indeed a little higher than the initial estimate ($\theta_0 = \begin{pmatrix} 0.0615 \\ 0.6670 \end{pmatrix}$) discussed in the first section.

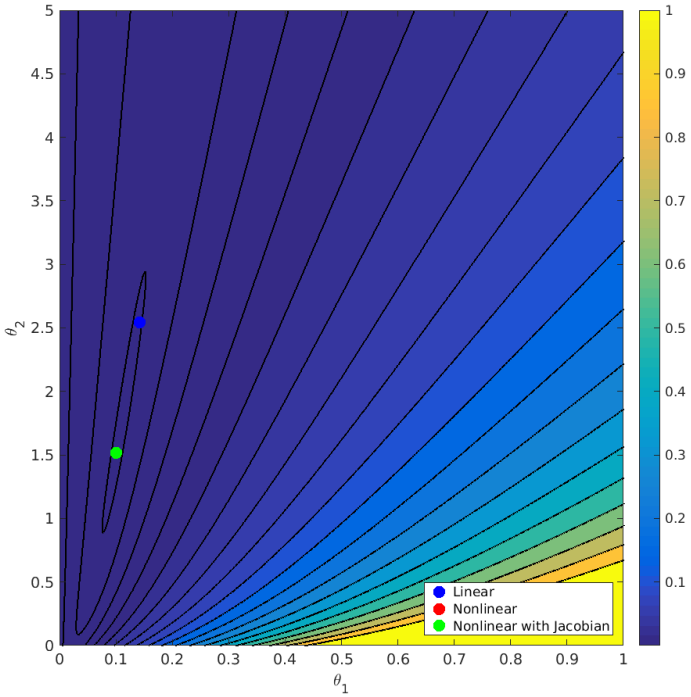

Figure 18: The estimated solutions θ^*_{LS} (blue dot) and θ^* (green dot) are shown. Note that the estimated θ^* with and without the Jacobian are identical, and therefore are overlapped in the plot - only one can be seen.

As this is a least squares problem, we solve it using the `lsqnonlin` function in MATLAB. We denote this parameter solution as θ^*. We solved it with and without supplying the Jacobian to the function to compare computational efficiency. Table 4 lists the computational efficiency results for `quadprog` (the linear case), and for `nonlinsq` (the nonlinear case) with and without the Jacobian. It is evident that in regards to speed `quadprog` (4.4 msec) is more than three times faster than `nonlinsq` wihout the Jacobian (15.8 msec). And as expected, supplying the Jacobian eases the calculation, and indeed we see a 28% reduction in computational time needed to derive the solution as well as 66% less function calls (see Table 4).

Linear estimation					
Parameter	Estimate	Lower bound	Upper bound	Covariance	
				θ_1	θ_2
θ_1	0.14	0.06	0.22	0.00	
θ_2	2.54	0.43	4.66	0.03	0.90
Nonlinear estimation					
Parameter	Estimate	Lower bound	Upper bound	Covariance	
				θ_1	θ_2
θ_1	0.10	0.07	0.13	0.00	
θ_2	1.51	0.66	2.37	0.01	0.15
Nonlinear estimation with Jacobian					
Parameter	Estimate	Lower bound	Upper bound	Covariance	
				θ_1	θ_2
θ_1	0.10	0.07	0.13	0.00	
θ_2	1.51	0.66	2.37	0.01	0.15

Table 5: The statistical characteristics of the parameter estimates of the different models are shown above. The lower and upper bounds represent the 95% confidence intervals of parameter uncertainty.

Our results are plotted in Figure 18. The three methods (θ^*_{LS}, and θ^* with and without the Jacobian) were started with the same initial guess ($\theta_0 = \left(\begin{smallmatrix}0.0615\\0.6670\end{smallmatrix}\right)$). Recall from Section 2.1 that our initial starting guess was made based on the insights from equations 16 and 17, and which was relatively close to the minimum found on the countour plot. From this starting guess, all three methods took three iterations to converge to the points illustrated in 18. Both methods converged to the ellipsis of minimum residual value. It is worth noting, however, that while the linear estimation is more computationally efficient, the nonlinear estimation results in better uncertainty bounds and have a lower sum of squared residuals value. Table 5 shows the estimated parameters and their 95% confidence intervals for both methods. The uncertainty in θ_1 is reduced 66% using nonlinear estimation relative to the linear estimation. Similarly, the parameter estimation uncertainty in θ_2 is reduced by 42%.

Figure 19 shows the linear and nonlinear models against the observational data. Both estimate similar θ_1 values of 0.14 and 0.10, respectively. This is the asymptotic reaction rate of the model for large substrate concentration. In contrast, θ_2 varies more between the two (see Table 5) and results in slightly different slopes. Consequently, if we look at the highest substrate concentrations in the plot, we notice that the linear model "misses" the last two data observations while the nonlinear model better captures these two data points. This "miss" by the linear model presumably may stem from the fact that it was fitted with the reciprocal data (Figure 16 shows a good fit in zones of high substrate concentration

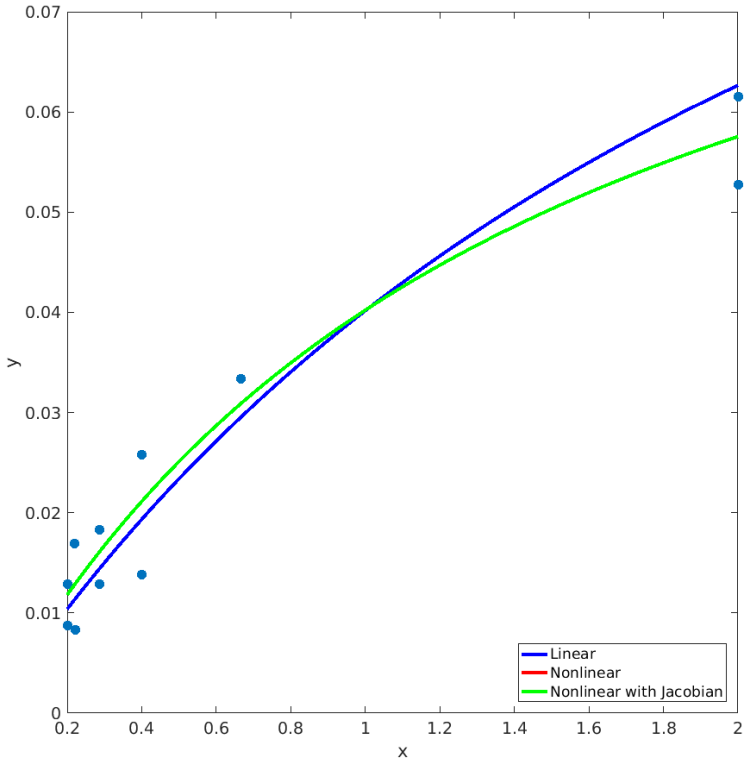

Figure 19: The linear and nonlinear fitted models and their predictions against the observational data are shown above. The x-axis represents the substrate concentration while the y-axis represents the reaction rate. Notice that the nonlinear model with and without the Jacobian are identical, and therefore one overlaps the other (i.e. only one can be seen).

with the reciprocal data). Regardless, given the parameter uncertainty involved, it is likely that the linear model overestimates reaction rates in zones of high substrate concentration.

To sum up, both methods converged to similar solutions. However, while the linear estimation was more computationally efficient using `quadprog`, the nonlinear estimation using `lsqnonlin` resulted in drastically better confidence bounds.

2.3 Time vs. Substrate concentration

Instead of having the data as substrate concentration vs. reaction rate, a Michaelis-Menten model can be estimated from time vs. substrate concentration. This is useful because in practice the reaction rate is difficult to obtain. The `ModelAndSensitivity` script provided in Lecture 12 is used with `n=1` (number of initial conditions, which is only the substrate concentration), `np=2` (number of parameters, which is θ_1 and θ_2), `xdot = -p(1)*x/(p(2)x)+` (which is the Micaelis-Menten model for the reaction rate) and a function is made for the derivatives which returns the partial derivatives of $\frac{\theta_1 x}{\theta_2 + x}$ with respect to x, θ_1 and θ_2.

The `ModelAndSensitivity` script is used for estimating the substrate concentration and the Jacobian using the `ode45` function, the residuals are found and are passed as input arguments to `lsqnonlin` along with the Jacobian to estimate the parameters θ_1 and θ_2.

We use the `lsqnonlin` algorithm because it is specifically suited for nonlinear least squares objectives. The `lsqnonlin` function has the option to choose between two numerical algo-

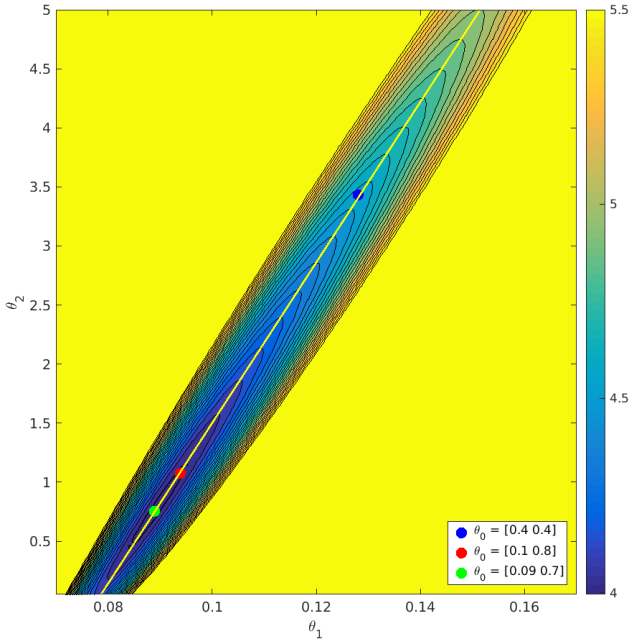

Figure 20: Solutions for Trust Region Reflective algorithm using Jacobian, with default parameters. Notice that the green and red dot converges to different solutions, even though they start close to each other.

rithms: Trust Region Reflective and Levenberg-Marquardt. The Trust Region Reflective (TRR) algorithm is better suited for problems with no constraints or with only bound constraints. In contrast, the Levenberg-Marquardt (LM) algorithm is unable to handle bound constraints and is better suited for underdetermined problems. That is, problems characterized by a system with less equations than unknown variables. Since our solution is bounded to positive values and the problem is not underdetermined, we opt for Trust Region Reflective algorithm in the `lsqnonlin` function options. For comparison purposes however, we solve using both numerical algorithms, (see Table 6 for results).

Trust Region Reflective		
θ_0	$\hat{\theta}$	$\phi(\hat{\theta})$
[0.4 0.4]	[0.128 3.43]	4.56
[0.1 0.8]	[0.0938 1.08]	4.01
[0.09 0.7]	[0.0889 0.753]	3.99
Levenberg Marquardt		
θ_0	$\hat{\theta}$	$\phi(\hat{\theta})$
[0.4 0.4]	[4.00 4.00]	128
[0.1 0.8]	[0.0940 1.09]	4.01
[0.09 0.7]	[0.0890 0.752]	4.00
Tweaked Trust Region Reflective		
θ_0	$\hat{\theta}$	$\phi(\hat{\theta})$
[0.4 0.4]	[0.128 3.43]	4.56
[0.1 0.8]	[0.0938 1.08]	4.01
[0.09 0.7]	[0.0889 0.753]	3.99

Table 6: All algorithms are using the Jacobian. The last one was run with parameters `StepTolerance = 1e-15, MaxFunctionEvaluations = 100000, MaxIterations = 10000`, defaults are `StepTolerance = 1e-6, MaxFunctionEvaluations = 200, MaxIterations = 400`

To explore how `lsqnonlin` behaves under the TRR and the LM numerical algorithms, we solved the problem using three different initial guesses. The initial guesses and where they converge using the TRR algorithm can be seen in Figure 20. We chose one initial guess that is very close to what seems to be the optimal value (green dot), one that is nearby the solution but shifted slightly along both axes (red dot), and another whose starting guess is far away from the optimal solution (blue dot).

The yellow line traverses the ellipses through the area of flattest (as opposed to steepest) descent, and was manually placed for illustrative purposes as the solutions tend to converge to it regardless of starting point (θ_0). Both TRR and LM converge to the yellow line when the starting guess is near the solution. When the starting guess is far away from the optimal solution, however, the TRR converges to the yellow line but the LM fails to do so. In fact, the LM does not get anywhere near the optimal solution nor the yellow line, and estimates parameters that deviate substantially from the true solution (see Table 6). Note that this behavior occurs under default values: `StepTolerance` $= 1e-6$, `MaxFunctionEvaluations` $= 200$, and `MaxIterations` $= 400$.

Next, we lower the `StepTolerance` to $1e-15$, and raise the `MaxFunctionEvaluations` to 100000 and the `MaxIterations` to 10000. We do this for starting guesses tangent to the yellow line to test whether they'll converge to a more optimal solution. Despite the changes in stopping criteria, they converge to the same solution obtained with the default settings. This suggests that the yellow line is not sufficiently steep, and extremely small step lengths would be necessary to improve the solution, but by a margin that would be nonconsequential.

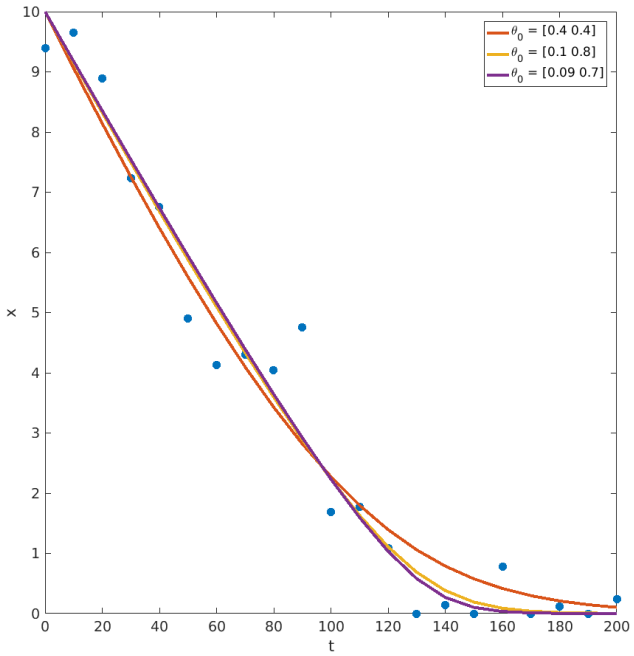

Figure 21: Solutions for Trust Region Reflective algorithm using Jacobian, with default parameters

In a nutshell, both TRR and LM converged to almost identical parameter estimates when the starting point was near the optimal solution. However, the LM algorithm failed to do so when started far away (see Table 4). Changing the step length and the stopping criteria did not make a difference in either algorithm. Our results indicate that the TRR algorithm performs better than the LM for this problem presumably because it is not

underdetermined and its solution is bounded. As a result, we will focus on `lsqnonlin` set with TRR hereafter.

In terms of the big picture, a lot of stable points (i.e. solutions) can be found for this system, which are represented by the yellow line. The starting guess "rolls down" straight onto the yellow line when finding an optimal solution. Therefore, this tells us that the value of θ^* depends largely on the initial starting guess.

Figure 21 shows the resulting model for each of the starting guesses (set with TRR) along with their predictions and the observational data. It is evident that despite different starting guesses, the estimated θ_1 values are very similar. On the other hand, the θ_2 estimates varied more. The effect of the latter can be seen for high values of t ($t \in 120-200$) where the model predictions become more convex (towards the data point cluster) for initial guesses that started closer to the optimal solution.

Table 7 lists the 95% confidence intervals of the parameter estimates. Notice that the confidence bounds are much narrower for the models with starting guesses closer to the optimal solution.

In conclusion, our results indicate that TRR performs better than LM for this nonlinear least squares problem. Further, we learned that the starting initial guess has an influence on the resulting parameter estimate and their uncertainty. Optimization tasks with objective functions similar to the problem analyzed in this report should therefore pay importance to the choice of numerical algorithm and starting initial guess to obtain better results.

$\theta_0 = [0.4\ 0.4]$					
Parameter	Estimate	Lower bound	Upper bound	Covariance	
				θ_1	θ_2
θ_1	0.13	0.07	0.18	0.00	
θ_2	3.43	-0.15	7.02	-0.04	2.93
$\theta_0 = [0.1\ 0.8]$					
Parameter	Estimate	Lower bound	Upper bound	Covariance	
				θ_1	θ_2
θ_1	0.09	0.07	0.12	0.00	
θ_2	1.08	-0.13	2.28	-0.01	0.33
$\theta_0 = [0.09\ 0.7]$					
Parameter	Estimate	Lower bound	Upper bound	Covariance	
				θ_1	θ_2
θ_1	0.09	0.07	0.11	0.00	
θ_2	0.75	-0.19	1.69	-0.00	0.20

Table 7: Parameter estimates, their covariance matrix, along with their 95% confidence intervals are listed above for three different starting guesses. The parameters are estimated using `lsqnonlin` set with the Trust Region Reflective numerical algorithm.

A Statistical Formulas

The following mathematical expressions have been used for the statistical analysis.

Variance of the noise

$$\hat{\sigma}^2 = \frac{1}{m-n}\mathbf{r}'\mathbf{r}$$

Where $\mathbf{r}$ is the residuals vector, and m, n are the dimensions of the matrix A.

Dispersion matrix of the parameters

$$\mathbf{\Sigma} = \sigma(\mathbf{A}'\mathbf{A})^{-\mathbf{1}}$$

And σ is the standard deviation of the noise.

Confidence interval of the parameters

$$\hat{x}_i \pm t(m-n)_{1-\alpha/2}\sigma((\mathbf{A}'\mathbf{A})^{-\mathbf{1}}{}_{ii})^{1/2}$$

Where $\hat{x}_i$ is the parameter estimation, $t(m-n)_{1-\alpha/2}$ is the t-student with $m-n$ degrees of freedom and $(\mathbf{A}'\mathbf{A})^{-\mathbf{1}}{}_{ii}$ is the corresponding value in the diagonal.

Confidence interval of the predictions

$$\hat{y}_i \pm t(m-n)_{1-\alpha/2}\sigma(\mathbf{a_i}(\mathbf{A}'\mathbf{A})^{-\mathbf{1}}_{\mathbf{ii}}\mathbf{a_i})^{1/2}$$

With $\hat{y}_i$ and $\mathbf{a_i}$ being the predicted value and the corresponding row in the matrix $\mathbf{A}$ respectively.

Prediction interval

$$\hat{y}_i \pm t(m-n)_{1-\alpha/2}\sigma(1+\mathbf{a_i}(\mathbf{A}'\mathbf{A})^{-\mathbf{1}}_{\mathbf{ii}}\mathbf{a_i})^{1/2}$$

As explained before, we see that the interval includes both the uncertainty of the expected value and the deviation of the data.

B Prediction Interval

The figure below is an example of the 95% prediction interval for the least squares estimation mentioned in section 1.2. We see that the interval is wider than the corresponding confidence interval of the predictions.

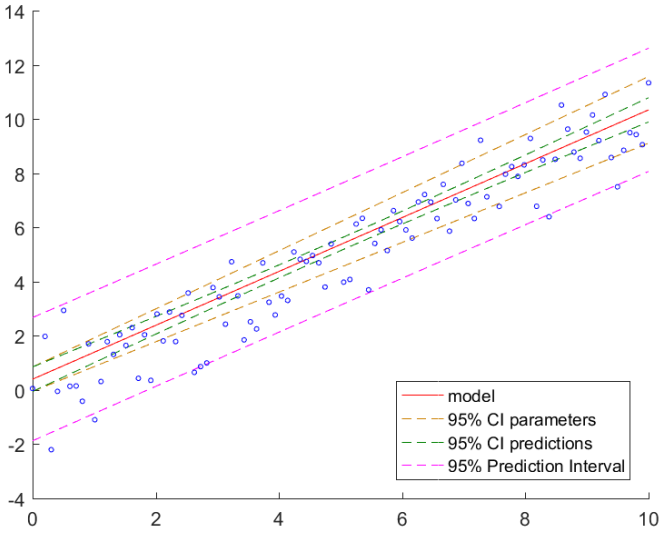

Figure 22: L_2 Prediction interval

C Code for Part 1

C.1 L1_norm.m

```
close all
load Problem1Data.mat
path(path,'../../exportfig');


%% L1 - norm

A = [t, ones(length(y),1)];
b = y;
f=[0; 0; ones(length(t),1)];


[m,n]=size(A);
I = eye(m,m);
M=[A I ;
  -A I];
B=[b;-b];

N = 100;
tic; %timing
for i = 1:N
x=linprog(f,-M,-B);
end
elapsedTime = toc/N;

figure
hold on;
x=x(1:2);
dataPlot;
plot(t,A*x,'r')
legend('data','true model','L1 model','location','southeast')
print('../img/L1_norm','-dpng')

stats(t,y,A,x,'../img/L1_hist','../img/L1_CI');
```

C.2 L2_norm.m

```
close all; clear;
load Problem1Data.mat
path(path,'../../exportfig');

%% 1.2 L2 - norm

A = [t, ones(length(y),1)];
b = y;

H = A'*A;
g = -A'*b;

N = 100;
tic; %Timing function
for i = 1:N
x=quadprog(H,g); %solving
end
elapsedTime = toc/N;

dataPlot;
hold on
plot(t,A*x,'r')
title('L-2 estimation')
legend('data','true model','L2 model','location','southeast')
hold off
print('../img/L2_norm','-dpng')

[noise_var,noise_std,x_std,x_cov] = stats(t,y,A,x,'../img/L2_hist','../img/L2_CI');
```

C.3 Linfinity_norm.m

```
close all
load Problem1Data.mat
path(path,'../../exportfig');

%% L-infinity - norm

A = [t, ones(length(y),1)];
b = y;
f=[0; 0; 1];


[m,n]=size(A);
e = ones(m,1);
M=[A e ;
  -A e];
B=[b;-b];

N = 100;
tic; %timing
for i = 1:100
x=linprog(f,-M,-B);
end
elapsedTime = toc/N;

figure
hold on;
x=x(1:2);
dataPlot;
plot(t,A*x,'r')
title('L-infinity estimation')
legend('data','true model','L-infinity model','location','southeast')
print('../img/LInf_norm','-dpng')

stats(t,y,A,x,'../img/LInf_hist','../img/LInf_CI');
```

C.4 Huber.m

```
close all
load Problem1Data.mat
path(path,'../../exportfig');


%% Huber
m=length(y);
tau = 1.0293;  %this is equivalent to the value of the data noise std dev
I = eye(m,m);
e=ones(m,1);

A = [t, ones(length(y),1)];
A = [-A -I I I];

b = y;

H = [zeros(2*m+2,2*m+2)    zeros(2*m+2,m);
     zeros(m,2*m+2)         eye(m,m)];

g = tau.*[zeros(2,1); e; e; zeros(m,1)];

%lower and upper bounds
L = [-Inf*ones(2,1);zeros(m,1);zeros(m,1);-Inf*ones(m,1)];

N = 100;
tic; %timing
for i=1:N
x=quadprog(H,g,[],[],A,-b,L); %solving
elapsedTime = toc/N;
end

A = [t, ones(length(y),1)];
x = x(1:2);
dataPlot;
hold on
plot(t,A*x,'r')
title('Huber estimation')
legend('data','true model','Huber model','location','southeast')
hold off
print('../img/Huber','-dpng')
stats(t,y,A,x,'../img/Huber_hist','../img/Huber_CI')
```

C.5 dataPlot.m

```
%% 1.1 Plot the data
scatter(t,y,6,'b')
hold on
plot(t,t,'k--')

set(gcf, 'Color', 'w');
```

C.6 stats.m

```
function [noise_var,noise_std,x_std,x_cov] = stats(t,y,A,x,filename1,filename2)

%% HISTOGRAMS

e=y-A*x; %errors
figure
subplot(2,1,1)
histogram(e,40)

A(e>5,:) = [];
y(e>5) = [];
t(e>5) = [];
e(e>5) = [];


subplot(2,1,2)
histogram(e,40)

set(gcf, 'Color', 'w');

print(filename1,'-dpng')
%% STATISTICAL ANALYSIS

m = length(y);    %Number of observations
n = length(x);    %Number of parameters

invAA = (A'*A)^-1;

noise_var   = (e'*e)/ (m - n);
noise_std = sqrt(noise_var);

x_cov = noise_var * invAA;
x_std = sqrt(diag(x_cov));

%% Confidence Intervals - PARAMETERS
alpha = 0.05; % alpha
ts = tinv(1-alpha/2, m-n); % t-student)

Upperx(1) = x(1) + (ts * noise_std * (invAA(1,1))^0.5);
Lowerx(1) = x(1) - (ts * noise_std * (invAA(1,1))^0.5);

Upperx(2) = x(2) + (ts * noise_std * (invAA(2,2))^0.5);
Lowerx(2) = x(2) - (ts * noise_std * (invAA(2,2))^0.5);

x1_CL = [Lowerx(1), Upperx(1)]; %For correct format for table function
x2_CL = [Lowerx(2), Upperx(2)];

%% Confidence Intervals - PREDICTIONS
yhat = A*x;

UpperPred = zeros(m,1);
LowerPred = zeros(m,1);
for i=1:m
    UpperPred(i) = yhat(i) - (ts * noise_std * ( A(i,:) * invAA * A(i,:)')^0.5);
    LowerPred(i) = yhat(i) + (ts * noise_std * ( A(i,:) * invAA * A(i,:)')^0.5);
end
% Prediction interval
%
```

```
% UpperPred1 = zeros(m,1);
% LowerPred1 = zeros(m,1);
% for i=1:m
%     UpperPred1(i) = yhat(i) - (ts * noise_std * ( A(i,:) * invAA * A(i,:)'+1)^0.5);
%     LowerPred1(i) = yhat(i) + (ts * noise_std * ( A(i,:) * invAA * A(i,:)'+1)^0.5);
% end


%% Plot confidence intervals

figure
hold on

h1 = plot(t,A*x,'r'); % Model

h2 = plot(t,A*Lowerx','Color',[0.8,0.5,0],'LineStyle','--'); % Confidence inteval of the parame
plot(t,A*Upperx','Color',[0.8,0.5,0],'LineStyle','--');

h3 = plot(t,LowerPred,'Color',[0,0.5,0],'LineStyle','--'); % Confidence interval of the predict
plot(t,UpperPred,'Color',[0,0.5,0],'LineStyle','--');

% h4 = plot(t,LowerPred1, 'm--'); % Prediction interval
% plot(t,UpperPred1, 'm--')

set(gcf, 'Color', 'w');

e=y-A*x;
y = y(e < 10);
t = t(e < 10);
scatter(t,y,6,'b')
legend([h1 h2 h3],'model','95% CI parameters','95% CI predictions','location','southeast')
print(filename2,'-dpng')
```

D Code for Part 2

D.1 main.m

```
clear; close all; clc;
path(path,'../../exportfig');
path(path,'..');

x = [2.0000 2.0000 0.6670 0.6670 0.4000 0.4000 0.2860 0.2860 0.2220 0.2200 0.2000 0.2000];
y = [0.0615 0.0527 0.0334 0.0334 0.0138 0.0258 0.0129 0.0183 0.0083 0.0169 0.0129 0.0087];
x = x'; y = y';
alpha = 0.05;
reps = 100;
timeL = zeros(1,reps); timeN = timeL; timeNJ = timeN;
theta0 = [0.0615 0.6670];

%% 2.1 Linear estimation of theta
% Plot raw data
figure('Position', [100, 0, 1000,1000]);
plot(x,y,'.','markersize',30);
xlabel('Concentration of substrate (x)','Fontsize',14)
ylabel('Reaction rate (y)','Fontsize',14)
set(gca,'fontsize',14);
set(gcf, 'Color', 'w');
export_fig '../img/2-1raw.png'

% Estimate model, after transforming to Linear problem
a = 1./x; b = 1./y;
A = [a, ones(length(b),1)];
H = 2*A'*A;
g = -2*A'*b;
for i = 1:reps
    T = tic;
    [est,~,~,outputL] = quadprog(H,g,[],[],[],[],[0 0],[inf inf]); %solving
    timeL(i) = toc(T);
end

% Plot reciprocal of data with model
figure('Position', [100, 0, 1000,1000]);
plot(a,b,'.','markersize',30);
hold on
plot(a,A*est,'b');
xlabel('1/x','Fontsize',14)
ylabel('1/y','Fontsize',14)
set(gca,'fontsize',14);
set(gcf, 'Color', 'w');
export_fig '../img/2-1reciprocal.png'

% Transform parameters back
thetaL(1) = 1/est(2);
thetaL(2) = est(1)*thetaL(1);
phiL = 1/2*sum((y-thetaL(1)*x./(thetaL(2) + x)).^2);

%% 2.2 Non linear estimation

%Without using jacobian
for i = 1:reps
    T = tic;
    [thetaN, resnormN, ~, ~, outputN, ~, jacobianN] = lsqnonlin(@michaelisMenten,theta0,[0 0],[
```

```
    timeN(i) = toc(T);
end

%Including Jacobian
opts = optimoptions('lsqnonlin','Jacobian','on');
for i = 1:reps
    T = tic;
    [thetaNJ, resnormNJ, ~, ~, outputNJ, ~, jacobianNJ] = lsqnonlin(@michaelisMenten,theta0,[0
    timeNJ(i) = toc(T);
end

fig = figure('Position', [100, 0, 1000,1000]);
phiContours(x,y)
export_fig '../img/2-2phi.png'
hold on
h1 = plot(thetaL(1),thetaL(2),'b.','markersize',40);
h2 = plot(thetaN(1),thetaN(2),'r.','markersize',40);
h3 = plot(thetaNJ(1),thetaNJ(2),'g.','markersize',40);
legend([h1 h2 h3],'Linear','Nonlinear','Nonlinear with Jacobian','location','southeast')
export_fig '../img/2-2phiPoints.png'

% Plot results
figure('Position', [100, 0, 1000,1000]);
plot(x,y,'.','markersize',30);
hold on
xp = linspace(min(x),max(x));
h1 = plot(xp,thetaL(1)*xp./(thetaL(2)+xp),'b','LineWidth',3);
h2 = plot(xp,thetaN(1)*xp./(thetaN(2)+xp),'r','LineWidth',3);
h3 = plot(xp,thetaNJ(1)*xp./(thetaNJ(2)+xp),'g','LineWidth',3);
xlabel('x','Fontsize',14)
ylabel('y','Fontsize',14)
set(gca,'fontsize',14);
set(gcf, 'Color', 'w');
legend([h1 h2 h3],'Linear','Nonlinear','Nonlinear with Jacobian','location','southeast')
export_fig '../img/2-2est.png'

phiN = resnormN/2; phiNJ = resnormNJ/2;
fprintf('Linear solution: \n \t phi: %.2e \n \t Average time over %d repetitions: %.4f s \n',ph
fprintf('Without Jacobian: \n \t phi: %.2e \n \t Average time over %d repetitions: %.4f s \n \t
fprintf('With Jacobian: \n \t phi: %.2e \n \t Average time over %d repetitions: %.4f s \n \t it

% Find stats and save tables
jacobianL = [-x./(thetaL(2)+x) thetaL(1)*x./(thetaL(2)+x).^2];

[covL, ci1L, ci2L] = findStats(x,y,thetaL,jacobianL,alpha);
[covN, ci1N, ci2N] = findStats(x,y,thetaN,full(jacobianN),alpha);
[covNJ, ci1NJ, ci2NJ] = findStats(x,y,thetaNJ,full(jacobianNJ),alpha);
saveStatsTable('2linear','$\theta_1$','$\theta_2$',thetaL(1),thetaL(2),ci1L,ci2L,covL);
saveStatsTable('2nonlinear','$\theta_1$','$\theta_2$',thetaN(1),thetaN(2),ci1N,ci2N,covN);
saveStatsTable('2nonlinearJacobian','$\theta_1$','$\theta_2$',thetaNJ(1),thetaNJ(2),ci1NJ,ci2NJ

% Save table comparing the methods
file = fopen('../tables/2comparison.tex','w');
fprintf(file,'\\begin{tabular}{l|cccc} \\hline \\hline \n');
fprintf(file,'Algorithm & $\\phi$ & Time [ms] & Iterations & Funciton calls  \\\\ \\hline');
fprintf(file,'\\texttt{quadprog} & %.3e & %.1f & %d & - \\\\ \n',phiL,mean(timeL)*1000,outputL.
fprintf(file,'\\texttt{nonlinsq} & %.3e & %.1f & %d & %d \\\\ \n',phiN,mean(timeN)*1000,outputN
fprintf(file,'\\texttt{nonlinsq} with Jacobian & %.3e & %.1f & %d & %d \\\\ \n',phiNJ,mean(time
fprintf(file,'\\hline \\hline \n');
fprintf(file,'\\end{tabular} \n');
fclose(file);
```

```
%% 2.3
clear; close all;
load MMBatchData.mat
opts = optimoptions('lsqnonlin','Jacobian','on');
figure('Position', [100, 0, 1000,1000]);
plot(data(:,1),data(:,2),'.','markersize',30);
hold on
theta01 = [0.4 0.4]; theta02 = [0.1 0.8]; theta03 = [0.09 0.7];

[thetaODE1, ~, ~, ~, ~, ~, jacobian1] = lsqnonlin(@residuals,theta01,[],[],opts,data);
yhat1 = estimateY(thetaODE1(1,:),data);
h1 = plot(data(:,1),yhat1,'LineWidth',3);
[thetaODE2, ~, ~, ~, ~, ~, jacobian2] = lsqnonlin(@residuals,theta02,[],[],opts,data);
yhat2 = estimateY(thetaODE2(1,:),data);
h2 = plot(data(:,1),yhat2,'LineWidth',3);
[thetaODE3, ~, ~, ~, ~, ~, jacobian3] = lsqnonlin(@residuals,theta03,[],[],opts,data);
yhat3 = estimateY(thetaODE3(1,:),data);
h3 = plot(data(:,1),yhat3,'LineWidth',3);

legend([h1 h2 h3],'\theta_0 = [0.4 0.4]','\theta_0 = [0.1 0.8]','\theta_0 = [0.09 0.7]','locati

xlabel('t','Fontsize',14)
ylabel('x','Fontsize',14)
set(gca,'fontsize',14);
set(gcf, 'Color', 'w');
export_fig '../img/2-3.png'

fprintf('trust-region-reflective \n')
fprintf('[0.4 0.4]: \n \t theta1: %.2e \n \t theta2: %.2e \n \t phi %.2e \n',thetaODE1(1),theta
fprintf('[0.1 0.8]: \n \t theta1: %.2e \n \t theta2: %.2e \n \t phi %.2e \n',thetaODE2(1),theta
fprintf('[0.09 0.7]: \n \t theta1: %.2e \n \t theta2: %.2e \n \t phi %.2e \n',thetaODE3(1),thet

fig = figure('Position', [100, 0, 1000,1000]);
phiContours2(data)
hold on
h1 = plot(thetaODE1(1),thetaODE1(2),'b.','markersize',40);
h2 = plot(thetaODE2(1),thetaODE2(2),'r.','markersize',40);
h3 = plot(thetaODE3(1),thetaODE3(2),'g.','markersize',40);
x = linspace(0.07,0.17,100); a = 68; b = -5.3;

plot(x,a*x + b,'y','LineWidth',2)
legend([h1 h2 h3],'\theta_0 = [0.4 0.4]','\theta_0 = [0.1 0.8]','\theta_0 = [0.09 0.7]','locati
axis([0.07 0.17 0.05 5])
export_fig '../img/2-3Phi.png'


% Get stats and save tables
alpha = 0.05;
[cov1, ci11, ci21] = findStats2(data,thetaODE1,full(jacobian1),alpha);
[cov2, ci12, ci22] = findStats2(data,thetaODE2,full(jacobian2),alpha);
[cov3, ci13, ci23] = findStats2(data,thetaODE3,full(jacobian3),alpha);
saveStatsTable('2-3(0.4 0.4)','$\theta_1$','$\theta_2$',thetaODE1(1),thetaODE1(2),ci11,ci21,cov
saveStatsTable('2-3(0.1 0.8)','$\theta_1$','$\theta_2$',thetaODE2(1),thetaODE2(2),ci12,ci22,cov
saveStatsTable('2-3(0.09 0.7)','$\theta_1$','$\theta_2$',thetaODE3(1),thetaODE3(2),ci13,ci23,co


% Try with different methods
opts = optimoptions('lsqnonlin','Jacobian','on','Algorithm','levenberg-marquardt');

thetaODE1 = lsqnonlin(@residuals,theta01,[],[],opts,data);
```

```
yhat1 = estimateY(thetaODE1(1,:),data);
thetaODE2 = lsqnonlin(@residuals,theta02,[],[],opts,data);
yhat2 = estimateY(thetaODE2(1,:),data);
thetaODE3 = lsqnonlin(@residuals,theta03,[],[],opts,data);
yhat3 = estimateY(thetaODE3(1,:),data);

fprintf('LM \n')
fprintf('[0.4 0.4]: \n \t theta1: %.2e \n \t theta2: %.2e \n \t phi %.2e \n',thetaODE1(1),theta
fprintf('[0.1 0.8]: \n \t theta1: %.2e \n \t theta2: %.2e \n \t phi %.2e \n',thetaODE2(1),theta
fprintf('[0.09 0.7]: \n \t theta1: %.2e \n \t theta2: %.2e \n \t phi %.2e \n',thetaODE3(1),thet

opts = optimoptions('lsqnonlin','Jacobian','on','StepTolerance',1e-15,'MaxFunctionEvaluations',

thetaODE1 = lsqnonlin(@residuals,theta01,[],[],opts,data);
yhat1 = estimateY(thetaODE1(1,:),data);
thetaODE2 = lsqnonlin(@residuals,theta02,[],[],opts,data);
yhat2 = estimateY(thetaODE2(1,:),data);
thetaODE3 = lsqnonlin(@residuals,theta03,[],[],opts,data);
yhat3 = estimateY(thetaODE3(1,:),data);

fprintf('trust-region-reflective, StepTolerance = 1e-15, MaxFunctionEvaluations = 100000, MaxIt
fprintf('[0.4 0.4]: \n \t theta1: %.2e \n \t theta2: %.2e \n \t phi %.2e \n',thetaODE1(1),theta
fprintf('[0.1 0.8]: \n \t theta1: %.2e \n \t theta2: %.2e \n \t phi %.2e \n',thetaODE2(1),theta
fprintf('[0.09 0.7]: \n \t theta1: %.2e \n \t theta2: %.2e \n \t phi %.2e \n',thetaODE3(1),thet
```

D.2 michaelisMenten.m

```
function [r, J] = michaelisMenten(theta,data)
x = data(:,1); y = data(:,2);
yhat = theta(1)*x./(theta(2) + x);
r = y - yhat;

J = zeros(length(x),length(theta));

if nargout > 1
   J(:,1) = -x./(theta(2)+x);
   J(:,2) = theta(1)*x./(theta(2)+x).^2;
end
```

D.3 phiContours.m

```
function phiContours(x,y)
% Compute the function values
x1 = linspace(0,1,1000);
x2 = linspace(0,5,1000);
[Theta1,Theta2]=meshgrid(x1,x2);

phi = zeros(size(Theta1));
for i = 1:size(Theta1,1)
    for j = 1:size(Theta1,2)
        phi(i,j) = 1/2*sum((y-Theta1(i,j)*x./(Theta2(i,j) + x)).^2);
    end
end

v = linspace(0,1^(1/3),20).^3;
contourf(Theta1,Theta2,phi,v,'linewidth',2);


xlabel('\theta_1','Fontsize',14)
ylabel('\theta_2','Fontsize',14)
set(gca,'fontsize',14);
set(gcf, 'Color', 'w');
colorbar
```

D.4 findStats.m

```
function [Cov, ci1, ci2] = findStats(x,y,theta,J,alpha)

m = length(x); n = length(theta);

t = tinv(alpha/2,m-n);
Var = 1/(m-n)*sum((y-theta(1)*x./(theta(2)+x)).^2);

H = J'*J;
C = diag(H^(-1));
Conf = t*sqrt(Var)*sqrt(C);
Cov = Var*H^(-1);

ci1 = sort([Conf(1) -Conf(1)] + theta(1));
ci2 = sort([Conf(2) -Conf(2)] + theta(2));
```

D.5 phiContours2.m

```
function phiContours2(data)
% Compute the function values
x1 = linspace(0.07,0.17,100);
x2 = linspace(0.05,5,100);
[Theta1,Theta2]=meshgrid(x1,x2);
y = data(:,2);

phi = zeros(size(Theta1));
for i = 1:size(Theta1,1)
    for j = 1:size(Theta1,2)
        yhat = estimateY([Theta1(i,j) Theta2(i,j)],data);
        phi(i,j) = 1/2*sum((y-yhat).^2);
    end
end

v = linspace(3.9^(1/3),5.5^(1/3),25).^3;
contourf(Theta1,Theta2,phi,v,'linewidth',1);

xlabel('\theta_1','Fontsize',14)
ylabel('\theta_2','Fontsize',14)
set(gca,'fontsize',14);
set(gcf, 'Color', 'w');
colorbar
```

D.6 findStats2.m

```
function [Cov, ci1, ci2] = findStats2(data,theta,J,alpha)
m = length(data); n = length(theta);

t = tinv(alpha/2,m-n);

Var = 1/(m-n)*sum((data(:,2)-estimateY(theta,data)).^2);

H = J'*J;
C = diag(H^(-1));
Conf = t*sqrt(Var)*sqrt(C);
Cov = Var*H^(-1);

ci1 = sort([Conf(1) -Conf(1)] + theta(1));
ci2 = sort([Conf(2) -Conf(2)] + theta(2));
```

D.7 residuals.m

```
function [r,J] = residuals(theta,data)
[yhat,J] = estimateY(theta,data);
r = data(:,2) - yhat;
```

D.8 estimateY.m

```
function [yhat,J] = estimateY(theta,data)
t = data(:,1);
n = 1; np = length(theta);
z0 = [10 0 0];
[T,Z] = ode45(@ModelAndSensitivity,t,z0,[],theta,n,np);
yhat = Z(:,1);
Sp = Z(:,2:3);
J = Sp;
```

D.9 ModelAndSensitivity.m

```
function zdot = ModelAndSensitivity(t,z,p,n,np)
x = z(1:n,1); % Unpack states
sp = z(n+1:end,1); % Unpack sensitivities
Sp = reshape(sp,n,np); % Sensitivities as a matrix
xdot = -p(1)*x/(p(2)+x); % Evaluate the model equations
[dfdx,dfdp] = Derivatives(t,x,p); % Evaluate the derivatives

Spdot = dfdx*Sp + dfdp;

zdot = [xdot; Spdot(:)]; % Return derivatives as a vector
```

D.10 Derivatives.m

```
function [dfdx,dfdp] = Derivatives(t,x,p)
dfdx = p(1)*p(2)/(p(2)+x).^2;
dfdp(1) = x/(p(2)+x);
dfdp(2) = p(1)*x/(p(2)+x).^2;
```