Project 2

A780 - Astrostatistics and Scientific Computing

Justin A. Kader and Robert E. Butler, III

Note: We show in this report only a subset of all plots generated; including them all would have been quite excessive. If you wish to see more specific plots, we will happily provide them.

1 Introduction

In Python we generated two data sets each with $n = 100$ data points, both having spread in Y (response variable) around a line with intercept 0 and slope m = 2.0. One data set had a constant spread σ around the line (homoscedastic) and the other data set had a varying σ around the line (heteroscedastic). We use the python function `numpy.random.normal` to do so:

```
ycon = np.random.normal(m*x,s,n)  # constant sigma
yvar = np.random.normal(m*x,sv,n) # variable sigma
```

The variable σ is stored in the vector `sv` above, and was generated as sv = np.linspace(min_s,max_s,n), where min_s, max_s are the minimum and maximum σ values (2 and 150), and n is the number of data points, 100.

The project involves fitting regression models to the data sets. For the homoscedastic data we used ordinary linear least squares to fit a linear model. For the heteroscedastic case we fit a linear model using weighted linear least squares to fit the data, and for this fit we also used statistical inference to look at the degree to which we could or could not reject the hypothesis that the data follow a slope and intercept of 0.

2 Linear Least Squares Regression versus Maximum Likelihood Method

Regression analysis is a statistical modeling technique involving several parameters and variables:

1. The unknown parameters β which represent a scalar or vector
2. The independent variables X
3. The dependent variable Y.

Regression models relate Y to a function of X and β:

$$y \approx f(X, \beta).$$

This is usually written formally as $E(Y|X) = f(X, \beta)$. The form of f must be specified and is based on knowledge about the relationship between Y and X that does not rely on the specific data in hand. If the number of data points is greater than the number of elements of the vector β, then the system is overdetermined and regression analysis techniques will generally provide tools for

In linear least-squares fitting, we find the the model that is linear in parameters β which minimizes the squared deviations (residuals) between the model and the response variable:

$$\sum_i^n \frac{[Y_i - f(x_i, a, b)]^2}{\sigma_i^2},$$

where σ_i are the measurement errors. For the case when the measured y_i are drawn from a normal distribution, the above equation is the definition of the χ^2 statistic. In other words, if the data are independent and normally distributed, then linear least squares method is identical to minimization of χ^2. If we take as an example the linear model $f(x_i, b) = 1 + bx_i$, then minimizing χ^2 with respect to b, and then solving for b, gives the best estimate for that parameter:

$$b = \frac{\sum_i^n \frac{y_i x_i}{\sigma_i^2} - \sum_i^n \frac{x_i}{\sigma_i^2}}{\sum_i^n \frac{x_i^2}{\sigma_i^2}}$$

For the maximum likelihood method, we develop a log-likelihood function and find the best estimate of a model parameter by maximizing the likelihood function, by setting the derivative of the likelihood with respect to the parameter equal to zero.

The likelihood function is written

$$L = \prod_I^n \frac{1}{\sigma_i \sqrt{2\pi}} e^{-[y_i - f(x_i, a, b)]^2 / 2\sigma_i^2}$$

If we maximize log(L) by taking its derivative with respect to b, and solving for b, we get

$$b = \frac{\sum_i^n \frac{y_i x_i}{\sigma_i^2} - \sum_i^n \frac{x_i}{\sigma_i^2}}{\sum_i^n \frac{x_i^2}{\sigma_i^2}},$$

Which is identical to what we got in the linear least squares method. Therefore, these are the same when the data are Gaussian, because then the linear least squares method is equivalent to minimizing χ^2, which is in turn gives best estimate parameters identical to those found via the maximum likelihood approach.

3 Unweighted Least Squares

In both Python and R, we created a dataset with the following properties (to reiterate what was said in the introduction):

- array size = 100
- slope of linear distribution = 2
- non-varying Gaussian $\sigma = 10$
- minimum σ for linearly varying case = 2

- maximum σ for linearly varying case = 150
- sigma in varying case defined by a linear interpolation between the min and max sigma with 100 points.

We also used the following random seeds:

- Python: 320
- R: 68

These were chosen somewhat arbitrarily, but we also attempted to use seeds that showcased the difference between unweighted and weighted fits most prominently.

For the unweighted linear least squares fit, the process was quite simple. In Python, we used the `linregress` function from the `scipy.stats` module. In R, we used the mostly analogous `lm` function, which is the most commonly used linear regression tool. In both situations, the functions output several useful/necessary parameters, including the slope and intercept of the best-fit line, hypothesis test statistics (discussed further in Section 5), and the R^2 value (also discussed in Section 5).

The results of the original unweighted fit in R for *constant sigma* is shown in Figure 1. It shows what one would expect, with a *very* high R^2 value (0.9997). The unweighted fit for *varying sigma* will be shown in Section 4 alongside the weighted fit.

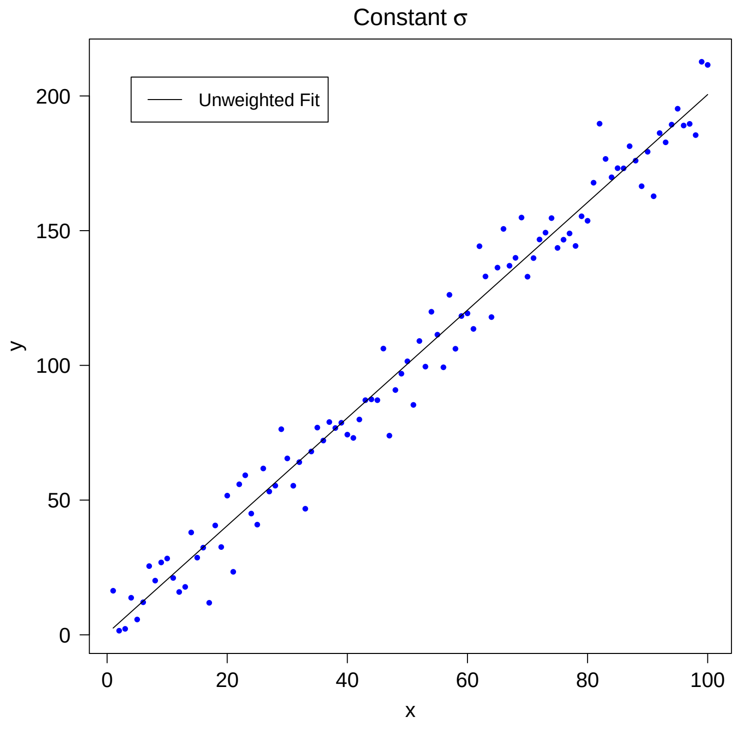

Figure 1: Unweighted fit with constant sigma in R.

4 Weighted Least Squares

The weighted least squares method uses the reciprocal of the variance of the response variable as the weight to be used in computation of the fit parameter(s). The gradient equations for the sum of squares are

$$-2\sum_i W_{ii}\frac{\partial f(x_i,\beta)}{\partial \beta_j} r_i = 0, \tag{1}$$

where each term is for the i^{th} data point, $f(x_i, \beta)$ is the model function we are fitting the data with, r_i is the residual between the fit and the response for the i^{th} data point, β_j is the j^{th} model parameter, and W_{ii} is the weight of the i^{th} data point. The weights are computed as

$$W_{ii} = \frac{1}{\sigma_i^2}. \tag{2}$$

If the response variable variances are known, i.e. if the data are generated with a known constant or varying σ, then it is a simple matter to compute the weights, we just enter the σ's into Equation (2). If, as in the general case, we do not know the values of the variance for the data, then we can use the residuals between the response variable and an ordinary least squares fit. This is illustrated in Figure 2. The steps are as follows (from Neter J, Kutner MH, Nachtsheim CJ, Wasserman W (1996) Applied linear statistical models. 4th ed. Boston: McGraw-Hill) :

- Fit the regression model with ordinary (unweighted) least squares and analyze the residuals.
- Estimate the variance function by regressing the absolute residuals on the appropriate predictors.
- Use the fitted values from the estimated variance function to obtain the weights w_i.

The OLS fit to the residuals has a value at each x_i, and we use the reciprocal of this value as the weight for our response variable at that point. We used the Python package `statsmodels.api.WLS` to compute the WLS fit to our heteroscedastic data. The result is shown in the middle panel of Figure 2 and in Figure 3.

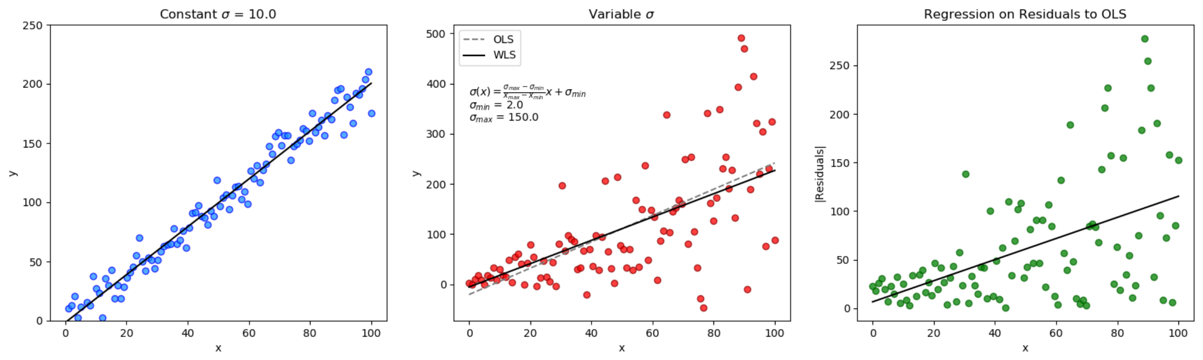

Figure 2: From left to right: randomly generated data about the line with intercept 0 and slope 2 with constant $\sigma = 10.0$ fitted with ordinary least squares, heteroscedastic data with σ varying linearly from 2 to 150 fit with OLS (dashed grey) and WLS (solid black), absolute residuals between OLS and heteroscedastic data from center panel vs. x, fit with OLS.

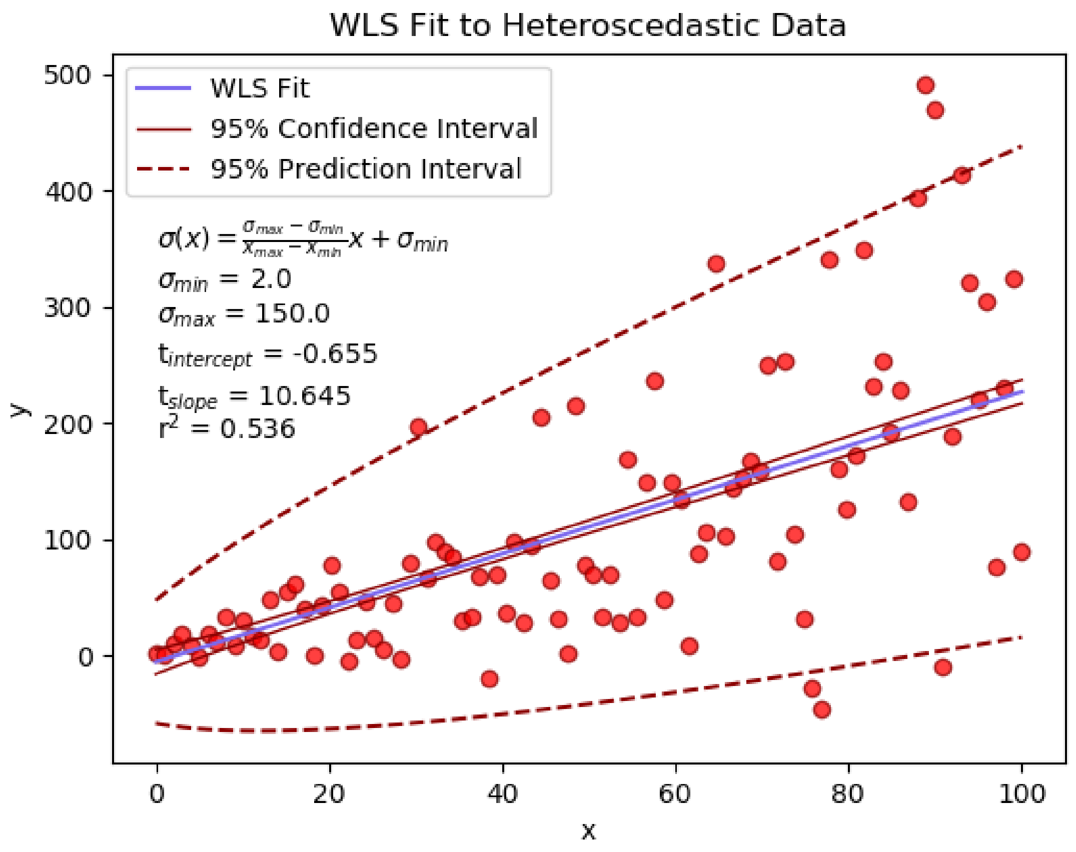

Figure 3: Weighted least squares fit to our heteroscedastic data, complete with 95% confidence and prediction intervals.

5 Inference About the Slope of the Regression Line and the R^2 Statistic

The Python package `statsmodels.api.WLS` returns relevant statistical inference about the fitted model. For each model parameter, the function `WLS` returns a Student's t-statistic, which reflects the degree to which, in our case, we can reject the (null) hypothesis that the intercept is zero or the hypothesis that the slope is zero (i.e. the bivariate data are uncorrelated). We find $t_{intercept}$ = -0.655 and t_{slope} = 10.645. Since our significance level is α = 0.05, we can clearly reject the hypothesis that the slope is zero. However, we do not reject the hypothesis that the intercept is zero, which is good, because the data were generated from a line with zero intercept. R-squared is also known as the Pearson correlation coefficient. It is defined as the covariance of the of the two variables divided by the product of their standard deviations. It can be interpreted as a measure of the correlation for bivariate data, i.e. the correlation between two variables X and Y. It ranges between -1 (for total negative correlation), and +1 (total positive correlation), where 0 means there is no correlation. We find $R^2 = 0.536$, which can be interpreted as a 54% probability that the data are positively correlated.

The R function `lm` can be easily used to make inferences about the slope of the regression line.

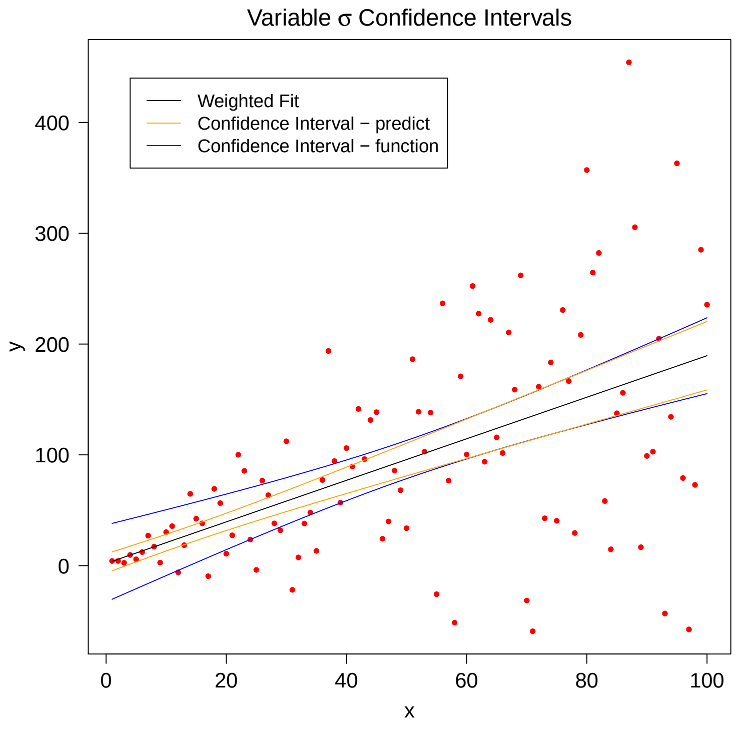

Figure 4: Varying sigma weighted fit, along with confidence intervals using two different methods in R. We will discuss these briefly in the presentation.

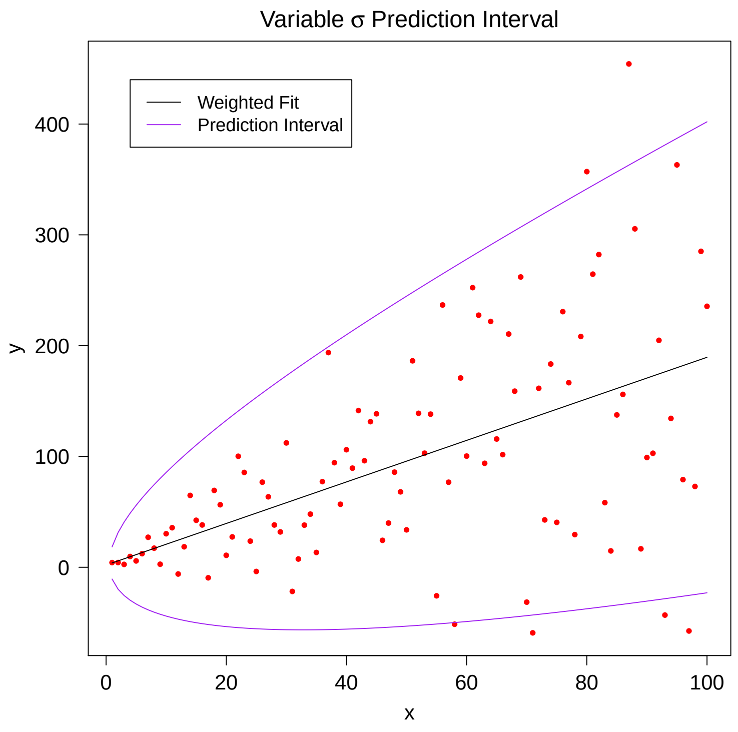

Figure 5: Varying sigma weighted fit, along with the prediction interval.

One can extract t and p values from the results for both the intercept and the slope. The null hypothesis used in the function for each of these tests is as described in the above paragraph: the parameter value is equal to zero (with a two-sided alternative). For intercepts, the p-values were usually reasonably large, never allowing us to reject the null. The p-values for the slopes, however, were incredibly small, allowing us to strongly reject the null (just like in Python, again as described above).

6 Varying Numbers of Data Points

In Python we generated 100 realizations of our heteroscedastic data, each time with a different number of data points n, where n ranged from 1 to 10,000. We plotted the Pearson r-squared statistic, the t-statistic for the intercept parameter, and the t-statistic for the slope parameter all versus n, to look for trends in the statistical nature of our fit as we vary the number of data points being fit.

We find that the Pearson r-squared statistic, which measures the level of correlation between the bivariate data generally remains constant at about $R^2 = 0.50$, especially between $n = 3000$ and $n = 8000$. Before about $n = 3000$, R^2 increases from 0.425 to 0.5, which makes sense because it is harder to tell the data are correlated if there are fewer data available to fit. The t-statistic in intercept varies somewhat with n, but stays within 0.75 of 0. Therefore, much of the time we actually would have rejected the hypothesis that the intercept is zero, regardless of the number of data. We think this is due to the fact that for heteroscedastic data generated in the way we've done it, the fits are not stable in the intercept parameter, because *small changes in slope can cause large changes in intercept.*

The t-statistic for the slope parameter increases monotonically from about 10 to 90 as we increase n from 1 to 8,000. This makes sense because our data have σ that vary in such a way that they "flare out" as they follow the linear trend, and adding more data will only continue to follow this shape, which is a shape that does not mislead a fitting algorithm from the underlying slope.

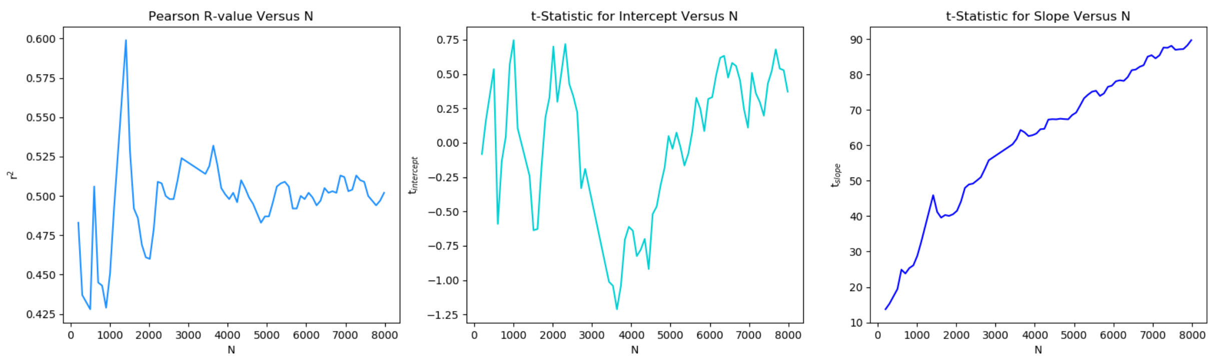

Figure 6: From left to right: Pearson r-squared versus n where we see this statistic remains fairly constant around 0.50 indicating the data are positively correlated with about a 50% probability, t-statistic in intercept versus n where we see that in a large number of cases we fail to reject the hypothesis that the intercept is zero but this is because since we chose the data to follow a line with zero intercept then any small change in slope causes large change in fitted intercept, t-statistic for slope parameter which increases monotonically with n.

7 Symmetrical Treatment of Variables in Linear Regression: bisector OLS

We used the Python package `BCES` to find the OLS fits assuming x is the dependent variable and also assuming y is the dependent variable (symmetrical treatment of variables), and the "bisector" fit which bisects these two fits, described in Isobe & Feigelson 1990.

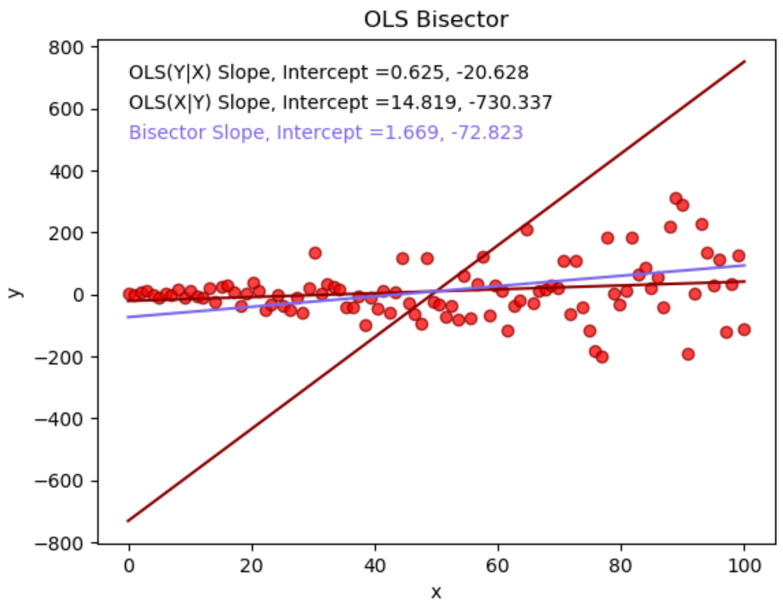

Figure 7: We used the Python package `BCES` to find the OLS fits assuming x is the dependent variabe and also assuming y is the dependent variable, and the "bisector" fit which bisects these two fits, described in Isobe & Feigelson 1990.

8 Outliers

In R we inserted an outlier in both the constant and varying sigma cases at $x = 20, y = 400$. As can be seen in the plots below, this was a pretty big outlier compared to the rest of the points. In Figure 8, we can see that the normal least-squares fit is pulled upward a fair amount compared to Figure 1, especially on the left end.

In Figure 9, we see a bit of a difference in the weighted fit as compared to Figure 4. There's also a clear kink in the confidence interval around the x-value of the outlier. The same effect can be seen between Figures 10 and 5. The prediction interval also has the kink, and is expanded quite noticeably in the case with an outlier. This isn't particularly surprising.

When testing the two cases using `rlm` in R instead of `lm`, we can see how a robust linear fit "fixes" the problem introduced by the outlier. I will not reproduce the plot here, but the fits return to essentially what they were when the outlier was not included. This is a useful tool if one needs to deal with outliers in a robust way.

We also extracted the hat matrix values from the fits in R, but they were surprisingly opaque. The

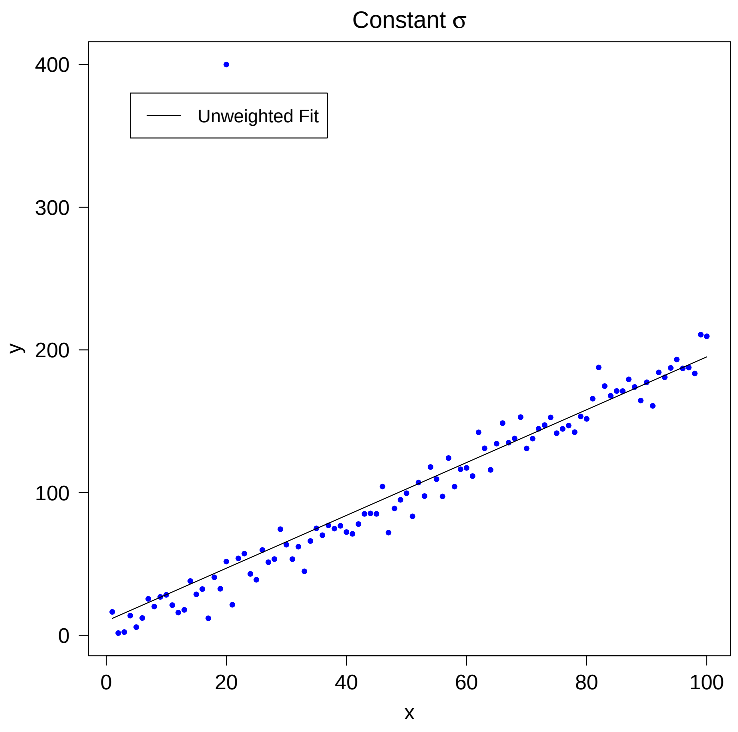

Figure 8: Constant sigma unweighted fit with outlier.

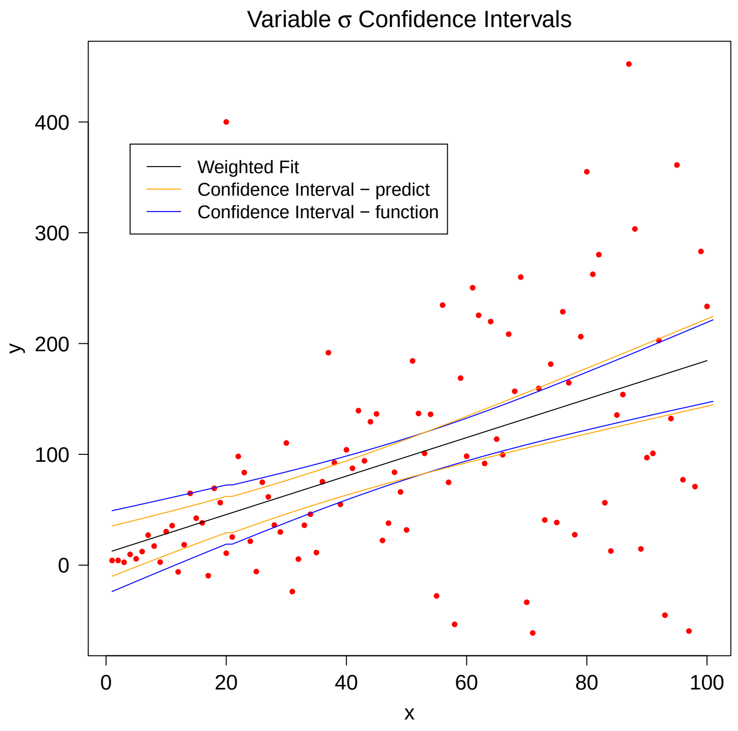

Figure 9: Varying sigma weighted fit with outlier, along with two different confidence intervals made with differing methods.

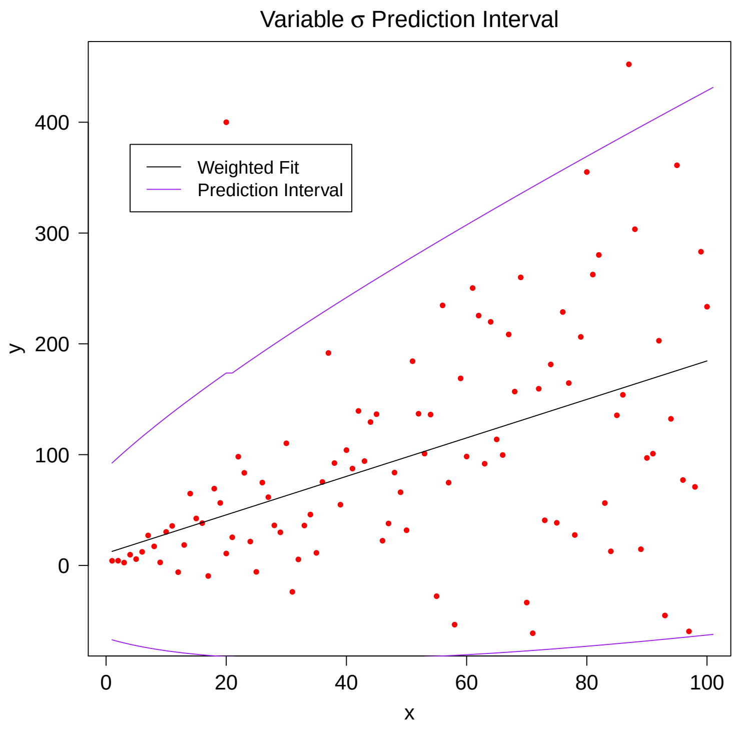

Figure 10: Varying sigma weighted fit with outlier, along with the prediction interval.

influence values for the outlier were never significantly larger than the others, and the sum of the hat matrix values were always 2. This was somewhat puzzling to us. Maybe we can discuss it in class briefly.