Weighting in Calibrations

Keith Chamberlain

23 February 2018

Abstract

The purpose of this post is to illustrate the use of weighting in calibrations. The technique shown is applicable in R, Python and Excel, even though Excel doesn't have weights in its linest() function. The regular linear regression functions are used. Where the functions have a weighting arguement (as in R's lm() function), the results are shown to be equivalent to a manual method that works across statistical softwares, regardless of whether weighting is offered by the routine. The only prerequisite is that the regression routine used be able to fit models without intercepts, as readers will be handling those manually. The reason this procedure is important is that the typical case of a calibration line includes the problem of heterogeneity of variance, which requires weighting or transformation of data to fix. The current document will introduce a dataset and illustrate the use of weighting in the bare linear model (lm()) function in R on that dataset.

Contents

Heteroscedasticity

Prior posts illustrated the generation of **surrogate calibration data** in R, Python, and Excel using R Studio (RStudio Team 2018) and Visual Studio Code, or VSCode (Microsoft 2018) and the Visual Basic for Applications (VBA) editor (Microsoft 2016) as the coding interfaces. Those data generated were pseudo random and based on a *seed*, which made the data reproducible every time the data were generated - at least within the same software. One issue that exists in **real** calibration data, which did not exist in the surrogate data, is the issue of heteroscedasticity, or heterogeneity of variance.

An assumption applied to calibration data

One of the regression assumptions is that the errors from a calibration model fit have the same distribution across the entire calibrated range (Judd, McClelland, and Ryan 2017, 38). When the variance is different, say, at the limit of detection, compared to the variance way out at the top of the curve, the estimated concentrations from back-solving the curve can become biased at points on the curve that do not fall on the mean of the instrument's response (Schwartz 1979).

The reasons for the difference in the spread of errors across the calibrated range may be systematic, such as change in the mean or variance of replicates of instrument response changing (Danzer 2007, 137), or mean or variance in replicates of concentration (Ketkar and Bzik 2000) at each level across the curve changing, or some function of these things. When the differences in variance across the calibrated range can be seen as systematic, then there are things analysts can do about it. Different types of weighting (Garden, Mitchell, and Mills 1980), and transformation are two of those things. In order to illustrate the use of weights in regression for calibration data, a data set exemplifying these issues (Chamberlain 2017) will be used.

The dataset

The calibration data exemplifying the heterogeneity of variance are well documented on Mendeley. Interested readers are encouraged to learn more about it there. We will be using the direct download link to acquire it as follows: https://data.mendeley.com/datasets/dwf4ddww3w/2/files/09a6fd07-5fee-4d48-9d7c-e099e6129f2d/AAC0215 version 2.csv?dl=1. The reproducible code in this post can be used to download the data as well. See Exhibit 1.

There are 7 concentration levels of an analyte. There are 10 replicates of each concentration, however, each individual prep was weighed separately, so that the curve has 70 independent levels. Data are concentration of the analyte and an internal standard, instrument response of the analyte and internal standard, and the weight of a reagent that was hypothesized to influence instrument response. The reagent amount did not influence instrument response

in the range tested (excluding the internal standard), however, the data are a rich toolset for illustrating heterogeneity of variance in regression and the need for weighting and transformation in calibration.

```
library(knitr)
library(ggplot2)
library(car)

# Read the dataset
dat1<-read.csv(paste("https://data.mendeley.com/datasets/",
  "dwf4ddww3w/2/files/09a6fd07-5fee-4d48-9d7c-e099e6129f2d/",
  "AAC0215 version 2.csv?dl=1", sep=""), header=TRUE)

# Exclude the outlier that was thrown in for illustrative
# purposes.
dat1<-dat1[dat1$Level!=71,]

# Add square variables for later use in weights.
dat1$AmtSqr<-dat1$Amount^2
dat1$RespSqr<-dat1$Response^2

# Pre-allocate a vector and define sum1 to contain
# the summary statistics for the calibration points
emptyVar<-vector(mode="numeric", length=7)
sum1<-data.frame(Amount=emptyVar, sdAmt=emptyVar,
                 Response=emptyVar, sdResp=emptyVar)

# Fill in sum1 with the stats that may be of interest.
for (i in 0:6){
  sum1$Amount[i+1]   =mean(dat1$Amount[(i*10+i):(i*10+10)])
  sum1$sdAmt[i+1]    =sd(dat1$Amount[(i*10+i):(i*10+10)])
  sum1$AmtSqr[i+1]   =mean(dat1$AmtSqr[(i*10+i):(i*10+10)])
  sum1$sdAmtSqr[i+1] =sd(dat1$AmtSqr[(i*10+i):(i*10+10)])
  sum1$Response[i+1] =mean(dat1$Response[(i*10+i):(i*10+10)])
  sum1$sdResp[i+1]   =sd(dat1$Response[(i*10+i):(i*10+10)])
  sum1$RespSqr[i+1]  =mean(dat1$RespSqr[(i*10+i):(i*10+10)])
  sum1$sdRespSqr[i+1]=sd(dat1$RespSqr[(i*10+i):(i*10+10)])
}
## Print the table with columns of interest. kable() requires {knitr}
kable(data.frame(Amount=sum1$Amount, sdAmt=sum1$sdAmt,
                 Response=sum1$Response, sdResp=sum1$sdResp))
```

Amount	sdAmt	Response	sdResp
0.0099829	0.0001213	397.8085	47.33064
0.0254900	0.0001392	796.6211	86.35132

Amount	sdAmt	Response	sdResp
0.0510722	0.0001971	1657.6099	89.55023
0.1306860	0.0002099	4331.4959	266.65047
0.2574007	0.0002512	8342.0455	374.58170
0.3875415	0.0016334	13556.8226	508.41172
0.6468812	0.0010471	23295.1199	1202.97315

Exhibit 1 and Table 1: Code to grab data and generate summary table.

As can be seen from the table above, the standard deviation of the replicates in instrument response, assuming the replicates can be considered as part of the same level, increase as concentration increases. The replicates for concentration do the same thing in all but the last level, where the variance seems to be tighter in level 7 than in level 6. The standard deviation of instrument response, however, indicates that the change is nearly 100 fold. As a result of this increase, readers may expect that the confidence bands for a curve fit to the data would change asymmetrically across the curve in a way that matches the observed standard deviations.

Confidence bands

See figure 1 below for a plot of the calibration data with 99.999% confidence bands superimposed. This percentage was chosen due to the bands being difficult to detect visually otherwise.

```
# Plot the data
fig1<-ggplot() +
  geom_point(aes(x=Amount, y=Response), data=dat1) +
  stat_smooth(aes(y=Response, x=Amount), data=dat1,
              method="lm", formula=y~poly(x,2),
              level=0.99999, colour="blue") +
  ggtitle("Untransformed Calibraiton")+
  xlab("Amount (mg)") +
  ylab("Instrument Response (arbitrary units)")
fig1
```

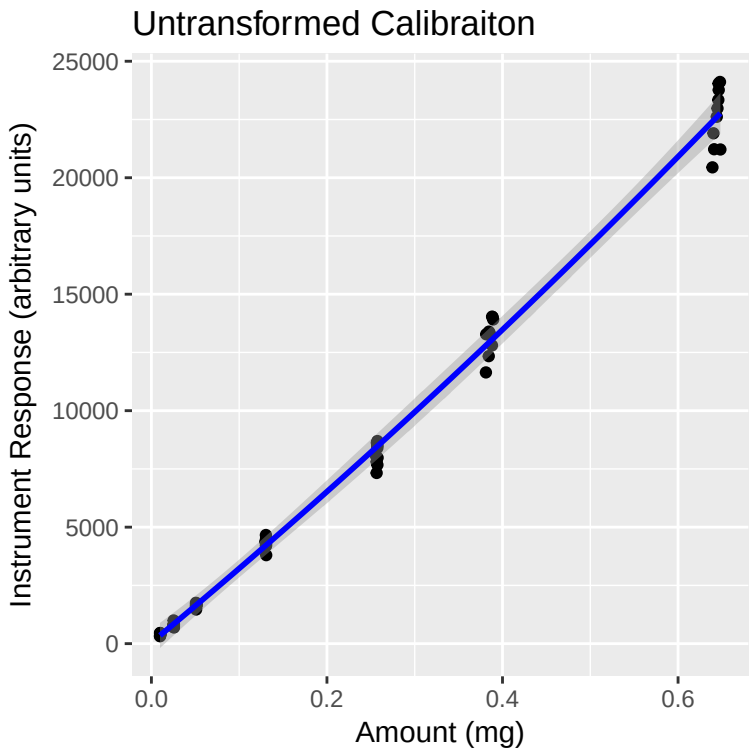

Figure 1: Calibration data with 99.999% confidence interval shaded in.

At high concentrations, the instrument responses are almost entirely outside of the confidence band. Even with a 99.999% confidence interval, at least six, that is, more than half of the ten data-points for the highest two levels, fall outside of the band. At trace levels, and to the third level, the instrument responses fall entirely within the confidence band. Even with a 90.0% confidence interval, two of the lowest three concentration levels still fall entirely within the band. Where a horn shaped band was expected, so as to match the spread in the data, a relatively flat ribbon was found instead, engulfing the lower levels, and hardly describing the top of the curve. A confidence band is more evident in a residuals plot.

Residuals

```
# Fit a calibration model
unweightedLinearFit<-lm(Response~Amount+AmtSqr,
                        data=dat1)

# Plot the residuals
fig2<-ggplot()+
  geom_point(aes(x=predict(unweightedLinearFit),
                 y=resid(unweightedLinearFit)))+
  stat_smooth(aes(x=predict(unweightedLinearFit),
                  y=resid(unweightedLinearFit)),
```

```
              method="lm", formula=y~x,
              level=0.99999) +
  ggtitle("Residuals Plot for Unweighted Fit")+
  xlab("Predicted")+ylab("Residuals")

fig2
```

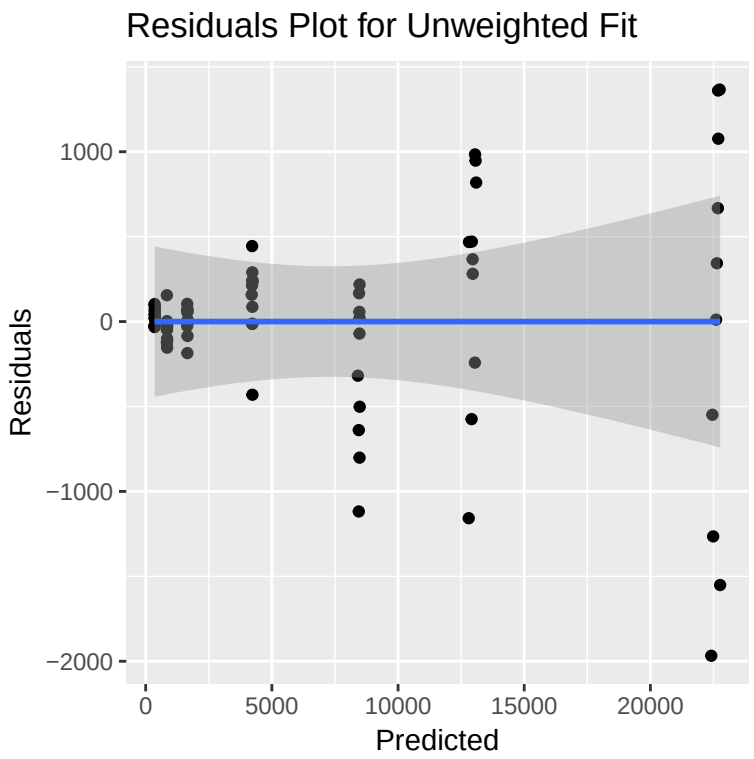

Figure 2: Residual plot from an unweighted quadratic fit.

The disparity between the confidence bands at low and high concentrations is more evident in figure 2. The actual data fan out far greater at higher concentrations than are predicted by the model, meaning an Analyst would have a tougher time than expected passing system suitability criteria at the sixth and especially 7th level of the curve. This is precisely the location where many analytical chemists choose to test their curves! The "best fit" part of the curve, where the confidence bands are thinnest, is actually between the fourth and fifth level in these data.

Quantile-Quantile Plots

The Q-Q plot can be used to assess whether the residuals from a fitted model meet the assumption that they come from a particular distribution. A Q-Q plot for the calibration data whose residuals are graphed above is shown next, using the **qqPlot()** function from the *{car}* package (Fox and Weisberg 2011) in R.

```
# qqPlot from package {car}
qqPlot(unweightedLinearFit, distribution = "t",
       main = "Q-Q Plot Unweighted Fit",
       ylab = "Studentized residuals",
       line = "quartiles", cex = 0.75,
       simulate = FALSE, envelope=0.99999)
```

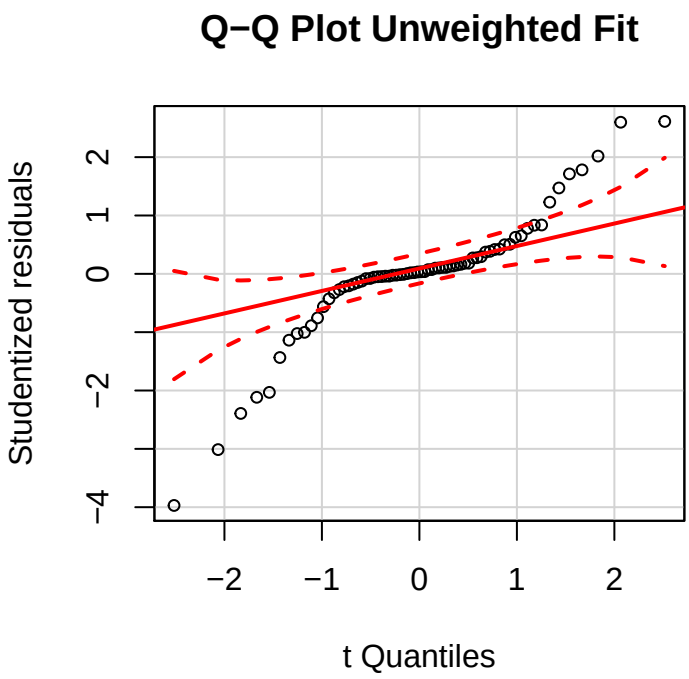

Figure 3: Quantile-quantile plot of the calibration data.

In figure 3, it is clear that the residuals follow a straight line only in the center of the plot - between [-1,1] quantiles. Many datum (circles) lay outside of the 99.999% confidence bands (dashed) outside of that region, even when the distribution assumed for the plot is a t-distribution. There are 12 outliers in the lower quartiles, and 7 in the upper quartiles. *A full 27% of the residuals are outliers from the unweighted model!* There is an option for a "Robust" fit with the **rml()** function in package *{MASS}* (Venables and Ripley 2002) which should take better care of the tails.

```
qqPlot(unweightedLinearFit, distribution="t",
       main = "Q-Q Plot Robust Fit",
       ylab = "Studentized residuals",
       line = "robust", , cex = 0.75,
       simulate = FALSE, envelope=0.9995)
```

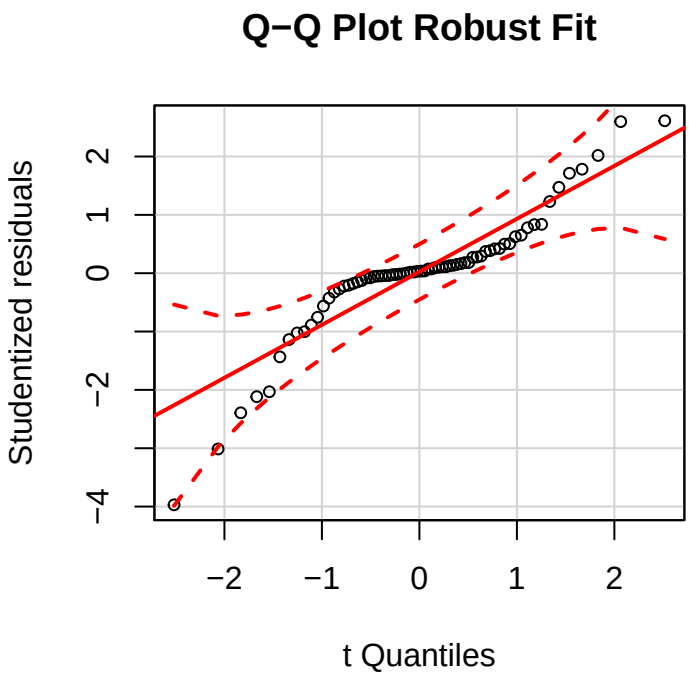

Figure 4: Robust quantile-quantile plot.

A "robust" line for fitting the distribution in a quantile-quantile plot can get all of the points roughly within the confidence bands, although a 99.95% confidence band is needed (dashed). There still seems to be some systematic trends happening at the tails that need explanation, and the residuals at the center of the line now have an s-shape along the line, suggesting that this (the un-weighted/un-transformed linear model) probably would not be the final model. Perhaps with weighting and/or transformation the residuals might look better.

Weighting

Weighting and transformation come in many shapes and sizes. The ultimate authority on weighting and transformation is Carroll and Ruppert (1988). A few flavors of weighting, and almost no transformation approaches (minus some specialized cases), are typical of calibration data encountered in analytical chemistry. Agilent's **OpenLAB CDS** (2016, 106) supports two transformations, logarithmic and exponential, and four weightings: $\frac{min(Amount)}{Amount}$, $\frac{min(Amount^2)}{Amount^2}$, $\frac{min(Response)}{Response}$ and $\frac{min(Response^2)}{Response^2}$ out of the box. Not everyone agrees that these are good for weighting, and other estimators are suggested instead.

Danzer (2007, 137) suggests that the square of the standard deviation of instrument response ($\frac{1}{s_i^2}$) be taken as the weights. Tellinghuisen (2009) uses estimation of a variance functions by route of iteratively re-weighted least squares (IRWLS; Kutner, Nachtsheim, and Neter (2004), p. 426) and also uses the inverse of the variance, though his variance functions are more complex than the form proposed by Danzer, especially at low concentrations.

There are a couple of problems with the IRWLS approach using the variance functions: volatility in the estimates used for obtaining the weights, its cousin, 'over-fit', and non-normal error. During validation for an analytical method, companies need to land on a calibration strategy that can be reproduced time and again. If there is volatility in the estimates over time, including estimates of the variance function, it may mean that the instrument is not as stable as expected, and calibrations need to be performed more often. At the same time, it may mean the parameters are over-fit and a simpler variance function is needed. Most commercial labs do not have the time to run five replicates of each level to come up with a variance function over several occasions to verify the variance function is stable over time. Thus, it is often the case that an assumption is made that the variance function does not change.

The second problem was that of a non-normal distribution in the errors from fitting. There are only so many variables that can be included in a linear calibration model... Amount, $Amount^2$ (this author does not recommend going much higher), with or without some transform of Amount and/or Response, and/or some kind of weighting of either Amount, Response, their means (mean functions), or their variances (variance functions). If one has not obtained normal error by regressing Response on Amount, weighted or transformed, it is unlikely they will be able to do so. As a result, estimates at different points along the curve may be biased. To be quite sure, this author has never seen analytical chemists display their residuals and/or Q-Q plots after fitting a calibration model to show that the weighting had the desired effect on the residuals.

The biggest issue, however, and the reason everyone is viewing this post, is how to do the weighting using regression in the first place, especially when the software being used does not support it, as is the case for the linest() function in Excel.

The generally untold story

The place to start with weighting is in R. If needed, the code can be referenced since it is open source, and a rich user base supports R on rhelp where readers can go ask questions. This author has already done some of the homework and will share it here. The weighting supported by Agilent will be illustrated, as well as how to transform the x and y values to reproduce the model fit as if the lm() function did not have a weighting argument.

Consider the first model fit for the data exhibited in this post, `lm1`. The model was

$$Response = b_0 + b_1 * Amount + b_2 * Amount^2$$

```
summary(unweightedLinearFit)
```

```
##
## Call:
## lm(formula = Response ~ Amount + AmtSqr, data = dat1)
##
```

```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1967.76   -96.72    19.18   201.54  1365.98
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    45.36     120.65   0.376  0.70812
## Amount      31217.90    1131.81  27.582  < 2e-16 ***
## AmtSqr       5906.31    1737.83   3.399  0.00114 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 574.8 on 67 degrees of freedom
## Multiple R-squared:  0.9945, Adjusted R-squared:  0.9943
## F-statistic:  6042 on 2 and 67 DF,  p-value: < 2.2e-16
```

Exhibit 2: Regression summary from the unweighted linear regression.

Take note of the parameter estimates in table 2. The model is saying that $Response = 45.36 + 31217.90 * Amount + 5906.31 * Amount^2$. Right now, unweighted least squares is being used, which is the same as setting all of the weights to 1.

Below, some weights are defined and some corresponding models are fit so that the process of calculating weights outside of weighted regression can be illustrated. The weights are taken from the Agilent OpenLAB CDS Reference Guide (2016, 106), that describes $\frac{min(Amount)}{Amount}$ and $\frac{min(Amount^2)}{Amount^2}$ weighting.

```
attach(dat1) # So the fields in dat1 can be referenced
             # directly from the calling environment
InvAmt    <-min(Amount)/Amount
InvAmtSqr <-min(AmtSqr)/AmtSqr
detach(dat1)

weightedLinearFit<-lm(Response~Amount+AmtSqr,
                      weights=InvAmt, data=dat1)

summary(weightedLinearFit)
```

```
##
## Call:
## lm(formula = Response ~ Amount + AmtSqr, data = dat1, weights = InvAmt)
##
## Weighted Residuals:
##      Min       1Q   Median       3Q      Max
## -247.073  -49.293    1.779   60.580  164.211
##
## Coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    68.13      28.55   2.387   0.0198 *
## Amount      30874.00     705.33  43.773  < 2e-16 ***
## AmtSqr       6456.47    1355.37   4.764 1.06e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 91.71 on 67 degrees of freedom
## Multiple R-squared:  0.996,  Adjusted R-squared:  0.9958
## F-statistic:  8246 on 2 and 67 DF,  p-value: < 2.2e-16
```

Exhibit 3: Regression summary from the weighted linear regression. Weights were supplied as an argument to lm().

With inverse Amount weighting, now the model is saying that $Response = 68.13 + 30874.00 * Amount + 6456.47 * Amount^2$, with the largest difference going to the squared term. The squared term effects the curvature of the model, and here the effect is even larger than it was in the unweighted regression. The weights are also accessible:

```
weights(weightedLinearFit)
```

```
##  [1] 1.00000000 0.99745521 0.99502437 0.98274997 0.98039020 0.97560733
##  [7] 0.97560733 0.97560733 0.96847203 0.96618024 0.38758801 0.38683826
## [13] 0.38647210 0.38498409 0.38498409 0.38424437 0.38313262 0.38240000
## [19] 0.38204218 0.38167017 0.19332755 0.19295447 0.19286333 0.19230694
## [25] 0.19212219 0.19157006 0.19157006 0.19147647 0.19111050 0.19083880
## [31] 0.07540012 0.07517338 0.07511690 0.07506051 0.07504614 0.07501799
## [37] 0.07500421 0.07496175 0.07491991 0.07472395 0.03834685 0.03818591
## [43] 0.03818591 0.03814221 0.03808765 0.03806590 0.03805866 0.03805511
## [49] 0.03804787 0.03802616 0.02571167 0.02569023 0.02549698 0.02548239
## [55] 0.02542085 0.02541274 0.02525711 0.02524754 0.02520773 0.02516649
## [61] 0.01533024 0.01530326 0.01528398 0.01521246 0.01519397 0.01517094
## [67] 0.01516519 0.01516059 0.01512733 0.01511645
```

Exhibit 4: Weights returned by the model.

[. . .] Which matches what was supplied as the weights argument:

```
InvAmt
```

```
##  [1] 1.00000000 0.99745521 0.99502437 0.98274997 0.98039020 0.97560733
##  [7] 0.97560733 0.97560733 0.96847203 0.96618024 0.38758801 0.38683826
## [13] 0.38647210 0.38498409 0.38498409 0.38424437 0.38313262 0.38240000
## [19] 0.38204218 0.38167017 0.19332755 0.19295447 0.19286333 0.19230694
## [25] 0.19212219 0.19157006 0.19157006 0.19147647 0.19111050 0.19083880
## [31] 0.07540012 0.07517338 0.07511690 0.07506051 0.07504614 0.07501799
## [37] 0.07500421 0.07496175 0.07491991 0.07472395 0.03834685 0.03818591
## [43] 0.03818591 0.03814221 0.03808765 0.03806590 0.03805866 0.03805511
```

```
## [49] 0.03804787 0.03802616 0.02571167 0.02569023 0.02549698 0.02548239
## [55] 0.02542085 0.02541274 0.02525711 0.02524754 0.02520773 0.02516649
## [61] 0.01533024 0.01530326 0.01528398 0.01521246 0.01519397 0.01517094
## [67] 0.01516519 0.01516059 0.01512733 0.01511645
```

Exhibit 5: The weights supplied to the model.

Thus, it is evident that when a vector of weights is supplied to lm(), when weights() is called on a fitted model that has weights, the weights are returned back in the same form.

A twist

So what is going on? It looks straight forward so far. Weights are calculated and supplied to the model, and the model is using them verbatim since we get the exact same values back. Well, this is only partly true. If the weights are taken as a direct transform of the x's and y, the parameter estimates are not identical.

```
# Incorrect method of weighting
attach(dat1)
wResp   <-Response*InvAmt
wAmt    <-Amount*InvAmt
wAmtSqr <-AmtSqr*InvAmt
incorrectWeighting<-lm(wResp~Amount+wAmtSqr)
detach(dat1)
summary(incorrectWeighting)
```

```
##
## Call:
## lm(formula = wResp ~ Amount + wAmtSqr)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -66.684 -18.570  -4.144  14.659 121.625
##
## Coefficients: (1 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 332.5867     6.1631  53.965   <2e-16 ***
## Amount        0.9586    20.2171   0.047    0.962
## wAmtSqr           NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 36.57 on 68 degrees of freedom
## Multiple R-squared:  3.306e-05,  Adjusted R-squared:  -0.01467
## F-statistic: 0.002248 on 1 and 68 DF,  p-value: 0.9623
```

Exhibit 6: Regression summary from incorrect manually weighted regression.

The model now looks like $Response = 332.5867 + 0.9586 * Amount$, which is does not coincide with the weighted regression from earlier when the weights were supplied to the function: $Response = 68.13 + 30874.00 * Amount + 6456.47 * Amount^2$. In fact, the squared term is missing altogether in the latest rendition. As an old professor of this author used to say to this kind of output, "SAS puked" (we were using SAS at the time).

Readers should be able to take the weights, use them to transform their data, fit an ordinary least squares regression as if it had no weights, and get an identical answer for the parameter estimates as weighted regression. What about, since the variance is $\epsilon^2 = \sum(w_i(Response_i - Model_i)^2$ (Judd, McClelland, and Ryan 2017, 17), which is what is being minimized with linear regression, if the square root of the weights is taken instead?

```
# Closer but still incorrect method of weighting
attach(dat1)
InvAmtSqrt<-sqrt(InvAmt) # Calculate the square root
# Recalculate the following with the new sqrt(weights)
wResp   <-Response*InvAmtSqrt
wAmt    <-Amount*InvAmtSqrt
wAmtSqr <-AmtSqr*InvAmtSqrt
closerIncorrectWeighting<-lm(wResp~wAmt+wAmtSqr)
detach(dat1)

summary(closerIncorrectWeighting)
```

```
##
## Call:
## lm(formula = wResp ~ wAmt + wAmtSqr)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -248.57  -45.49   12.66   63.16  164.62
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    65.48      31.24   2.096   0.0399 *
## wAmt        29545.98    1257.69  23.492  < 2e-16 ***
## wAmtSqr      7436.22    1719.20   4.325 5.18e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 92.54 on 67 degrees of freedom
## Multiple R-squared:  0.988,  Adjusted R-squared:  0.9877
## F-statistic:  2763 on 2 and 67 DF,  p-value: < 2.2e-16
```

Exhibit 7: Regression output from a *closer but still wrong method of weighting* model.

Well, the parameter estimates seem closer, but $Response = 65.48 + 29545.98 * Amount + 7436.2 * Amount^2$ is still not identical to $Response = 68.13 + 30874.00 * Amount + 6456.47 * Amount^2$. It is, however, an improvement since the squared term **at least has a result**. Now, the intercept may be separately taken as another variable in the model, perhaps a series of ones. Maybe if the intercept is then omitted, and the value is supplied as a variable manually, a different curve will result...

```
# Still incorrect method of weighting
wb0     <-rep(1,70) # Series of 1s

evenCloserIncorrectWeighting<-lm(wResp~0+wb0+wAmt+wAmtSqr) # Manual intercept

summary(evenCloserIncorrectWeighting)
```

```
##
## Call:
## lm(formula = wResp ~ 0 + wb0 + wAmt + wAmtSqr)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -248.57  -45.49   12.66   63.16  164.62
##
## Coefficients:
##         Estimate Std. Error t value Pr(>|t|)
## wb0        65.48      31.24   2.096   0.0399 *
## wAmt    29545.98    1257.69  23.492  < 2e-16 ***
## wAmtSqr  7436.22    1719.20   4.325 5.18e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 92.54 on 67 degrees of freedom
## Multiple R-squared:  0.9966, Adjusted R-squared:  0.9965
## F-statistic:  6632 on 3 and 67 DF,  p-value: < 2.2e-16
```

Exhibit 8: Regression summary from an "even closer but still wrong method of weighting" model.

Well, this matches the previous model, but it is not exactly the same as the original weighted regression where the weights were supplied to the lm() function. At least there seems to be some consistency on what is happening with the intercept as a series of ones, since excluding the intercept and supplying a variable manually for it reproduced the intercept seen earlier. This seems somewhat strange, as there is no variance in a series of ones, so how can a parameter be estimated from that? Anyway, maybe if that series of ones is also multiplied against the weights, along with all of the other variables in the model, it will work.

```
# Correct method of weighting
wb0     <-rep(1,70)*InvAmtSqrt # Also weight the series of 1s
```

```r
correctlyWeighted<-lm(wResp~0+wb0+wAmt+wAmtSqr) # Manual intercept

summary(correctlyWeighted)
```

```
##
## Call:
## lm(formula = wResp ~ 0 + wb0 + wAmt + wAmtSqr)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -247.073  -49.293    1.779   60.580  164.211
##
## Coefficients:
##         Estimate Std. Error t value Pr(>|t|)
## wb0        68.13      28.55   2.387   0.0198 *
## wAmt    30874.00     705.33  43.773  < 2e-16 ***
## wAmtSqr  6456.47    1355.37   4.764 1.06e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 91.71 on 67 degrees of freedom
## Multiple R-squared:  0.9967, Adjusted R-squared:  0.9966
## F-statistic:  6754 on 3 and 67 DF,  p-value: < 2.2e-16
```

Exhibit 9: Regression summary from a correctly manually weighted model fit.

OK, so the original weighted regression where the weights were supplied to lm() resulted in the model:

$$Response = 68.13 + 30874.00 * Amount + 6456.47 * Amount^2$$

We just took the **square root of the weights**, multiplied **that** against all values, the Y's, the x's, and even a vector of 1's that was supplied that was added to the model like any other variable, and fit the model **without** an intercept, to come up with:

$$Response = 68.13 + 30874.00 * Amount + 6456.47 * Amount^2$$

. . . which **is** identical to the original weighted model. What does this mean for a forced intercept?

Forced

Say the intercept is expected to be zero. There may be a solid theoretical foundation for this, such as expecting zero instrument response when no analyte is present. This is not always the case, but lets assume it is *in this instance*. A weighted model where the weights are supplied to the lm() function, the model is fit "without an intercept", and one where we

transform all of the variables by the square root of the weights, and leave out the vector of ones, should also result in the same output.

```
# Forced origin, normal weighting
weightedFitForced<-lm(Response~0+Amount+AmtSqr, weights=InvAmt, data=dat1)

summary(weightedFitForced)
```

```
##
## Call:
## lm(formula = Response ~ 0 + Amount + AmtSqr, data = dat1, weights = InvAmt)
##
## Weighted Residuals:
##     Min      1Q  Median      3Q     Max
## -236.80  -41.29   16.53   64.38  175.56
##
## Coefficients:
##        Estimate Std. Error t value Pr(>|t|)
## Amount  31902.9      577.2  55.267  < 2e-16 ***
## AmtSqr   4810.5     1206.5   3.987 0.000166 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 94.83 on 68 degrees of freedom
## Multiple R-squared:  0.9964, Adjusted R-squared:  0.9963
## F-statistic:  9473 on 2 and 68 DF,  p-value: < 2.2e-16
```

Exhibit 10: Regression summary from the automatically weighted, forced model.

The model above with the forced intercept is $Response = 0 + 31902.9 * Amount + 4810.5 * Amount^2$. The slope of Amount seems to be fairly stable across the models so far, with most of the differences being in the squared term. Now to try the hypothesized manual forced model where the manually weighted variables are supplied without model weights.

```
# Manual forced
manuallyForced<-lm(wResp~0+wAmt+wAmtSqr) # Manual intercept

summary(manuallyForced)
```

```
##
## Call:
## lm(formula = wResp ~ 0 + wAmt + wAmtSqr)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -236.80  -41.29   16.53   64.38  175.56
##
```

```
## Coefficients:
##         Estimate Std. Error t value Pr(>|t|)
## wAmt     31902.9      577.2  55.267  < 2e-16 ***
## wAmtSqr   4810.5     1206.5   3.987 0.000166 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 94.83 on 68 degrees of freedom
## Multiple R-squared:  0.9964, Adjusted R-squared:  0.9963
## F-statistic:  9473 on 2 and 68 DF,  p-value: < 2.2e-16
```

Exhibit 11: Regression summary from a manually weighted and forced through zero model.

Forcing the intercept to zero, omitting the vector of ones, and the manually weighted variables instead of using model weights results in $Response = 31902.9 * Amount + 4810.5 * Amount^2$, which is *identical* to the model that forced the intercept to zero AND supplied model weights. Does this approach work for other types of weighting?

Inverse squared

In the code exhibit below, a linear model is fit with the weights as the normalized inverse Amounts squared. The intercept is still forced to zero.

```
fitInvWeightsSquared<-lm(Response~0+Amount+AmtSqr, weights=InvAmtSqr,
         data=dat1)
summary(fitInvWeightsSquared)
```

```
##
## Call:
## lm(formula = Response ~ 0 + Amount + AmtSqr, data = dat1, weights = InvAmtSqr)
##
## Weighted Residuals:
##     Min      1Q  Median      3Q     Max
## -66.684 -18.570  -4.144  14.659 121.625
##
## Coefficients:
##        Estimate Std. Error t value Pr(>|t|)
## Amount 33940.88     628.95  53.965   <2e-16 ***
## AmtSqr    97.83    2063.18   0.047    0.962
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 36.57 on 68 degrees of freedom
## Multiple R-squared:  0.9884, Adjusted R-squared:  0.9881
## F-statistic:  2898 on 2 and 68 DF,  p-value: < 2.2e-16
```

Exhibit 12: Regression summary for an automatically weighted, forced through zero model with different weights.

Using squared weights results in a (forced) model where $Response = 0 + 33940.88 * Amount + 97.83 * Amount^2$, and the squared term now looks far from useful, like it seemed previously. The curvature in the model is being accounted for by the weights instead of a parameter estimate.

If the strategy developed so far continues to work, the weights should be plug & play. The new weighted variables should just need to be calculated and a forced model fit to get the **exact** same result as the model where the weights were supplied (above).

```
attach(dat1)
InvAmtSqr_Sqrt<-sqrt(InvAmtSqr) # Calculate the square root
# Calculate the following with the new sqrt(weights)
wResp2   <-Response*InvAmtSqr_Sqrt
wAmt2    <-Amount*InvAmtSqr_Sqrt
wAmtSqr2 <-AmtSqr*InvAmtSqr_Sqrt
fitManuallyInvWeightsSquared<-lm(wResp2~0+wAmt2+wAmtSqr2)
detach(dat1)
summary(fitManuallyInvWeightsSquared)
```

```
##
## Call:
## lm(formula = wResp2 ~ 0 + wAmt2 + wAmtSqr2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -66.684 -18.570  -4.144  14.659 121.625
##
## Coefficients:
##          Estimate Std. Error t value Pr(>|t|)
## wAmt2    33940.88     628.95  53.965   <2e-16 ***
## wAmtSqr2    97.83    2063.18   0.047    0.962
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 36.57 on 68 degrees of freedom
## Multiple R-squared:  0.9884, Adjusted R-squared:  0.9881
## F-statistic:  2898 on 2 and 68 DF,  p-value: < 2.2e-16
```

Exhibit 13: Regression summary for a manually weighted model forced through zero model with different weights.

Where $Response = 0 + 33940.88 * Amount + 97.83 * Amount^2$ was expected, the manual result is $Response = 0 + 33940.88 * Amount + 97.83 * Amount^2$. Wala. The strategy holds.

To summarize, supplying weights to the lm() function in R is **identical** to taking the square

root of those weights times each variable, and times a vector of 1s, and then fitting a model *without* an intercept using those new weighted variables. Forcing the intercept is simply omitting the vector of ones from the model. There are Matrix Algebra explanations for what was just performed with weighted regression, though this author forgot most of those years ago and does not expect readers to know them. It may be interesting now to take a look at what the residual and Q-Q plots look like for a weighted fit, since the absence of model diagnostics is this author's main beef with the Analytical Chemistry community.

Diagnostics

See figure 5 for a side-by-side of the Quantile-Quantile plots for a calibration model with a forced intercept fit using the $\frac{min(Amount)}{Amount}$ weights, and the $\frac{min(Amount^2)}{Amount^2}$ weights.

```
# qqPlot from package {car}
par(mfrow = c(1,2), las = 0.5, mar = c(4,3,3,2)+.1)
qqPlot(manuallyForced, distribution = "t",
       main = expression(frac("min("~"Amount"^1~")","Amount"^1)),
       ylab = "Studentized residuals",
       line = "quartiles", cex = 0.75,
       simulate = FALSE, envelope=0.9995)
qqPlot(fitManuallyInvWeightsSquared, distribution = "t",
       main = expression(frac("min("~"Amount"^2~")","Amount"^2)),
       ylab = "Studentized residuals",
       line = "quartiles", cex = 0.75,
       simulate = FALSE, envelope=0.9995)
```

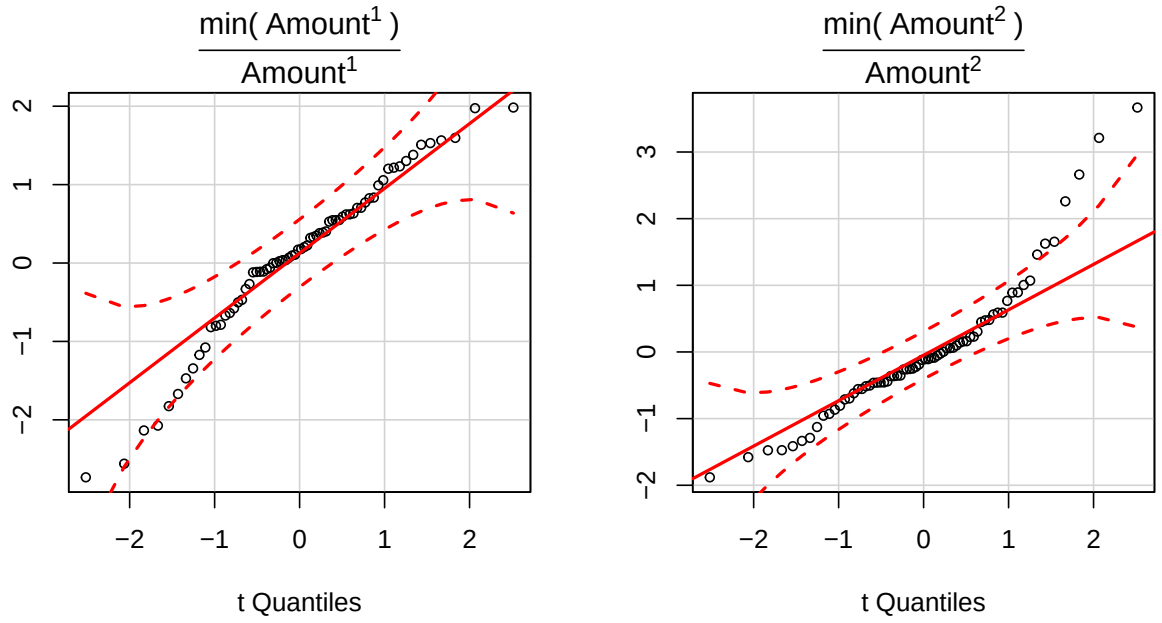

Figure 5: Forced weightings with a 99.95% confidence interval.

Notice that in figure 5, the confidence envelope for the two graphs is set to 99.95%, just as in figure 4. There is still some issues happening at the [-3,-1] quartile range in the $\frac{min(Amount)}{Amount}$ weighted model's Q-Q plot, but that same section looks good in the graph of the Q-Q plot from the $\frac{min(Amount^2)}{Amount^2}$ weighted model. On the other side, $\frac{min(Amount)}{Amount}$ looks good from a range of approximately [-1,3], while the inverse is true in the squared inverse weighted model. More problematic in the squared inverse weighted model's Q-Q plot are the outliers. 10% of the points (7/70) are outliers, even with a 99.95% confidence interval.

There are several different things that can be addressed for this calibration. One is trying different weightings, individually and/or in combination. Another is to see if the included intercept model's quartiles look better. Another still is to call the results so far good enough and use a robust technique to get the rest of the way. Finally, one could find weights that are essentially a transform that is right in the middle of the two weights that resulted in the Q-Q plots in figure 5, meaning perhaps there exists some monotonic transform of the data that is essentially right in the middle of $\frac{min(Amount)}{Amount}$ and $\frac{min(Amount^2)}{Amount^2}$ weighting. But just to show, the Q-Q plots of the forced models with a typical 95% confidence interval.

```
# qqPlot from package {car}
par(mfrow = c(1,2), las = 0.5, mar = c(4,3,3,2)+.1)
qqPlot(manuallyForced, distribution = "t",
       main = expression(frac("min("~"Amount"^1~")","Amount"^1)),
       ylab = "Stude ntized residuals",
       line = "quartiles", cex = 0.75,
       simulate = FALSE, envelope=0.95)
qqPlot(fitManuallyInvWeightsSquared, distribution = "t",
       main = expression(frac("min("~"Amount"^2~")","Amount"^2)),
       ylab = "Studentized residuals",
       line = "quartiles", cex = 0.75,
       simulate = FALSE, envelope=0.95)
```

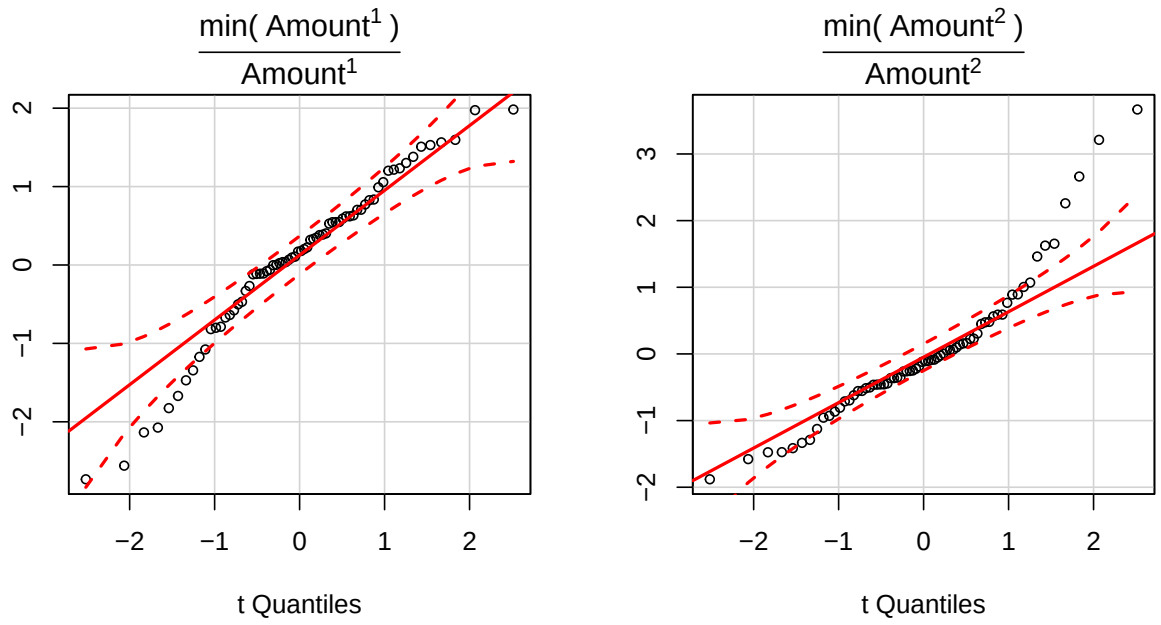

Figure 6: Forced weightings with a 95% confidence interval.

Both $\frac{min(Amount)}{Amount}$ and $\frac{min(Amount^2)}{Amount^2}$ weighting have 10% outliers, although the magnitude of them is far less when using $\frac{min(Amount)}{Amount}$. Originally, there were 7 outliers in the upper quartile range and 12 outliers in the lower quartile range with the unweighted calibration model, which amounted to 27% of data. This author would be inclined to accept the simplicity of $\frac{min(Amount)}{Amount}$ weighting for routine laboratory work with this assay, provided the assay proves to give good results in a particular range of interest.

```
# Manual forced
attach(dat1)
# New weights
InvResp   <- min(Response)/Response
InvRespSqr<- min(RespSqr)/(RespSqr)
wResp     <- Response*InvResp
wResp2    <- Response*InvRespSqr
wAmt      <- Amount*InvResp
wAmt2     <- Amount*InvRespSqr
wAmtSqr   <- AmtSqr*InvResp
wAmtSqr2  <- AmtSqr*InvRespSqr
detach(dat1)

# Fit manual models
manuallyResponseForced<-lm(wResp~0+wAmt+wAmtSqr)
manuallyResponseSqrForced<-lm(wResp2~0+wAmt2+wAmtSqr2)

# qqPlot from package {car}
par(mfrow = c(1,2), las = 0.5, mar = c(4,3,3,2)+.1)
```

```
qqPlot(manuallyResponseForced, distribution = "t",
       main = expression(frac("min("~"Response"^1~")","Response"^1)),
       ylab = "Studentized residuals",
       line = "quartiles", cex = 0.75,
       simulate = FALSE, envelope=0.95)
qqPlot(manuallyResponseSqrForced, distribution = "t",
       main = expression(frac("min("~"Response"^2~")","Response"^2)),
       ylab = "Studentized residuals",
       line = "quartiles", cex = 0.75,
       simulate = FALSE, envelope=0.95)
```

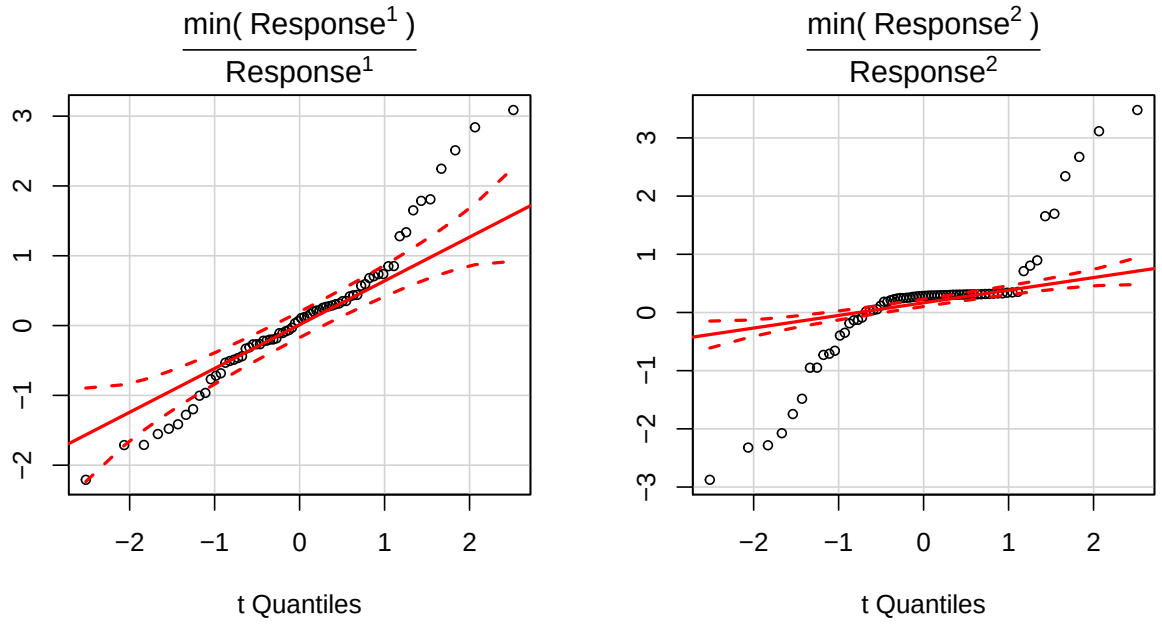

Figure 7: Q-Q Plots using the inverse of Response as the weighting.

Clearly, $\frac{min(Amount)}{Amount}$ weighting out performs $\frac{min(Amount^2)}{Amount^2}$, $\frac{min(Response)}{Response}$, and $\frac{min(Response^2)}{Response^2}$ weighting in these data. $\frac{min(Response)}{Response}$ is close, it just did not result in the same gains in the upper quartile range as $\frac{min(Amount)}{Amount}$ weighting did. $\frac{min(Response^2)}{Response^2}$ weighting made things worse. With $\frac{min(Response^2)}{Response^2}$ weighting, now 36% of the residuals are outliers!

Ignored intercepts

Just to be sure the handling of the intercept is not breaking the Q-Q plots in the lower quartile range, this author is providing the Q-Q plots for the $\frac{min(Amount)}{Amount}$ and $\frac{min(Amount^2)}{Amount^2}$ models again, only this time the models are fit with the vector of ones times the weights as a parameter in the model as well.

```r
# Manual forced
attach(dat1)
# New weights
InvAmt    <- min(Amount)/Amount
InvAmtSqr <- min(AmtSqr)/(AmtSqr)
wResp     <- Response*InvAmt
wResp2    <- Response*InvAmtSqr
wAmt      <- Amount*InvAmt
wAmt2     <- Amount*InvAmtSqr
wAmtSqr   <- Amount^2*InvAmt
wAmtSqr2  <- Amount^2*InvAmtSqr
wb0       <- rep(1,70)*InvAmt
wb02      <- rep(1,70)*InvAmtSqr
detach(dat1)

# Fit manual models
manuallyAmountUnforced<-lm(wResp~wb0+wAmt+wAmtSqr)
manuallyAmtSqrUnforced<-lm(wResp2~wb02+wAmt2+wAmtSqr2)

# graphical parameters
par(mfrow = c(1,2), las = 0.5, mar = c(4,3,3,2)+.1)
# qqPlot from package {car}
qqPlot(manuallyAmountUnforced, distribution = "t",
       main = expression(frac("min("~"Amount"^1~")","Amount"^1)),
       ylab = "Studentized residuals",
       line = "quartiles", cex = 0.75,
       simulate = FALSE, envelope=0.95)
qqPlot(manuallyAmtSqrUnforced, distribution = "t",
       main = expression(frac("min("~"Amount"^2~")","Amount"^2)),
       ylab = "Studentized residuals",
       line = "quartiles", cex = 0.75,
       simulate = FALSE, envelope=0.95)
```

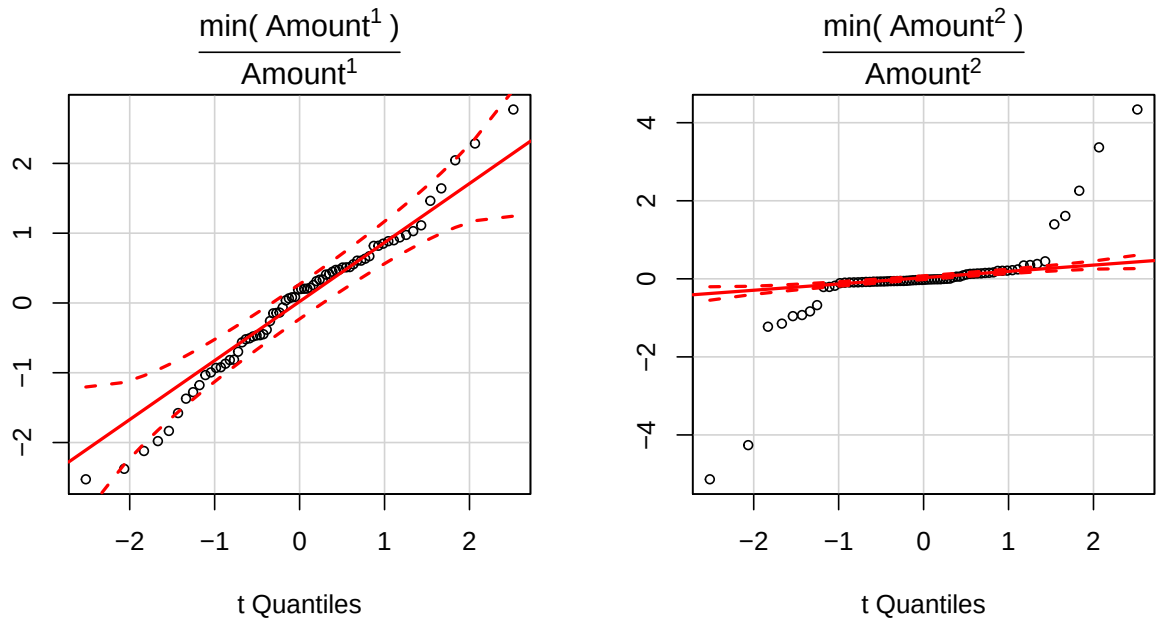

Figure 8: Q-Q Plots using the inverse of Amount as the weighting, and not forcing the intercept through zero.

Shocking! The handling of the intercept mattered. In the model with $\frac{min(Amount)}{Amount}$ weights, the residuals now fall within the 95% confidence interval except for 3, and even those are really close. The model with $\frac{min(Amount^2)}{Amount^2}$ weights just produced a Q-Q plot that looks horrific, even compared to the one that forced the intercept through zero. Clearly inverse-squared weighting is not the way to go for these data. Readers still must check the residuals plot as well to see if the goal of obtaining equal variance across the curve was met.

```
# Set graphical parameters
par(las = 0.5, mar = c(4,3,3,2)+.1)
# Plot the residuals
fig9<-ggplot()+
  geom_point(aes(x=predict(manuallyAmountUnforced),
                 y=resid(manuallyAmountUnforced)))+
  stat_smooth(aes(x=predict(manuallyAmountUnforced),
                  y=resid(manuallyAmountUnforced)),
              method="lm", formula=y~x,
              level=0.99999) +
  ggtitle(expression("Residuals for"~frac("min(Amount)","Amount")~"Weights"))+
  xlab("Predicted")+ylab("Residuals")

fig9
```

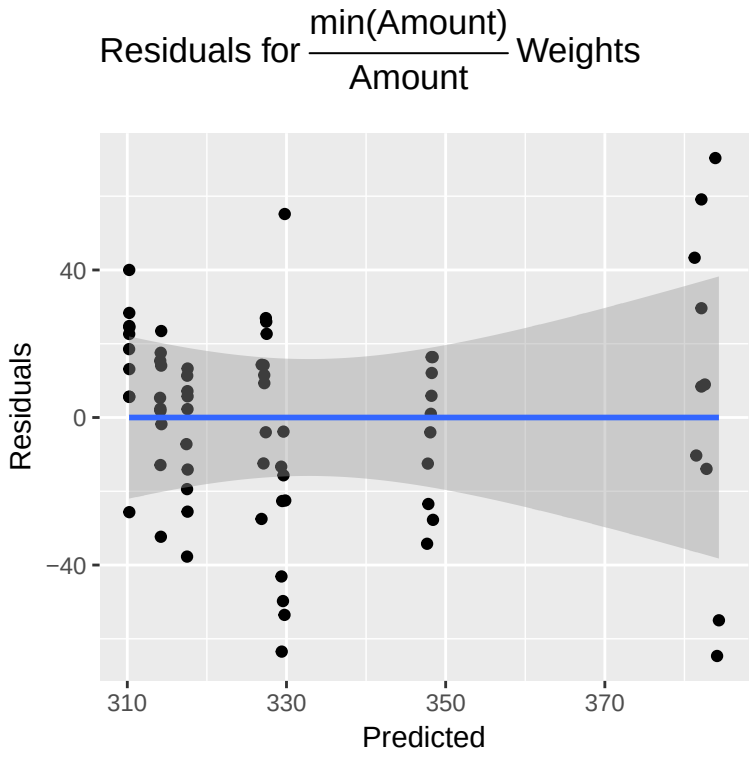

Figure 9: Residuals plot for model weighting by inverse Amount that is not forcing the intercept.

Figure 9 shows that the residuals at the bottom of the curve now have a wider spread. The residuals at the of the curve are squeezed in more. This is clearly an improvement on the residuals evidenced in figure 2 from the unweighted fit, although the levels of the calibration are not approximately equally spaced on the Predicted axis. So what about using a transform instead of weighting?

Transform

The code in the exhibit below uses a "monotonic transform" rather than weighting. Furthermore, the transform is taken on both axes. The difference between weighting and the transform used here is that the data are taken to a power that stays the same across X and Y, rather than a muliplicand that changes across X and Y. Trial and error was used to land on an appropriately simple transform of the power of 1/5th for these data. What was tuned was the look of the Q-Q and Residuals plots.

```
# Transformed variables
attach(dat1)
tResp <- Response^(.2)
tAmt  <- Amount^(.2)
tAmtSqr <- tAmt^2
tInvAmt <- min(tAmt)/tAmt
```

```r
detach(dat1)

# Fit a calibration model
xformedModel<-lm(tResp~tAmt+tAmtSqr)

summary(xformedModel)
```

```
##
## Call:
## lm(formula = tResp ~ tAmt + tAmtSqr)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.16491 -0.05699  0.01299  0.05452  0.13805
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.7774     0.1525   5.098 3.03e-06 ***
## tAmt          5.5028     0.4903  11.223  < 2e-16 ***
## tAmtSqr       1.9284     0.3719   5.185 2.17e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0776 on 67 degrees of freedom
## Multiple R-squared:  0.9971, Adjusted R-squared:  0.997
## F-statistic: 1.16e+04 on 2 and 67 DF,  p-value: < 2.2e-16
```

```r
# Graphical parameters
```

Exhibit 14: Regression summary from unweighted, transformed model.

These transformed data say that $transformedResponse = 0.7774 + 5.5028 * transformedAmount + 1.9284 * transformedAmount^2$, and all of the parameters contribute to predictions. For back-solving to the x-axis, keep in mind that an inverse transform will have to be performed. Since the data were raised to the power of 1/5, it can be raised to the power of 5/1 to invert the transform operation.

```r
# graphical parameters
par(las = 0.5, mar = c(4,3,3,2)+.1)
# qqPlot from package {car}
qqPlot(xformedModel, distribution = "t",
       main = expression("Q-Q Plot for Data"^{.2}~"Transform"),
       ylab = "Studentized residuals",
       line = "quartiles", cex = 0.75,
       simulate = FALSE, envelope=0.95)
```

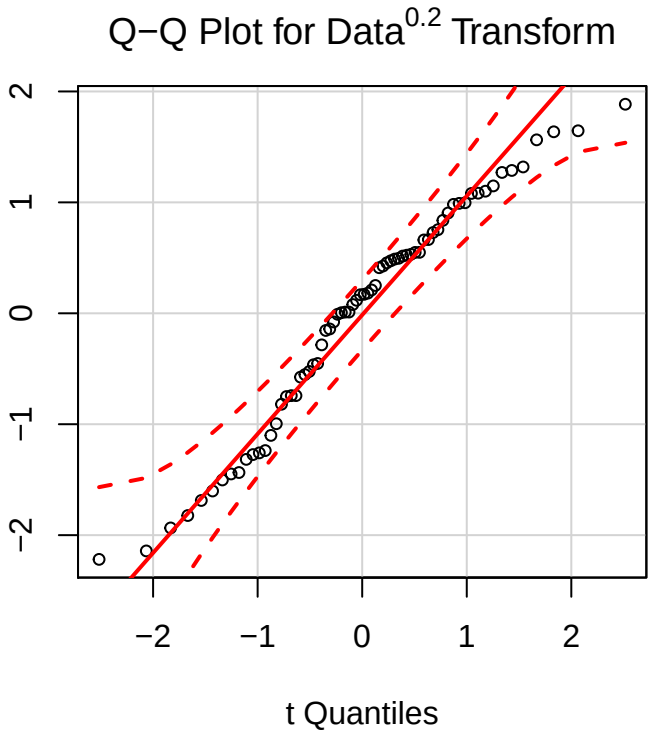

Figure 10: Q-Q Plot of a model fit after transforming data instead of weighting, with a 95% confidence interval.

The $Data^{0.2}$ transform achieved what none of the weighting strategies looked at could. Every single datum is within the 95% confidence interval on the Q-Q plot. Furthermore, the Q-Q plot evidences a line that explains the residuals from [-2,1.5] quantiles now. The other half of the test, however, is the residuals plot, so that plot is shown next.

```
# graphical parameters
par(las = 0.5, mar = c(4,3,3,2)+.1)

fig11<-ggplot()+
  geom_point(aes(x=predict(xformedModel),
                 y=resid(xformedModel)))+
  stat_smooth(aes(x=predict(xformedModel),
                  y=resid(xformedModel)),
              method="lm", formula=y~x,
              level=0.99999) +
  ggtitle(expression("Residuals for Data"^{.2}~"Transform"))+
  xlab("Predicted")+ylab("Residuals")
fig11
```

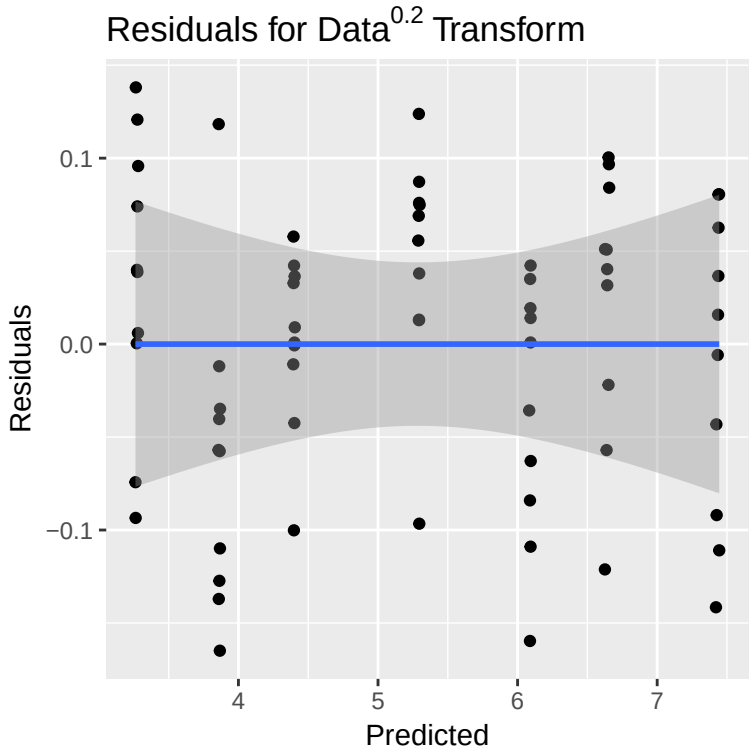

Figure 11: Residuals for $Data^{0.2}$ transform and a 99.999% confidence interval (same as the prior residuals plot).

Not only does the variance at each level look more evenly spread in figure 11, which indicates the problem of heterogeneity of variance has been taken care of sufficiently, the levels of the curve now look approximately equally spaced, which takes care of another regression assumption that was not discussed in this post. Furthermore, the confidence bands are more centered about data and symmetric. Clearly, for analytical work with this assay, if this author had to choose what method to employ for routine work it would be the $Data^{0.2}$ **transform over any of the weightings**. The $\frac{min(Amount)}{Amount}$ weighting is a close second due to its ease to employ (it is supported in the instrument's quantitative software and thus already validated).

This author hopes you enjoyed this post. I had a lot of fun making it!

Cheers.

References

Agilent Technologies. 2016. *Agilent OpenLAB CDS: Data Analysis Reference Guide*. Waldbronn: M8410-90031.

Carroll, R J, and D Ruppert. 1988. *Transformation and Weighting in Regression*. London, UK, UK: Chapman & Hall, Ltd.

Chamberlain, Keith A. 2017. "70 point Acetic acid curve by GC/FID demonstrates

heterogeneity of variance in analytical data." Mendeley Data, v2. https://doi.org/http://dx.doi.org/10.17632/dwf4ddww3w.2.

Danzer, Klaus. 2007. *Analytical Chemistry, theoretical and metrological foundations*. Berlin Heidelberg: Springer-Verlag. https://doi.org/10.1007/b103950.

Fox, John, and Sanford Weisberg. 2011. *An {R} Companion to Applied Regression*. Second. Thousand Oaks {CA}: Sage. http://socserv.socsci.mcmaster.ca/jfox/Books/Companion.

Garden, John S., Douglas G. Mitchell, and Wayne N. Mills. 1980. "Nonconstant variance regression techniques for calibration-curve-based analysis." *Analytical Chemistry* 52 (14). American Chemical Society:2310–5. https://doi.org/10.1021/ac50064a017.

Judd, Charles M., Gary H. McClelland, and Carey S. Ryan. 2017. *Data Analysis: A Model Comparison Approach To Regression, ANOVA, and Beyond*. 3rd ed. Routledge.

Ketkar, S. N., and T. J. Bzik. 2000. "Calibration of Analytical Instruments. Impact of Nonconstant Variance in Calibration Data." *Analytical Chemistry* 72 (19). American Chemical Society:4762–5. https://doi.org/10.1021/AC000018S.

Kutner, Michael H., Christopher J. Nachtsheim, and John Neter. 2004. *Applied Linear Regression Models*. 4th ed. McGraw-Hill.

Microsoft. 2016. "Visual Basic for Applications (VBA)." https://msdn.microsoft.com/en-us/vba/language-reference-vba/articles/visual-basic-language-reference.

———. 2018. "Visual Studio Code." https://code.visualstudio.com/.

R Core Team. 2017. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.r-project.org/.

RStudio Team. 2018. "RStudio: Integrated Development Environment for R." www.rstudio.com.

Schwartz, Lowell M. 1979. "Calibration curves with nonuniform variance." *Analytical Chemistry* 51 (6). American Chemical Society:723–27. https://doi.org/10.1021/ac50042a033.

Tellinghuisen, Joel. 2009. "Variance function estimation by replicate analysis and generalized least squares: A Monte Carlo comparison." *Chemometrics and Intelligent Laboratory Systems* 99 (2):138–49. https://doi.org/10.1016/j.chemolab.2009.09.001.

Venables, W N, and B D Ripley. 2002. *Modern Applied Statistics with S*. Fourth. New York: Springer. http://www.stats.ox.ac.uk/pub/MASS4.

Xie, Yihui. 2014. "knitr: A Comprehensive Tool for Reproducible Research in {R}." In *Implementing Reproducible Computational Research*, edited by Victoria Stodden, Friedrich Leisch, and Roger D Peng. Chapman; Hall/CRC. http://www.crcpress.com/product/isbn/9781466561595.

———. 2015. *Dynamic Documents with {R} and knitr*. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. https://yihui.name/knitr/.

———. 2017. *knitr: A General-Purpose Package for Dynamic Report Generation in R.* https://yihui.name/knitr/.