Transformations and Linear Regression

This document examines what happens when your model assumptions are wrong and then provides a real-world data set to do diagnostics and what happens when you transform data.

A Simulation

We'll take the simulation we did for linear regression, but we'll add in the fact that the linear model applies for $\log(y)$ rather than for y and we'll see what happens.

To do this, we'll define coefmat and predmat as before and set up β_0, β_1 and σ:

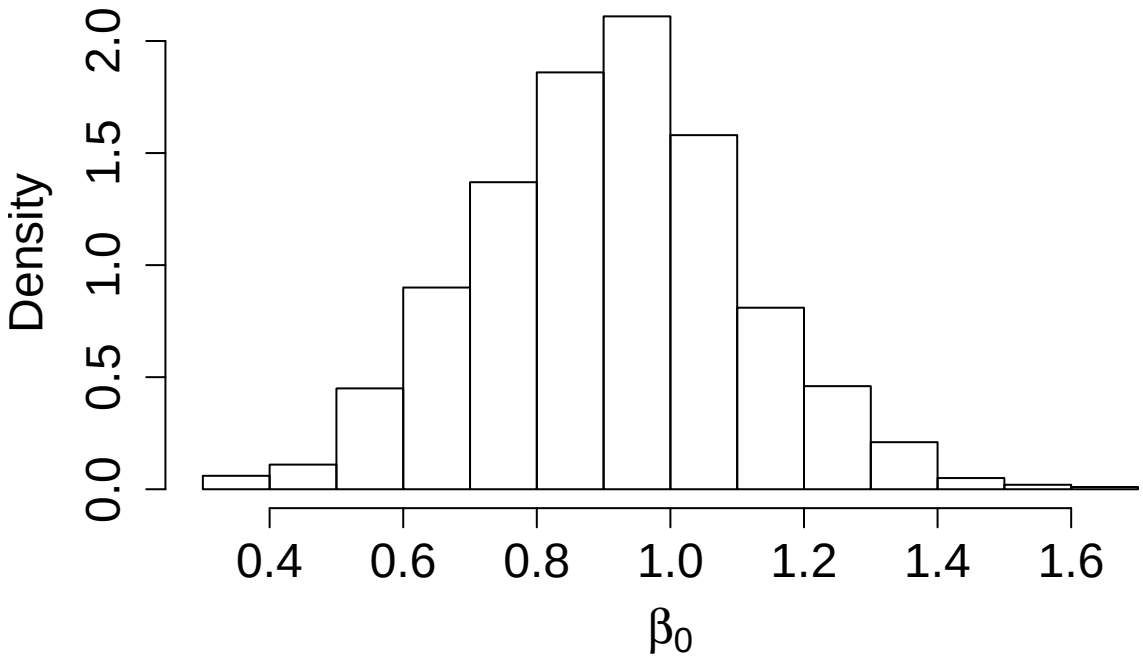
```
beta0 = 0;   beta1 = 1; sigma = 0.25
X = seq(0,1,by=0.1)
coefmat = matrix(0,1000,2)
predmat = matrix(0,1000,11)
```

The only difference in the code below is that I exponentiate Y. Note that this means that even the "best" estimate for β_0 and β_1 will be far from the correct value because they apply to $\log(y)$ not to y.

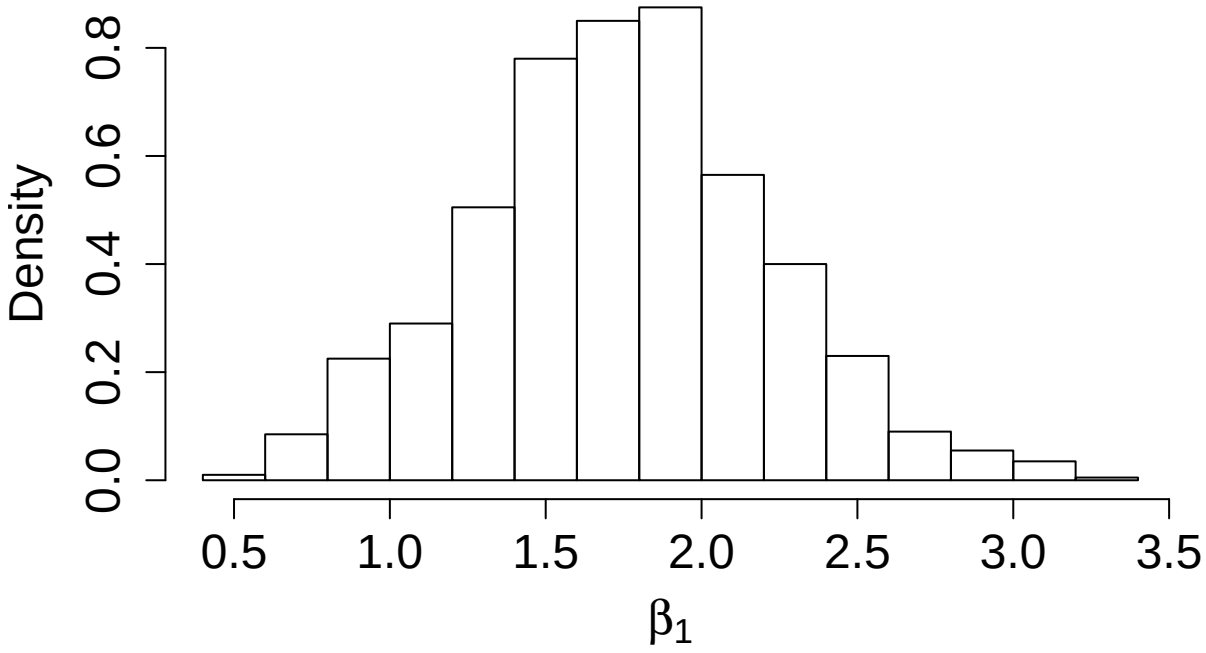
```
for(sim in 1:1000){
  epsilon = rnorm(11,mean=0,sd=sigma) # New observation errors
  Y =  beta0 + beta1*X + epsilon      # New response values
  Y = exp(Y)

  mod = lm(Y~X)                       # Refit the model
  coefmat[sim,] = mod$coefficients    # Store fitted coefs

  predmat[sim,] = mod$fit             # Store fitted values
}
```

Now if we look at the histograms of the parameters we get

```
hist(coefmat[,1],prob=TRUE,xlab=expression(beta[0]),main='',cex.lab=1.5,cex.axis=1.5)
```




```
hist(coefmat[,2],prob=TRUE,xlab=expression(beta[1]),main='',cex.lab=1.5,cex.axis=1.5)
```



Lets have a look at the variance of each of these

```
var0=var(coefmat[,1])
var1=var(coefmat[,2])
```

If we wanted to calculate confidence intervals for the last of these models we'd get

```
confint(mod)
```

```
##                 2.5 %   97.5 %
## (Intercept) 0.1166946 1.387361
## X           1.1176421 3.265461
```

which we can compare to using the variances obtained by simulation:

```
c(mod$coef[1]-1.96*sqrt(var0),mod$coef[1]+1.96*sqrt(var0))
```

```
## (Intercept) (Intercept)
##   0.3611914   1.1428646
```

```
c(mod$coef[2]-1.96*sqrt(var1),mod$coef[2]+1.96*sqrt(var1))
```

```
##        X        X
## 1.272535 3.110569
```

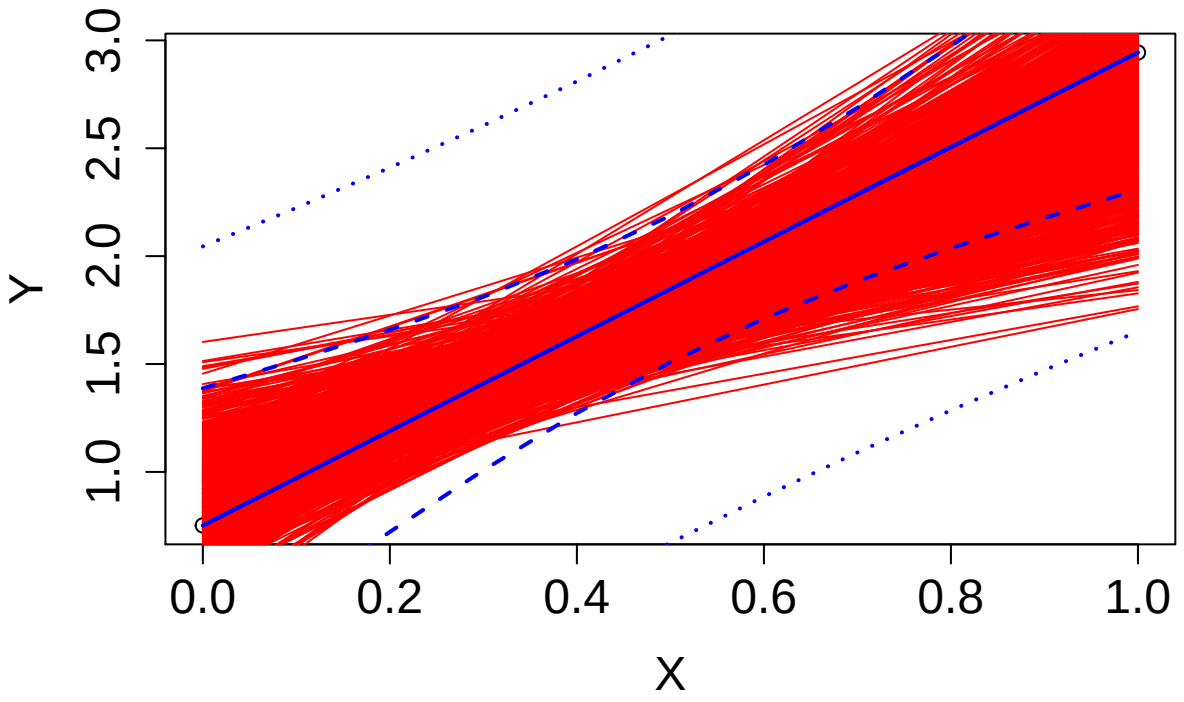
Where we see some noticeable differences, particularly in the confidence interval for β_0.

We can also plot out the predicted values as before, but add in the confidence intervals from the estimated prediction

```
preds = predict(mod,interval="prediction")
```

```
## Warning in predict.lm(mod, interval = "prediction"): predictions on current data refer to _future_ r
```

```
plot(X,preds[,1],ylab='Y',cex.lab=1.5,cex.axis=1.5)
for(i in 1:1000){ lines(X,predmat[i,],col='red') }

matplot(X,preds,type='l',lwd=2,col='blue',lty=c(1,3,3),add=TRUE)

preds2 = predict(mod,interval="confidence")
matplot(X,preds2,type='l',lwd=2,col='blue',lty = c(1,2,2),add=TRUE)
```



Here we can also see the effect a poorly specified model.

Of course, if we ran the same simulation using $\log(y)$ as a response, we would be back in the standard model framework and everything would work nicely.

An Example: Whitecap Data

These data come from Monahan, O'Muircheartaigh and Fiszgerald, 1981: **Determination of surface wind speed from remotely measured whitecap coverage, a feasibility assessment**, *Proceedings of an*

EARSeL-ESA Symposium: Application of Remote Sensing on the Continental Shelf, Voss, Norway, European Space Agency and were given to me by Ed Monahan (a Cornell alum) to be used in a meta-analysis.

The data set is found in

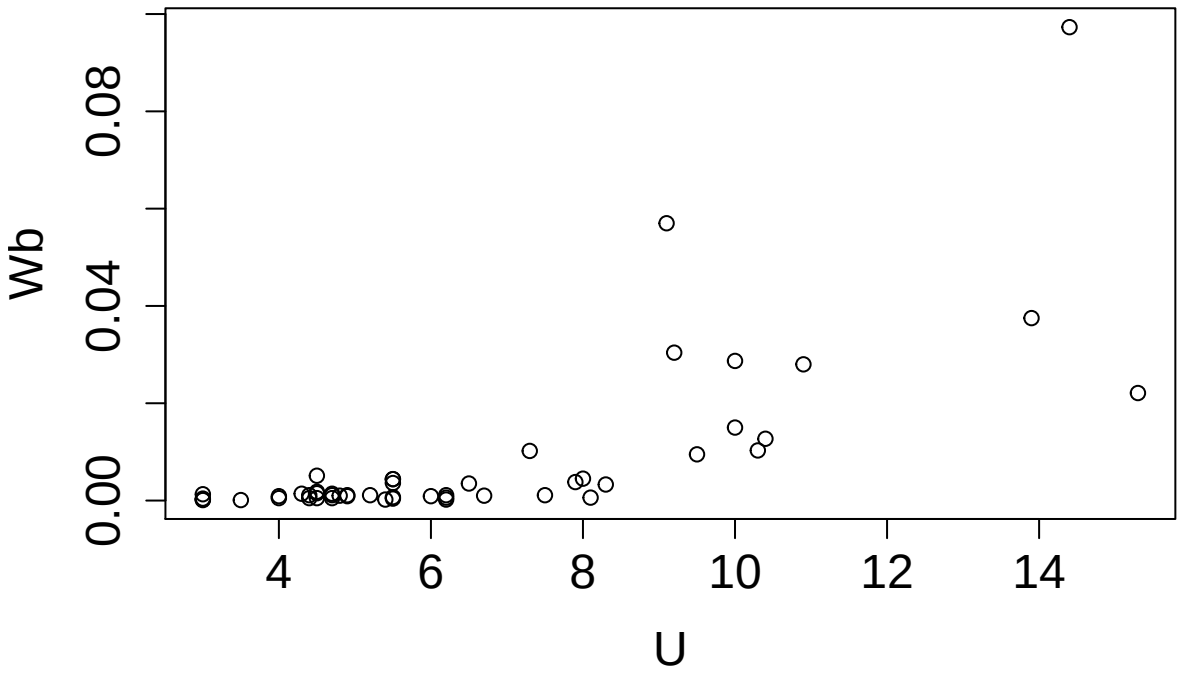
```
Wbdat = read.table('MMF81.csv',sep=',',head=TRUE)
```

In these data, `Wb" indicates the proportion of ocean surface that is covered in sea foam whileU` is the windspeed at the time of measurement (these data were collected from the North Sea in 1980 and Wb was calculated by cutting out portions of a photograph of the sea). This relationship is a crucial component of modeling gas transfer between the atmosphere and oceans and has implications for forecasts of climate under differing CO2 scenarios.

Exploring the data

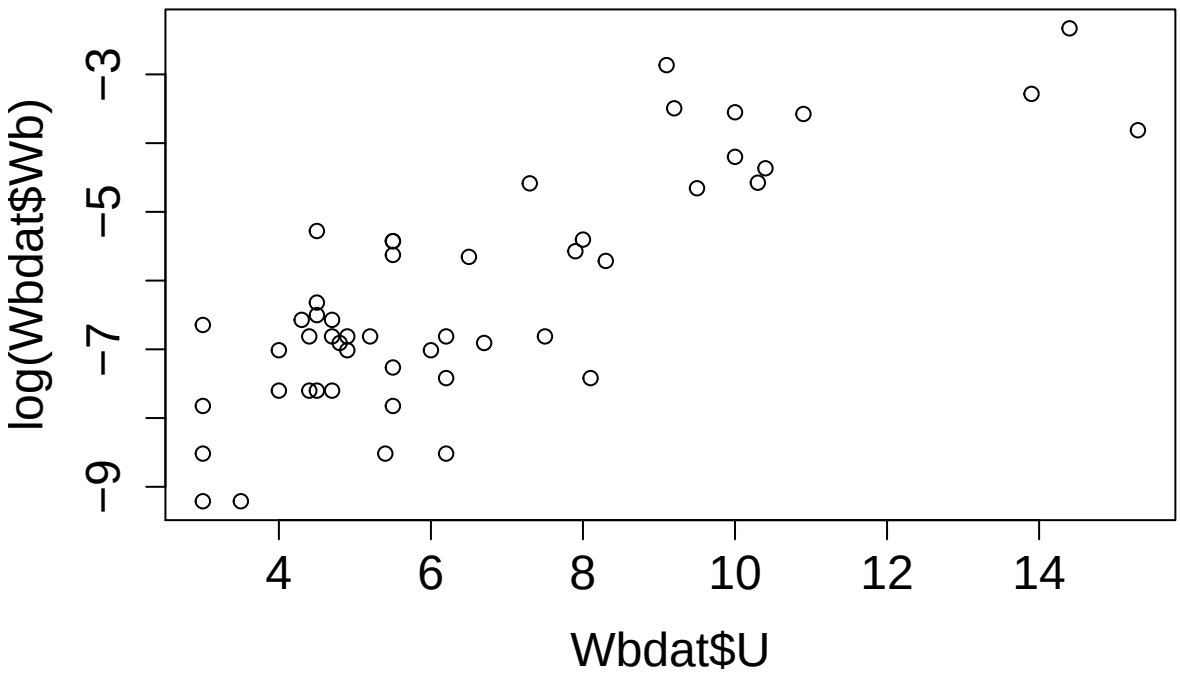
If we plot these, we see a typical "trumpet" shape in the data

```
plot(Wbdat,cex.lab=1.5,cex.axis=1.5)
```



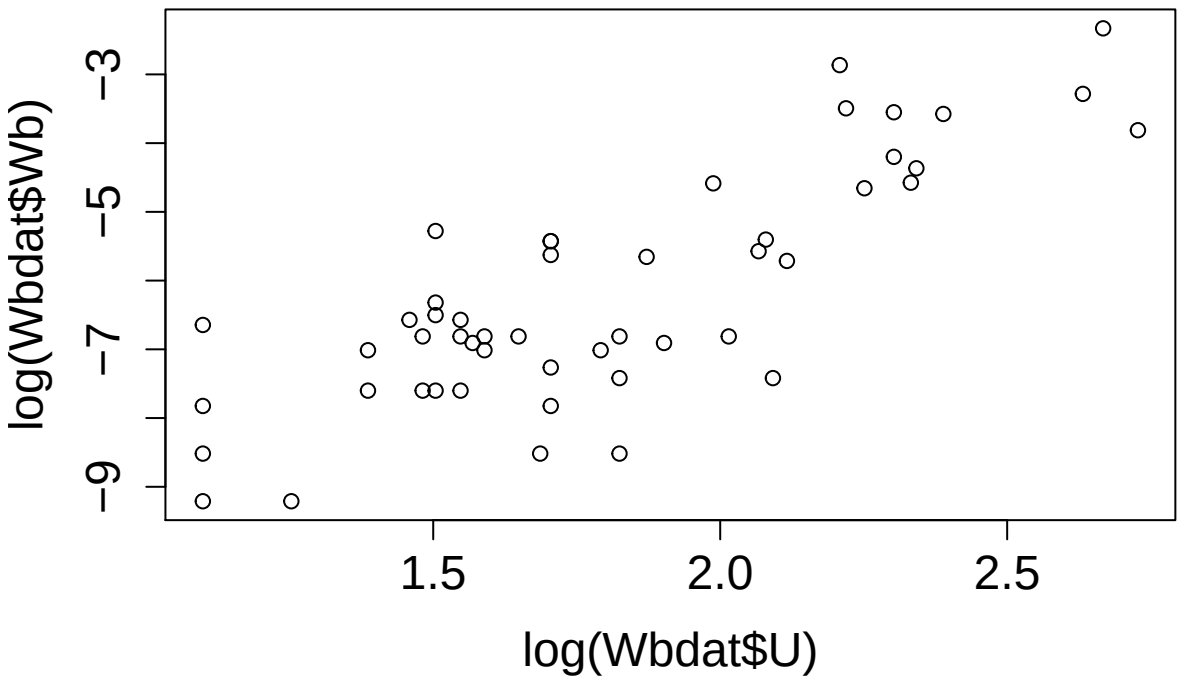
An obvious thing to look at, then, is to take a log transformation of Wb:

```
plot(Wbdat$U,log(Wbdat$Wb),cex.lab=1.5,cex.axis=1.5)
```



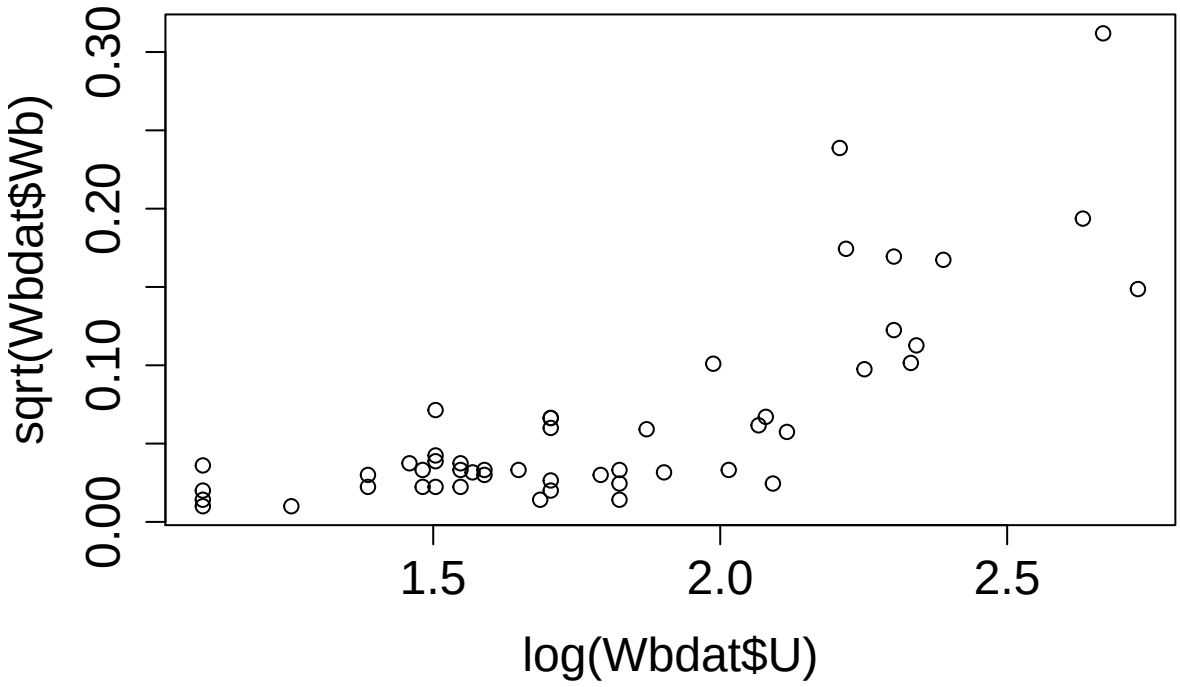
This looks better – certainly the variance doesn't appear to change over time – but we can see some distinct evidence of curvature, so let's try using $\log(U)$ as a covariate instead:

```
plot(log(Wbdat$U),log(Wbdat$Wb),cex.lab=1.5,cex.axis=1.5)
```

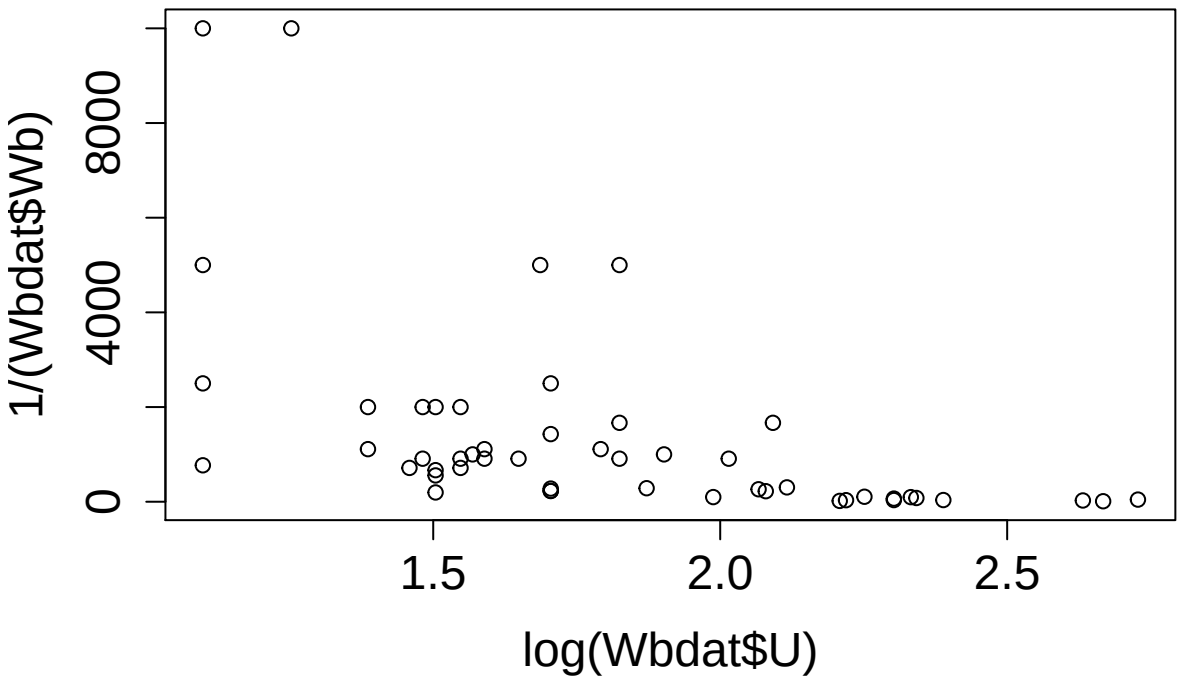


This looks much more like a reasonable model. Just checking out alternative transformations see some distinct evidence of curvature, so let's try using $\log(U)$ as a covariate instead:

```
plot(log(Wbdat$U),sqrt(Wbdat$Wb),cex.lab=1.5,cex.axis=1.5)
```




```
plot(log(Wbdat$U),1/(Wbdat$Wb),cex.lab=1.5,cex.axis=1.5)
```



We see that the square root transform isn't strong enough while the inverse transform is too strong – the trumpet now goes the other way.

Fitting $\log(Wb)$ to $\log(U)$

A quick note on this relationship, we have

$$\log(Y) = \beta_0 + \beta_1 \log(X) + \epsilon$$

If we exponentiate each side we get

$$Y = e^{\beta_0 + \beta_1 \log(X) + \epsilon} = e^{\beta_0} e^{\beta_1 \log(X)} e^{\epsilon} = C X^n e^{\epsilon}$$

if $C = e^{\beta_0}$ and $n = \beta_1$. So really we are estimating a power of X while observing that the measurement variaibility increases inproportion to X^n.

So now let's fit a linear model for $\log(Y)$ and $\log(X)$:

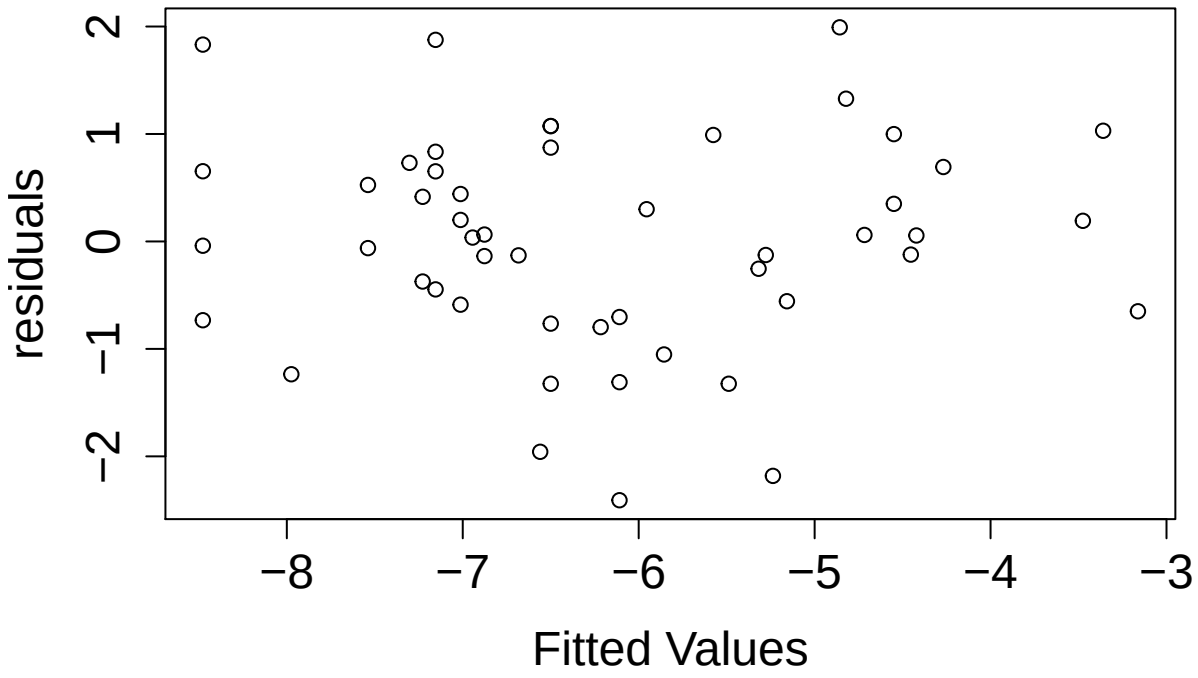
```
Wbmod = lm(log(Wb)~log(U),data=Wbdat)
summary(Wbmod)
```

```
##
## Call:
## lm(formula = log(Wb) ~ log(U), data = Wbdat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -2.40810 -0.63449  0.04588  0.68300  1.99257
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -12.0613     0.6377 -18.913  < 2e-16 ***
## log(U)        3.2623     0.3459   9.432 1.66e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.003 on 48 degrees of freedom
## Multiple R-squared:  0.6495, Adjusted R-squared:  0.6422
## F-statistic: 88.95 on 1 and 48 DF,  p-value: 1.66e-12
```

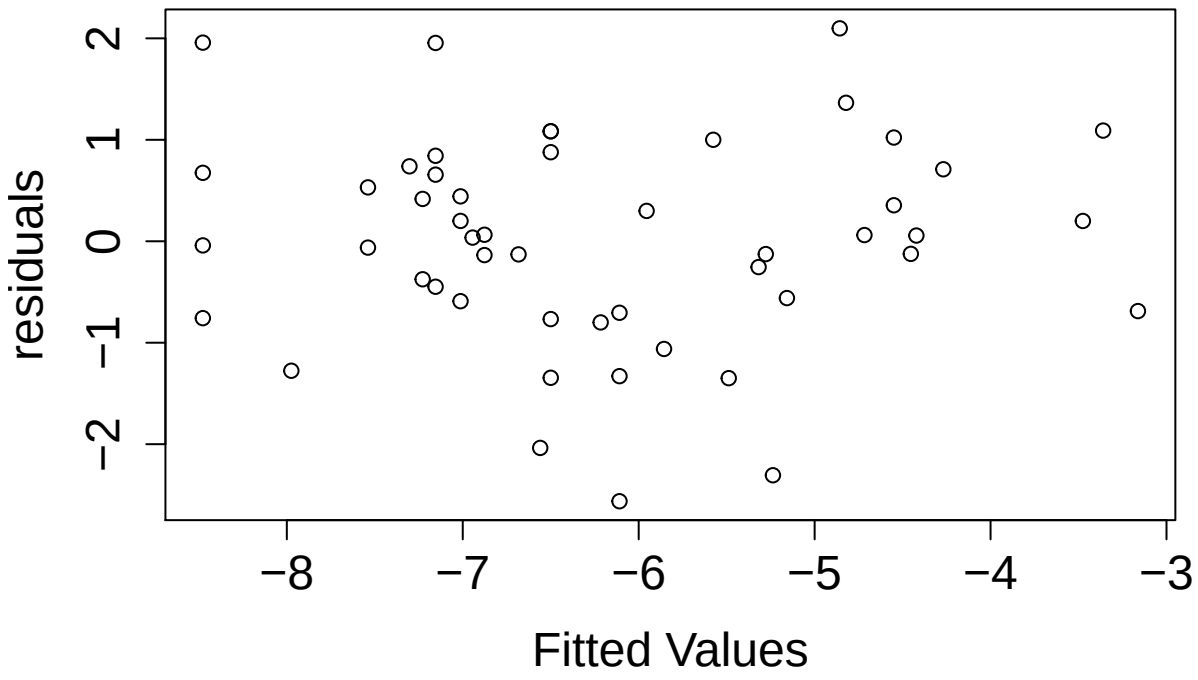
Let's do some due dilligence to look for further model issues. First we will plot residuals versus fitted values

```
plot(Wbmod$fit,Wbmod$res,xlab='Fitted Values',ylab='residuals',cex.lab=1.5,cex.axis=1.5)
```



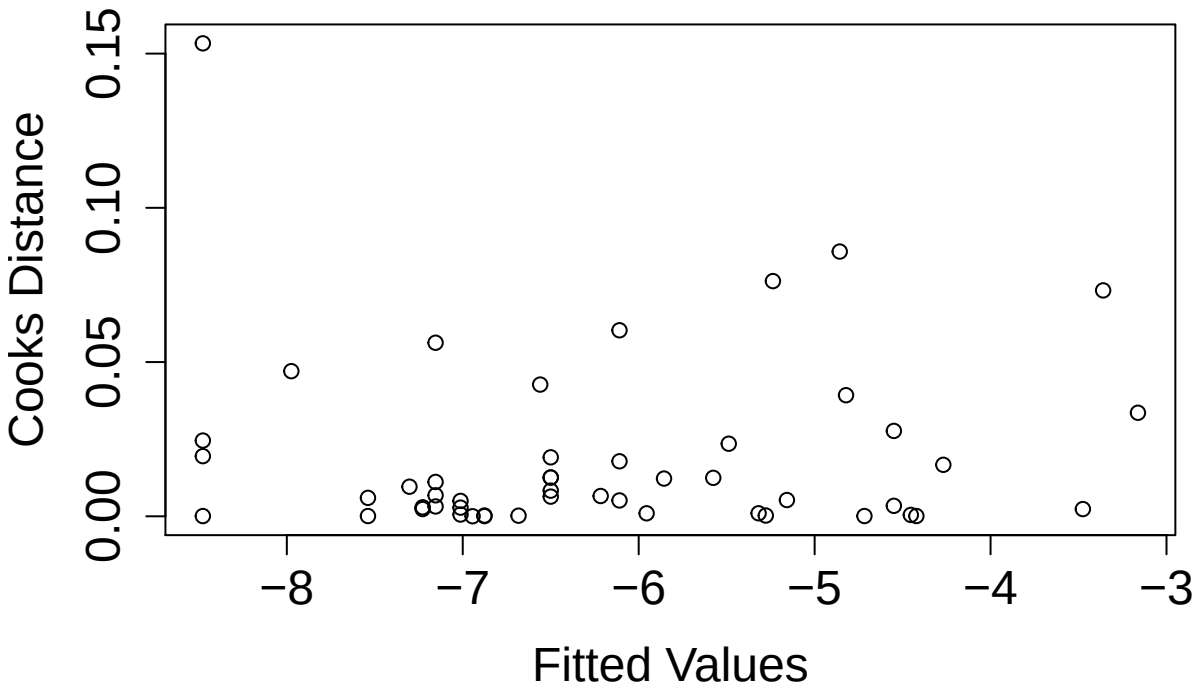
we should also extract studentized residuals and fit them

```
library(MASS)
sres = studres(Wbmod)
plot(Wbmod$fit,sres,xlab='Fitted Values',ylab='residuals',cex.lab=1.5,cex.axis=1.5)
```



There are a few values around 2, but nothing huge. Let's also look at Cook's distance

```
cdist = cooks.distance(Wbmod)
plot(Wbmod$fit,cdist,xlab='Fitted Values',ylab='Cooks Distance',cex.lab=1.5,cex.axis=1.5)
```



One large value, let's look at this

```
cdist
```

```
##            1            2            3            4            5
## 6.578160e-03 1.533232e-01 5.144510e-03 6.373869e-03 2.454938e-02
##            6            7            8            9           10
## 5.981618e-03 1.808996e-05 2.352300e-03 5.626836e-02 9.591973e-03
##           11           12           13           14           15
## 1.949720e-02 1.113599e-02 1.222724e-02 1.257358e-02 6.802007e-03
##           16           17           18           19           20
## 3.184934e-03 2.447365e-04 5.443811e-05 4.988177e-03 1.257358e-02
##           21           22           23           24           25
## 8.345758e-05 7.267612e-05 4.705417e-02 2.917577e-03 5.782049e-04
##           26           27           28           29           30
## 1.912033e-02 1.976972e-04 8.313110e-03 2.472316e-04 1.246494e-02
##           31           32           33           34           35
## 8.859683e-05 3.397414e-03 2.338328e-03 8.582468e-02 9.743071e-04
##           36           37           38           39           40
## 4.496537e-04 1.783209e-02 5.269845e-03 2.768044e-02 9.632496e-04
##           41           42           43           44           45
## 2.352849e-02 2.809328e-03 6.030435e-02 7.624934e-02 4.270659e-02
##           46           47           48           49           50
## 7.322869e-02 3.925868e-02 3.356429e-02 1.669071e-02 9.476934e-05
```

looks like it's observation 2; checking that this doesn't make a big difference

```
Wbmod2 = lm(log(Wb)~log(U),data=Wbdat[-2,])
summary(Wbmod2)
```

```
##
## Call:
## lm(formula = log(Wb) ~ log(U), data = Wbdat[-2, ])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.37282 -0.57782  0.04559  0.74050  1.96445
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -12.3980     0.6431  -19.28  < 2e-16 ***
## log(U)        3.4275     0.3465    9.89 4.56e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9745 on 47 degrees of freedom
## Multiple R-squared:  0.6755, Adjusted R-squared:  0.6686
## F-statistic: 97.82 on 1 and 47 DF,  p-value: 4.565e-13
```

Some differences in the value of β_1 but the conclusions remain the same.

Predicted values

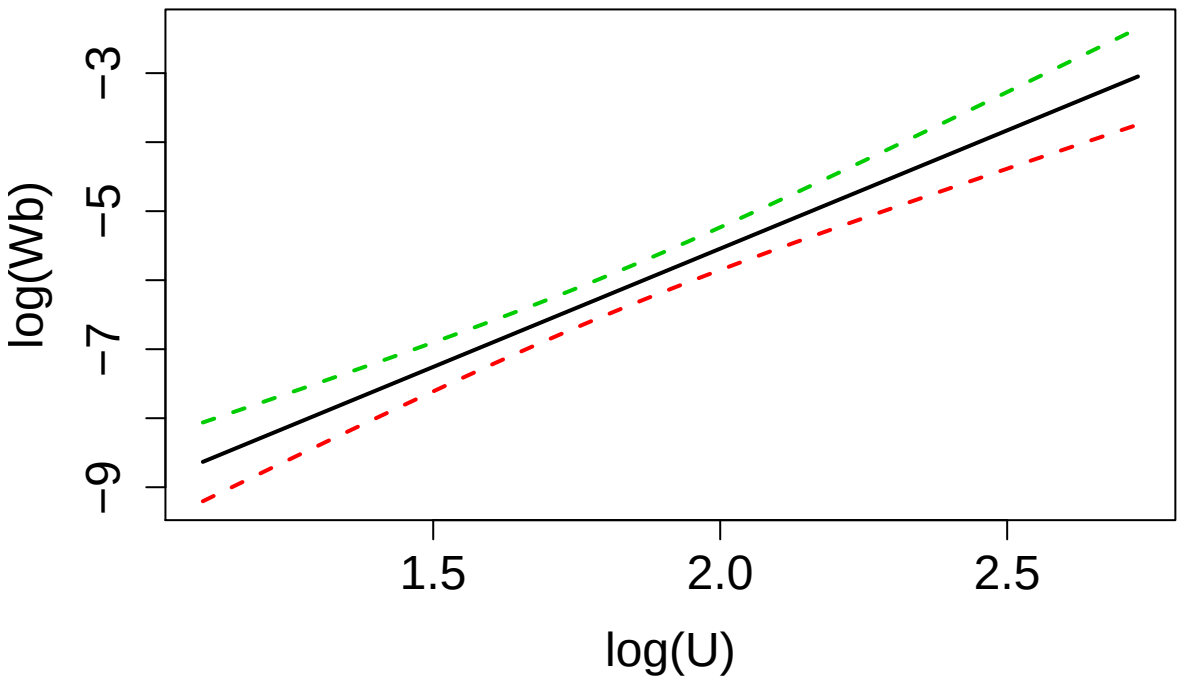
We can now make predictions for these data. I'll just use the values of U in the data set for the moment. In order to plot, I'll create a new data set

```
plotdat = data.frame(U = seq(min(Wbdat$U),max(Wbdat$U),len=101), Wb = rep(0,101))
```

This just places 101 values over the range of U – we fill in anything we want for Wb since we won't be using it.

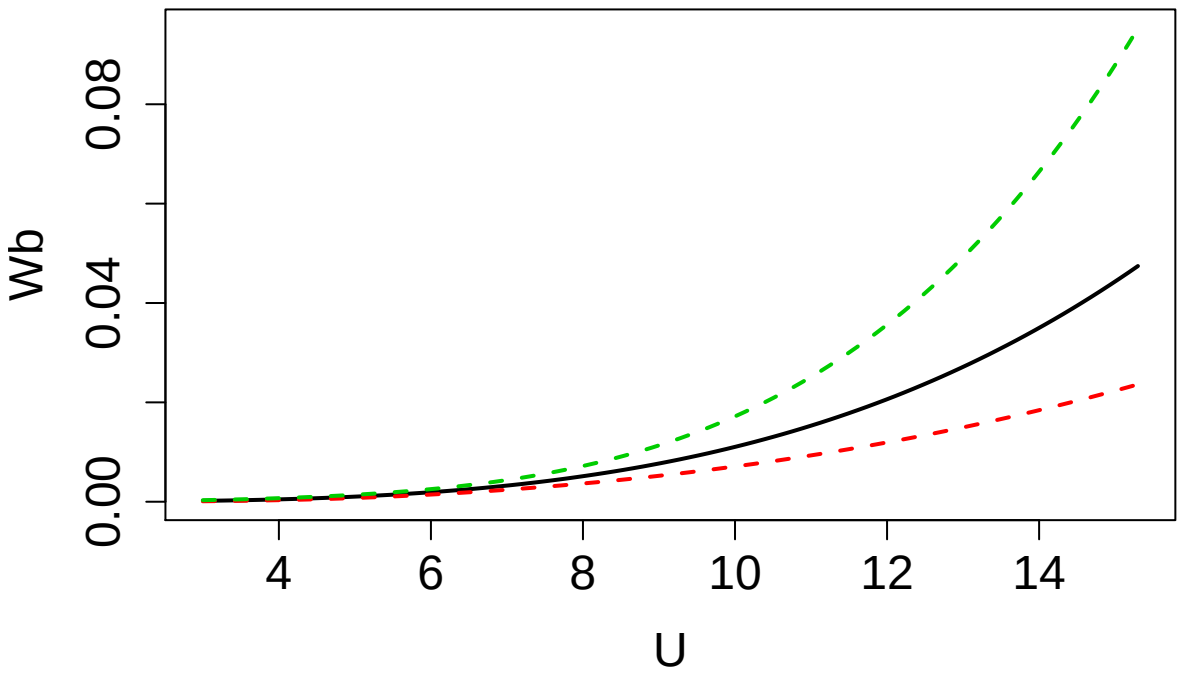
Now we'll predict and plot predictions and confidence intervals (the matplot function is handy for plotting multiple lines at once – it plots every column of a data frame or matrix).

```
preds.c = predict(Wbmod2,newdata=plotdat,interval="confidence")
matplot(log(plotdat$U),preds.c,type='l',lty=c(1,2,2),lwd=2,
         xlab='log(U)',ylab='log(Wb)',cex.lab=1.5,cex.axis=1.5)
```



Note that this is still plotting predictions of $\log(Wb)$ from $\log(U)$. We might want to put both of these back on the original scale. For this, the correct thing to do is to take the end points of the confidence intervals and then back-transform by the exponential function. This is easy in in this case, because we just exponentiate preds

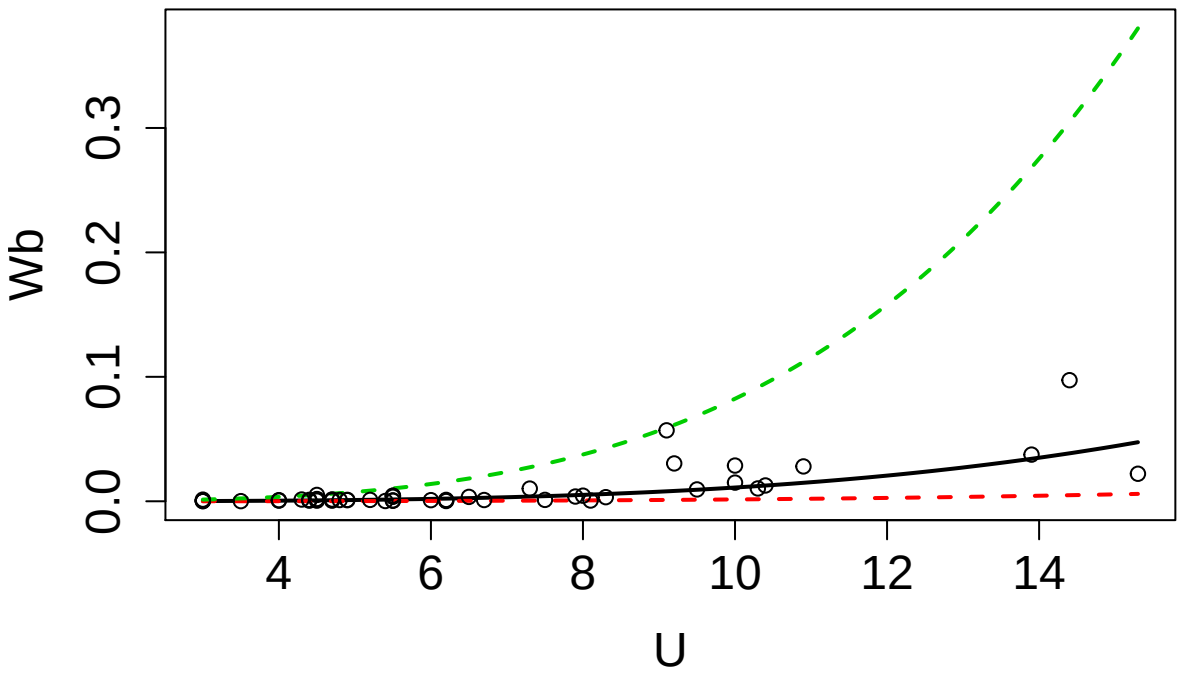
```
matplot(plotdat$U,exp(preds.c),type='l',lty=c(1,2,2),lwd=2,xlab='U',ylab='Wb',
        cex.lab=1.5,cex.axis=1.5)
```



Note that the confidence intervals become much wider at the high end of the graph. They also become assymetric since the exponential transform pushes high values up much faster than it changes low values.

The same thing applies to prediction intervals. If we add the original data to this graph, we see it describes its distribution pretty well

```
preds.p = predict(Wbmod2,newdata=plotdat,interval="prediction")
matplot(plotdat$U,exp(preds.p),type='l',lty=c(1,2,2),lwd=2,xlab='U',ylab='Wb',
        cex.lab=1.5,cex.axis=1.5)
points(Wbdat)
```



You can also experiment with not re-transforming from $\log(U)$ (but keeping the transformation of $\log(Wb)$) if you like.