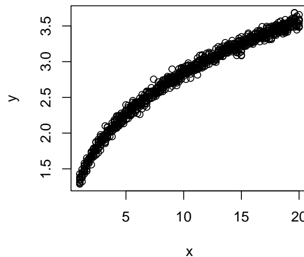
Quantitative Understanding in Biology
2.2 Fitting Model Parameters

Jason Banfelder

October 12th, 2023

Let us consider how we might fit a power law model to some data. We will begin by simulating the power law relationship $y = \alpha x^{\beta}$.

```
alpha.true <- 1.33
beta.true <- 0.33
d <- data.frame(x = rep(seq(1, 20, 0.2), 10))
d$y <- alpha.true * d$x ^ beta.true + rnorm(d$x, sd = 0.05)
plot(y ~ x, data = d)
```



Note that in this example, we've generated ten samples at each x value we consider. The motivation for doing this in this example will become clear in few moments; however, an important point to make is that if you have multiple data points like this, you should include all of them in your regression, as we do here. Some people mistakenly average the y values for each distinct x, and then regress over the averaged y values. This hides variation in the data, and can lead to erroneous conclusions.

With this dataset in hand, and pretending that we do not know the true values of α and β, we proceed to transform our model and data so it is amenable to linear regression. Our

model becomes...

$$\ln(y) = \ln(\alpha) + \beta \ln(x) \tag{1}$$

...and we transform our data accordingly. Keep in mind that the function for the natural logarithm in R is `log(x)`; to compute $\log_{10}(x)$, you would use `log(x, 10)`.

```
d$x.t <- log(d$x)
d$y.t <- log(d$y)
```

The regression is straightforward to perform in R...

```
m <- lm(y.t ~ x.t, data = d)
summary(m)

##
## Call:
## lm(formula = y.t ~ x.t, data = d)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.072782 -0.011778  0.000041  0.011793  0.072099
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.2880165  0.0019775   145.6   <2e-16 ***
## x.t         0.3286302  0.0008728   376.5   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01962 on 958 degrees of freedom
## Multiple R-squared:  0.9933,Adjusted R-squared:  0.9933
## F-statistic: 1.418e+05 on 1 and 958 DF,  p-value: < 2.2e-16
```

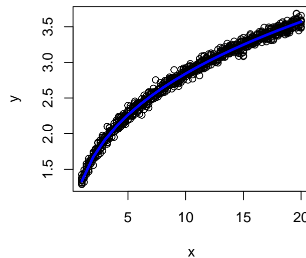
The estimate of β (which is the coefficient of the transformed x term) is 0.329, gratifyingly close to our true value of 0.33. Looking back at the transformed model, we see that the estimate of the intercept term is an estimate of $\ln(\alpha)$; to determine the estimate for α itself, we compute $\alpha = e^{\ln(\alpha)} = e^{0.288} = 1.334$. We have done a pretty good job of recovering the true values of both parameters.

Whenever we perform a regression, it is always useful to plot the regressed, best-fit curve to the data. The R function `predict` is useful for this task; you pass it a model and a dataframe containing the x-values for which you want to generate predictions. We note that the linear model predicts $\ln(y)$ from $\ln(x)$, so we must transform the x values going into the model, and untransform its results to make them suitable for plotting against our

© Copyright 2008, 2023; J Banfelder, Weill Cornell Medical College

original data.

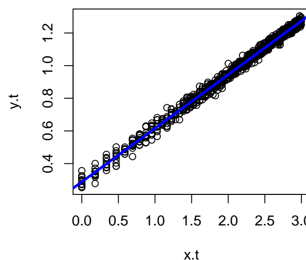
```
x <- 1:20
lines(x, exp(predict(m, newdata = data.frame(x.t = log(x)))),
      col = "blue", lwd = 3)
```



Not surprisingly, the curve passes through our data quite nicely.

It is also informative to plot the transformed data, and the fitted curve through it. We use a shortcut function, `abline`, which is suitable for plotting results only from straight-line regression.

```
plot(y.t ~ x.t, data = d); abline(m, col = "blue", lwd = 3)
```



Here we notice something interesting; the data are more scattered at lower values of $\ln(x)$ than at higher values of $\ln(x)$. This is a hint that there is a subtle problem we will encounter shortly.

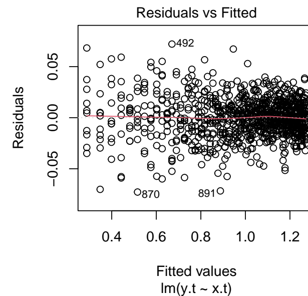
Whenever we are performing a regression, it is always a good idea to plot the regressed curve through your data whenever possible (this can be tricky when you have multiple explanatory variables). This is an obvious and powerful way to assess how well your model describes the data.

Additionally, it is important to critically evaluate the residuals that your fit produced. A

© Copyright 2008, 2023; J Banfelder, Weill Cornell Medical College

common technique is to plot residuals against predicted values. For linear fits, this can be quickly done with the command:

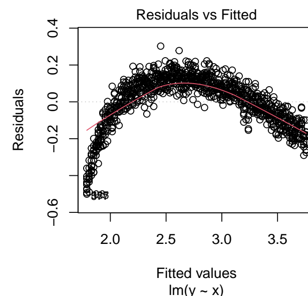
```
plot(m, which = 1)
```



You are looking for any pattern in the residuals (and hoping there will be none). To help, R draws a non-parametric best-fit line through this plot. If your fit is good, the line will hew close to the x-axis. If there is a pattern to the residuals, there is a strong indication that the model isn't telling the whole story. In our case, we see that the line does indeed hew close to the x-axis, but again we see that there is more spread in the lower values of $\ln(x)$. It goes without saying that you should also be cognizant of the magnitude of the residuals (i.e., pay attention to the relative scales of the x- and y-axes).

To demonstrate a poor fit, let's try a similar plot for a fit of a straight-line model to the original, untransformed data.

```
plot(lm(y ~ x, data = d), which = 1)
```



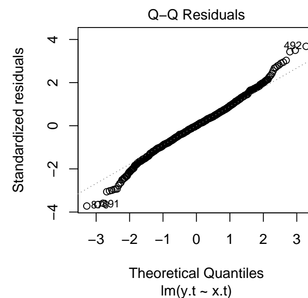
You probably didn't need this residual plot to figure out that the fit to the data was awful. In many cases, however, plots such as these can bring out subtle patterns in residuals that are not apparent when looking at a best-fit curve through scattered data.

A third technique to assess the quality of your model is to assess the distribution of the

© Copyright 2008, 2023; J Banfelder, Weill Cornell Medical College

residuals. Ideally, the residuals will be normally distributed around zero (if the mean is not around zero you would have seen this in the previous plot). For linear fits, R makes this exceptionally convenient:

```
plot(m, which = 2)
```



Here we see a non-trivial deviation from normality; another, perhaps not-so-subtle clue that something is up.

At this point, you might be wondering why we are so concerned about the scatter of the residuals when it is clear that the best-fit line does a very good job of predicting our data. After all, the plot of the regressed curve went through our data very well, and we did an excellent job of recovering the parameters that we know to be correct in our exercise.

The reason is that our statistical regression model can do more than just fit a curve and give us estimates of our parameters. It also gives us confidence intervals for those parameters, and for predicted values!

```
confint(m)

##                 2.5 %    97.5 %
## (Intercept) 0.2841358 0.2918972
## x.t         0.3269174 0.3303431
```

The CI for β can be interpreted directly. Again, for α we must untransform each end of the interval.

```
exp(confint(m)[1, ])

##    2.5 %   97.5 %
## 1.328613 1.338965
```

Note that the CI for this transformed parameter is not symmetric; that is OK and to be expected when fitting involves non-linear transformations.

So not only do we get our best-fit parameters, but we get an estimate of their uncertainty.

© *Copyright 2008, 2023; J Banfelder, Weill Cornell Medical College*

However, these CIs are only valid if the model meets our expectation of normally distributed residuals; this is formally known as homoscedasticity. In our case, the heteroscedasticity is not too bad and our CIs are probably usable as a rough and somewhat optimistic estimate of the uncertainty in the parameters; the correct CIs are probably a bit wider than those reported here.

In addition to reporting CIs of our model parameters, we can also use the variance information contained in our model to estimate the CIs of new values predicted by the model. For example, if we wish to know the 95% CI for measuring a new value of y at $x = 10$, we could compute:

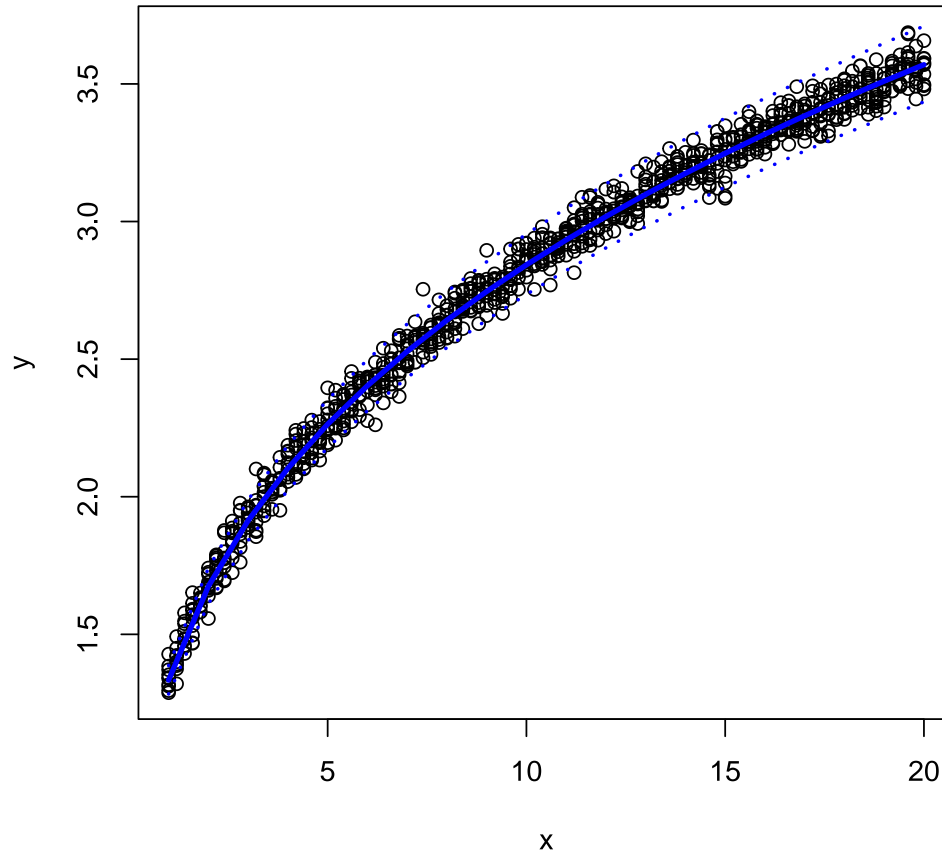
```
exp(predict(m, newdata = data.frame(x.t = log(10)), interval = "p"))

##       fit      lwr      upr
## 1 2.84259 2.735188 2.954209
```

This tells us that if we sample a new point at $x = 10$, there is a 95% chance that the y value will be between 2.735 and 2.954. Repeated application of this reasoning allows us to compute and plot an error band around our best fit curve. . .

```
plot(y ~ x, data = d)
x <- 1:20
lines(x, exp(predict(m, newdata = data.frame(x.t = log(x)))),
      col = "blue", lwd = 3)
eb <- exp(predict(m, newdata = data.frame(x.t = log(x)), interval = "p"))
lines(x, eb[, 2], col = "blue", lty = 3, lwd = 2)
lines(x, eb[, 3], col = "blue", lty = 3, lwd = 2)
```

© Copyright 2008, 2023; J Banfelder, Weill Cornell Medical College



Now we can see the deleterious effect of heteroscedasticity in our transformed model. We see that the error band predicted by the transformed fit is thicker at high values of x and thinner at low values of x. Note that when we refer to the thickness of error band here, we refer to its height (the vertical space between the blue dashed lines in the plot). Don't make the mistake of interpreting the thickness as the width normal to the regressed curve. In fact, this thickness varies by a factor of 2.67 across the range of values that we worked with:

```
(eb[length(x), 3] - eb[length(x), 2]) / (eb[1, 3] - eb[1, 2])

## [1] 2.666225
```

However, we know that the error band should be a constant thickness (look back to how

© Copyright 2008, 2023; J Banfelder, Weill Cornell Medical College

we generated the simulated data):

```
d$y <- alpha.true * d$x ^ beta.true + rnorm(d$x, sd = 0.05)
```

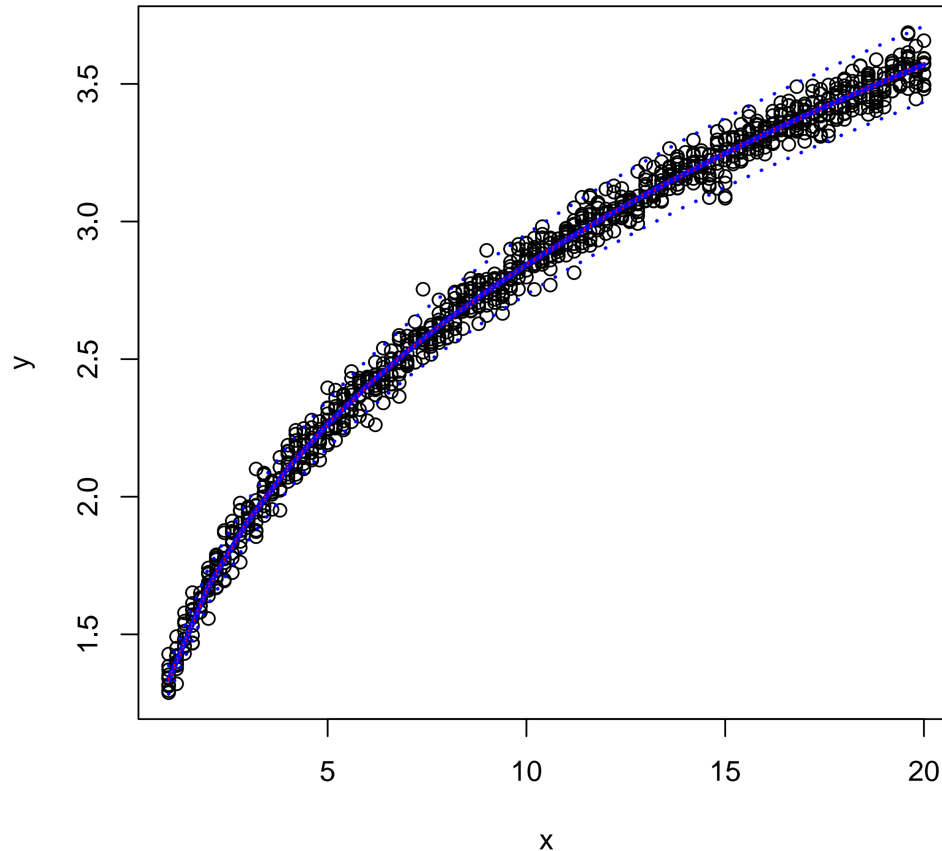
We used the normal distribution with an SD of 0.05 on every term, so we know that the thickness of our error band should be constant. [Challenge question: what should the thickness of the error band be?]

The error band that we drew is analogous to the standard deviation of the distribution. In other words, it answers (approximately, in this case) the question: given a value for x, what is the range in which we expect 95% of y values will fall.

A different error band can also be computed that is analogous to the SEM, and answers this question: given a value for x, what is the 95% CI for the mean of y at this point. To compute and plot this error band, specify `interval="c"` when you use the predict function.

```
eb.c <- exp(predict(m, newdata=data.frame(x.t = log(x)), interval = "c"))
lines(x, eb.c[, 2], col = "red", lty = 3)
lines(x, eb.c[, 3], col = "red", lty = 3)
```

© Copyright 2008, 2023; J Banfelder, Weill Cornell Medical College



As can be seen, there is not much uncertainty in the best-fit curve; we had quite a bit of data to work with: 10 points for each x value. You should be able to predict what would happen to the blue and the red bands if we repeated this exercise with only one point per x value.

In this example, heteroscedasticity was introduced by the logarithmic transformation that we performed to enable us to do a linear regression. This is not always going to be the case; sometimes a variable transformation will fix a heteroscedasticity problem. If it does, the transformation is encouraged as good practice. Furthermore, this may be a hint that the "natural variables" for the system are the transformed ones, not the originally measured ones.

© Copyright 2008, 2023; J Banfelder, Weill Cornell Medical College

The alternative to transforming the model to make it amenable to linear regression is to perform a non-linear regression. This is a computationally more intensive procedure that usually involves an iterative optimization. Fortunately, the computer will take care of most of the details (although things can and do go wrong). As an exercise you can perform a non-linear regression of the original model in R; we will investigate an alternative model just to mix things up a bit.

A simple inspection of the plot of our synthesized power law data might suggest alternative models. One possible example is an exponential approach to an asymptote following the mathematical form:

$$y = \alpha \left(1 - \beta e^{-\gamma x}\right) \tag{2}$$

Fitting this model to our data using non-linear regression in R is similar to the linear regression case:

```
em <- nls(y ~ alpha * (1 - beta * exp(-gamma * x)),
          start = list(alpha = 5, beta = 1, gamma = 1),
          data = d)
summary(em)

##
## Formula: y ~ alpha * (1 - beta * exp(-gamma * x))
##
## Parameters:
##       Estimate Std. Error t value Pr(>|t|)
## alpha 3.963700   0.017348  228.49   <2e-16 ***
## beta  0.691045   0.001618  427.06   <2e-16 ***
## gamma 0.091155   0.001352   67.43   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05687 on 957 degrees of freedom
##
## Number of iterations to convergence: 6
## Achieved convergence tolerance: 4.589e-06
```

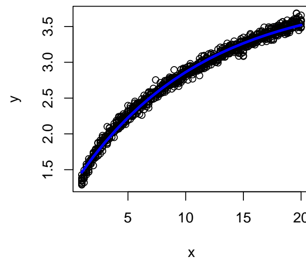
When performing a non-linear regression, we use the `nls` (non-linear least squares) function in R. The model formula doesn't do any of the odd things that it would in the `lm` function; here all of the terms are interpreted as written. Additionally, since non-linear regression implies an iterative optimization, we need to specify a starting point for the parameters of the model. R will repeatedly adjust these parameters to drive the SSQ of the residuals to a minimum. While a very accurate guess is not required, a reasonable starting point is helpful. All of the dangers of numerical optimizations apply here: optimizations can

© *Copyright 2008, 2023; J Banfelder, Weill Cornell Medical College*

wander off into irrelevant parameter space, find a local minimum instead of the global optimum, fail to converge, etc.

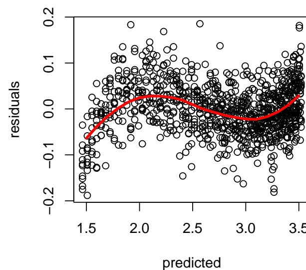
Assuming all goes well with our non-linear regression, we begin again by plotting the regressed curve against our data.

```
plot(y ~ x, data = d)
lines(x, predict(em, newdata = data.frame(x = x)), col = "blue", lwd = 3)
```



The curve passes through the data, but there are clearly systematic deviations. A plot of residuals against predicted values will make this quite explicit. Sadly, R does not have an automated plotting routine for non-linear models, so we'll have to do the work ourselves.

```
r <- data.frame(residuals = residuals(em), predicted = predict(em))
plot(residuals ~ predicted, data = r)
fit <- loess(residuals ~ predicted, data = r)
x <- seq(1.5, 3.5, 0.1)
lines(x, predict(fit, newdata = data.frame(predicted = x)),
      col = "red", lwd = 3)
```



Here we see a clear pattern in the residuals. This is an indication that the model is not explaining the data well. While we could probably use the pattern detected in the residuals

to inform additional terms in the model that would result in a better fit, it is also good practice to reconsider the system under study and think about extended or alternative mechanisms and the models they would imply.

In the example above, we introduced the `loess` function. This is yet another fitting model that R provides. It is useful for smoothing data, and handy when all you want is 'artistic' curve fitting. You can also produce QQ plots of the residuals from a non-linear regression; recall the `qqnorm` and `qqline` functions. We won't do that here because we've already rejected this model based on the previous plot.

The `confint` function also works for models returned from non-linear regression. Again, we don't do this here because the model has already been rejected. However, if you choose to do the exercise of performing non-linear regression on the power law model, you'll want to use the results of this function. These would be your best estimates for the model parameters, as the plots of residuals should confirm the model as being appropriate to describe this data.

Sadly, while the `predict` function works for model objects from non-linear regression, the interval argument is not supported. So while you can report CIs for your model parameters, you can't easily work up error bands for your plots. Hopefully, R will gain this ability soon.

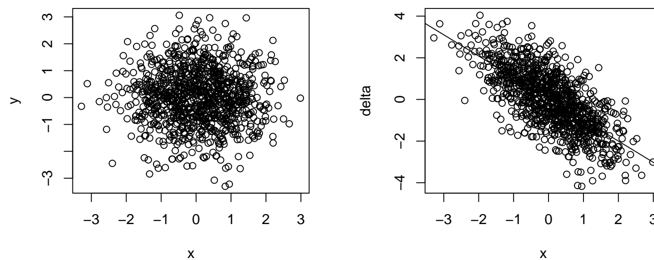
Danger: The Regression Fallacy

In regression studies, you need to be careful that x and y represent separate measurements. Here is an example of how you can get into trouble.

```
trap <- data.frame(x = rnorm(1000), y = rnorm(1000))
trap$delta <- trap$y - trap$x
plot(y ~ x, data = trap)
plot(delta ~ x, data = trap)
fallacy <- lm(delta ~ x, data = trap)
abline(fallacy)
summary(fallacy)
```

```
##
## Call:
## lm(formula = delta ~ x, data = trap)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.3251 -0.6840 -0.0026  0.6514  3.0074
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.04236    0.03250   1.303    0.193
## x           -1.02072    0.03286 -31.058   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.026 on 998 degrees of freedom
## Multiple R-squared:  0.4915,Adjusted R-squared:  0.491
## F-statistic: 964.6 on 1 and 998 DF,  p-value: < 2.2e-16
```



It seems that there is a strong relationship at work here, but it is all a fallacy. The x and y values are completely independent. The regression was Δ against x, and Δ is not a separate measurement from x. This phenomenon is often referred to as 'regression to the mean'.

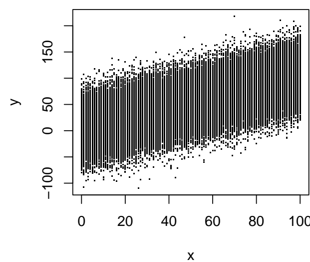
R^2 is not the best measure of what has been achieved in a regression

Consider this very artificial system.

```
x <- rep(0:100, 1000)
y <- x + 2 + rnorm(x, sd = 30)
plot(y ~ x, pch = '.')
m <- lm(y ~ x)
summary(m)
```

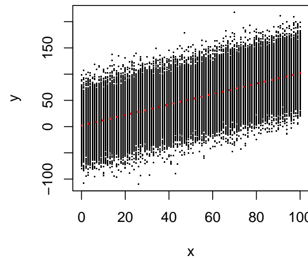
```
##
## Call:
```

```
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -137.494  -20.165    0.007   20.174  145.834
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.166229   0.187140   11.57   <2e-16 ***
## x           0.996973   0.003233  308.35   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 29.96 on 100998 degrees of freedom
## Multiple R-squared:  0.4849,Adjusted R-squared:  0.4849
## F-statistic: 9.508e+04 on 1 and 100998 DF,  p-value: < 2.2e-16
```



Notice that r^2 appears quite poor. However...

```
eb <- predict(m, newdata = data.frame(x = 0:100), interval = "c")
lines(0:100, eb[, 2], col = "red", lty = 3, lwd = 2)
lines(0:100, eb[, 3], col = "red", lty = 3, lwd = 2)
```



You can see that we have done a rather good job of determining the underlying parameters and quantifying the variance in our system. Because the system contains so much noise, it is not possible to have precise predictive power. However, we have recovered the underlying model quite well.

Exercise

For the data generated on page 1, perform a non-linear regression to the model $y = \alpha x^{\beta}$. Prepare all appropriate plots, and estimate the 95% CI of the model parameters. How well does the model explain the data?

Compute the average y value at each x value for the generated data. Using these averages, perform the same regression, preparing similar plots and reporting the parameters' 95% CIs. How do the results differ?

© Copyright 2008, 2023; J Banfelder, Weill Cornell Medical College