Using R for Linear Regression

In the following handout words and symbols in **bold** are R functions and words and symbols in *italics* are entries supplied by the user; *underlined* words and symbols are optional entries (all current as of version R-2.4.1). Sample texts from an R session are highlighted with gray shading.

Suppose we prepare a calibration curve using four external standards and a reference, obtaining the data shown here:

```
> conc
[1] 0 10 20 30 40 50

> signal
[1] 4 22 44 60 82
```

The expected model for the data is

\[
\text{signal} = \beta_0 + \beta_1 \times \text{conc}
\]

where \(\beta_0\) is the theoretical y-intercept and \(\beta_1\) is the theoretical slope. The goal of a linear regression is to find the best estimates for \(\beta_0\) and \(\beta_1\) by minimizing the residual error between the experimental and predicted signal. The final model is

\[
\text{signal} = b_0 + b_1 \times \text{conc} + e
\]

where \(b_0\) and \(b_1\) are the estimates for \(\beta_0\) and \(\beta_1\) and \(e\) is the residual error.

Defining Models in R

To complete a linear regression using R it is first necessary to understand the syntax for defining models. Let’s assume that the dependent variable being modeled is \(Y\) and that \(A\), \(B\) and \(C\) are independent variables that might affect \(Y\). The general format for a linear model is

\[
\text{response} \sim \text{op1 term1 op2 term2 op3 term3...}
\]

---

1. When discussing models, the term ‘linear’ does not mean a straight-line. Instead, a linear model contains additive terms, each containing a single multiplicative parameter; thus, the equations

\[
\begin{align*}
    y &= \beta_0 + \beta_1 x \\
    y &= \beta_0 + \beta_1 x_1 + \beta_2 x_2 \\
    y &= \beta_0 + \beta_{11} x^2 \\
    y &= \beta_0 + \beta_1 x_1 + \beta_2 \log(x_2)
\end{align*}
\]

are linear models. The equation \(y = \alpha x^\beta\), however, is not a linear model.
where *term* is an object or a sequence of objects and *op* is an operator, such as a + or a −, that indicates how the term that follows is to be included in the model. The table below provides some useful examples. Note that the mathematical symbols used to define models do not have their normal meanings!

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Model</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y ~ A</td>
<td>Y = β₀ + β₁A</td>
<td>Straight-line with an implicit y-intercept</td>
</tr>
<tr>
<td>Y ~ -1 + A</td>
<td>Y = β₁A</td>
<td>Straight-line with no y-intercept; that is, a fit forced through (0,0)</td>
</tr>
<tr>
<td>Y ~ A + I(A^2)</td>
<td>Y = β₀ + β₁A + β₂A²</td>
<td>Polynomial model; note that the identity function I( ) allows terms in the model to include normal mathematical symbols.</td>
</tr>
<tr>
<td>Y ~ A + B</td>
<td>Y = β₀ + β₁A + β₂B</td>
<td>A first-order model in A and B without interaction terms.</td>
</tr>
<tr>
<td>Y ~ A:B</td>
<td>Y = β₀ + β₁AB</td>
<td>A model containing only first-order interactions between A and B.</td>
</tr>
<tr>
<td>Y ~ A*B</td>
<td>Y = β₀ + β₁A + β₂B + β₃AB</td>
<td>A full first-order model with a term; an equivalent code is Y ~ A + B + A:B.</td>
</tr>
<tr>
<td>Y ~ (A + B + C)^2</td>
<td>Y = β₀ + β₁A + β₂B + β₃C + β₄AB + β₅AC + β₆AC</td>
<td>A model including all first-order effects and interactions up to the n&lt;sup&gt;th&lt;/sup&gt; order, where n is given by ( )^n. An equivalent code in this case is Y ~ A<em>B</em>C – A:B:C.</td>
</tr>
</tbody>
</table>

### Completing a Regression Analysis

The basic syntax for a regression analysis in R is

```
lm(Y ~ model)
```

where *Y* is the object containing the dependent variable to be predicted and *model* is the formula for the chosen mathematical model. The command `lm()` provides the model’s coefficients but no further statistical information; thus

```
> lm(signal ~ conc)

Call:
  lm(formula = signal ~ conc)

Coefficients:
 (Intercept)         conc
      3.60        1.94
```
To obtain more useful information, and to obtain access to many more useful functions for manipulating the data, it is best to create an object that contains the command for the model

```r
> lm.r = lm(signal ~ conc)
```

This object can then be used as an argument for other commands. To obtain a more complete statistical summary of the model, for example, we use the `summary()` command.

```r
> summary(lm.r)

Call:
  lm(formula = signal ~ conc)

Residuals:
   1       2      3      4      5
  0.4  -1.0   1.6  -1.8    0.8

Coefficients:
            Estimate Std. Error   t value   Pr(>|t|)
(Intercept)   3.6000     1.2329   2.920     0.0615 .
conc          1.9400     0.0503   38.543 3.84e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.592 on 3 degrees of freedom
Multiple R-Squared: 0.998, Adjusted R-squared: 0.9973
F-statistic: 1486 on 1 and 3 DF, p-value: 3.842e-05
```

The section of output labeled ‘Residuals’ gives the difference between the experimental and predicted signals. Estimates for the model’s coefficients are provided along with their standard deviations (‘Std Error’), and a t-value and probability for a null hypothesis that the coefficients have values of zero. In this case, for example, we see that there is no evidence that the intercept ($\beta_0$) is different from zero and strong evidence that the slope ($\beta_1$) is significantly different than zero. At the bottom of the table we find the standard deviation about the regression ($s_r$ or residual standard error), the correlation coefficient and an F-test result on the null hypothesis that the $MS_{reg}/MS_{res}$ is 1.

Other useful commands are shown below:

```r
> coef(lm.r)  # gives the model’s coefficients

(Intercept) conc
  3.69   1.94
```
Evaluating the Results of a Linear Regression

Before accepting the result of a linear regression it is important to evaluate its suitability at explaining the data. One of the many ways to do this is to visually examine the residuals. If the model is appropriate, then the residual errors should be random and normally distributed. In addition, removing one case should not significantly impact the model’s suitability. R provides four graphical approaches for evaluating a model using the `plot()` command.

```
> resid(lm.r) # gives the residual errors in Y
  1  2  3  4  5
  0.4 -1.0 1.6 -1.8  0.8

> fitted(lm.r) # gives the predicted values for Y
  1  2  3  4  5
  3.6 23.0 42.4 61.8 81.2
```
The plot in the upper left shows the residual errors plotted versus their fitted values. The residuals should be randomly distributed around the horizontal line representing a residual error of zero; that is, there should not be a distinct trend in the distribution of points. The plot in the lower left is a standard Q-Q plot, which should suggest that the residual errors are normally distributed. The scale-location plot in the upper right shows the square root of the standardized residuals (sort of a square root of relative error) as a function of the fitted values. Again, there should be no obvious trend in this plot. Finally, the plot in the lower right shows each points leverage, which is a measure of its importance in determining the regression result. Superimposed on the plot are contour lines for the Cook’s distance, which is another measure of the importance of each observation to the regression. Smaller distances means that removing the observation has little affect on the regression results. Distances larger than 1 are suspicious and suggest the presence of a possible outlier or a poor model.

Sometimes a model has known values for one or more of its parameters. For example, suppose we know that the true model relating the signal and concentration is

\[
\text{signal} = 3.00 \times \text{conc}
\]

Our regression model is

\[
\text{signal} = 3.60 + 1.94 \times \text{conc}
\]

We can use a standard t-test to evaluate the slope and intercept. The confidence interval for each is

\[
\beta_0 = b_0 \pm ts_{b_0} \quad \beta_1 = b_1 \pm ts_{b_1}
\]

where \(s_{b_0}\) and \(s_{b_1}\) are the standard errors for the intercept and slope, respectively. To determine if there is a significant difference between the expected (\(\beta\)) and calculated (\(b\)) values we calculate \(t\) and compare it to its standard value for the correct number of degrees of freedom, which in this case is 3 (see earlier summary).

\[
> \text{tb1} = \text{abs}((3.00 - 1.94)/0.05033); \text{tb1} \quad \# \text{calculate absolute value of t}
\]

\[
[1] 21.06100
\]

\[
> \text{pt(tb1, 3, lower.tail = FALSE)} \quad \# \text{calculate probability for t}
\]

\[
[1] 0.0001170821 \quad \# \text{double this value for a two-tailed evaluation; difference is significant at } p = 0.05
\]

\[
> \text{tb0}=\text{abs}((0-3.60)/1.23288);\text{tb0} \quad \# \text{calculate absolute value of t}
\]

\[
[1] 2.919992
\]
Here we calculate the absolute value of t using the calculated values and standard errors from our earlier summary of results. The command

\[ \text{pt(value, degrees of freedom, lower.tail = FALSE)} \]

returns the one-tailed probability that there is no difference between the expected and calculated values. In this example, we see that there is evidence that the calculated slope of 1.94 is significantly different than the expected value of 3.00. The expected intercept of 0, however, is not significantly different than the calculated value of 3.60. Note that the larger standard deviation for the intercept makes it more difficult to show that there is a significant difference between the experimental and theoretical values.

Using the Results of a Regression to Make Predictions

The purpose of a regression analysis, of course, is to develop a model that can be used to predict the results of future experiments. In our example, for instance, the calibration equation

\[ \text{signal = 3.60 + 1.94} \times \text{conc} \]

Because there is uncertainty in both the calculated slope and intercept, there will be uncertainty in the calculated signals.

Suppose we wish to predict the signal for concentrations of 0.05, 0.15, 0.25, 0.35 and 0.45 along with the confidence interval for each. We can use the `predict()` command to do this; the syntax is

\[ \text{predict(model, data.frame(pred = new pred), level = 0.95, interval = “confidence”)} \]

where `pred` is the object containing the original independent variables and `new pred` is the object containing the new values for which predictions are desired, and `level` is the desired confidence level.

```r
> newconc=c(5,15,25,35,45);newconc
[1] 5  15  25  35  45

> predict(lm.result,data.frame(conc = newconc), level = 0.9, interval = "confidence")
```
where ‘lwr’ is the lower limit of the confidence interval and ‘upr’ is the upper limit of the confidence interval. R does not contain a feature for finding the confidence intervals for predicted values of the independent variable for specified values of dependent variables, a common desire in chemistry. Too bad.

Adding Regression Lines to Plots

For straight-lines this is easy to accomplish.

```r
> plot(conc, signal)
> abline(lm.r)
```

gives the following plot.

For data that does not follow a straight-line we must be more creative.
> height = c(100, 200, 300, 450, 600, 800, 1000)
> distance = c(253, 337, 395, 451, 534, 574)  # data from Galileo

> lm.r = lm(distance ~ height + I(height^2)); lm.r  # a quadratic model

Call:
  lm(formula = distance ~ height + I(height^2))

Coefficients:
  (Intercept)          height    I(height^2)
  200.211950     0.706182    -0.000341

> newh = seq(100, 1000, 10); newh                   # create heights for predictions
[1] 100 110 120 130 140 150 160 170 180 190 200
[12] 210 220 230 240 250 260 270 280 290 300 310
[23] 320 330 340 350 360 370 380 390 400 410 420
[34] 430 440 450 460 470 480 490 500 510 520 530
[45] 540 550 560 570 580 590 600 610 620 630 640
[56] 650 660 670 680 690 700 710 720 730 740 750
[67] 760 770 780 790 800 810 820 830 840 850 860
[78] 870 880 890 900 910 920 930 940 950 960 970
[89] 980 990 1000

> fit = 200.211950 + 0.706182*newh - 0.000341*newh^2; fit  # calculate distance for new heights using model

[17] 360.7677 366.0223 371.2085 376.3266 381.3766 386.3583 391.2718 396.1171
[25] 400.8942 405.6032 410.2439 414.8164 419.3207 423.7568 428.1248 432.4245
[33] 436.6560 440.8193 444.9144 448.9414 452.9001 456.7906 460.6129 464.3670
[41] 468.0530 471.6707 475.2202 478.7015 482.1146 485.4595 488.7363 491.9448
[49] 495.0851 498.1572 501.1612 504.0969 506.9644 509.7637 512.4948 515.1578
[57] 517.7525 520.2790 522.7373 525.1274 527.4493 529.7031 531.8886 534.0059
[65] 536.0550 538.0359 539.9487 541.7932 543.5695 545.2776 546.9176 548.4893
[73] 549.9928 551.4281 552.7952 554.0941 555.3249 556.4874 557.5817 558.6078
[81] 559.5657 560.4555 561.2770 562.0303 562.7154 563.3323 563.8811 564.3616
[89] 564.7739 565.1180 565.3940

> plot(height, distance)                        # original data
> lines(newh, fit, lty=1)                       # display best fit
The resulting plot is shown here.