Curve Fitting with the Linear Model

lm(formula, weights, …)

Inputs:

Although there are far more input parameters than listed above, these will be the two used for 99% of your curve fitting problems.

`formula` - the relationship between the measured value and the predictor variable(s). For un-weighted, linear fitting, $y = a_0 + a_1x$, the call would be, `lm(y ~ x)`. The intercept is implied. For $y = a_1x$ the call would be, `lm(y ~ x-1)`, where "`-1`" means remove the intercept. For a quadratic equation one has to first define something like, `x2 <- x^2`, then use, `lm(y ~ x+x2)`, or without an intercept, `lm(y ~ x+x2-1)`. An alternative approach would define, `X <- cbind(x,x^2)`, and then call, `lm(y ~ X)`.

`weights` - the weighting for each point given by $1/\sigma_i^2$. One common weight occurs with fitting counting data where, `w[i] <- 1/y[i]`, making the call, `lm(y ~ x,weights=w)`.

Outputs:

The outputs are usually obtained by code like, `fit <- lm(y ~ x+x2)`. The model object "fit" is a list with many components. The most useful are: `$coefficients`, `$residuals`, `$fitted.values`, and `$df.residual` (the residual degrees of freedom).

Functions that extract or generate model information:

`coef(fit)` extracts the coefficients, `residuals(fit)` extracts the residuals, `fitted(fit)` extracts the fitted values, and `vcov(fit)` extracts the variance-covariance matrix.

`summary(fit)` generates a summary of the fit. An example for a quadratic fit is shown below. Use `?summary.lm()` to see a complete listing of available output parameters.

```
Call:
lm(formula = y ~ x + x2)
Residuals:
    Min      1Q  Median      3Q     Max
-1.0820 -0.5709  0.1225  0.3071  1.5428

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.96289    0.66141   1.456  0.18354
x            4.66772    0.30773  15.168 3.53e-07 ***
x2          -0.11140    0.02964  -3.758  0.00556 **
---
Signif. codes: 0‘***’ 0.001‘**’ 0.01‘*’ 0.05‘.’ 0.1‘ ’ 1
```

```
Residual standard error: 0.8682 on 8 degrees of freedom
Multiple R-squared: 0.9957,     Adjusted R-squared: 0.9946
F-statistic: 928.6 on 2 and 8 DF,  p-value: 3.384e-10
```

The order statistics description of the residuals is found under `Residuals`.

`Coefficients` is a matrix containing the coefficient name, the value, and the associated error. The t-values are for the null hypothesis, H_0 = "the coefficient is zero". Thus a large number signifies that the coefficient is required for the best fit to the data. A low t-value means that the coefficient may not be required. In the above example, the intercept is not all that important for explaining the data.

`Residual standard error` is the error of the fit.

`Multiple R-squared` is the multiple R-squared.

`F-statistic` tests the hypothesis, H_0 = "there is no relationship between *x* and *y*". Again, a large value means that a statistically significant relationship exists.

As an example, the coefficient errors would be obtained by, `sumry <- summary(fit)`, then, `coeffError <- sumry$coefficients[,2]`, to extract the second column of the coefficient matrix.

`anova(fit)` presents an analysis of variance view of the fit. The output corresponding to the above summary is shown below.

```
Analysis of Variance Table
Response: y
              Df  Sum Sq Mean Sq  F value Pr(>F)
     x         1 1389.21 1389.21 1843.158 9.554e-11 ***
     x2        1   10.65 10.65   14.126   0.005556 **
Residuals      8    6.03 0.75
```

For this example, the total sum of squares,

$$SSTO = \sum_i \left(y_i - \bar{y} \right)^2 = 1405.888$$

which is the total of the column labeled Sum Sq. The residuals sum of squares is given by,

$$SSE = \sum_i \left(y_i - \hat{y}_i \right)^2 = 6.03$$

and comes from the error of the fit given above (8×0.8682^2). Again for the example, the sum of squares due to *x* is 1389.21, while that for x^2 is 10.65. Thus the quadratic term doesn't reduce the total sum of squares nearly as much as the linear term. The column labeled `F value` is the F-test for the H_0 = "the coefficient of this term is zero". The

table value is $F_{table}(0.975;1,8) = 7.57$, thus the test fails in both cases, i.e. neither coefficient can be dropped.

`predict(fit, newdata, interval = c("none", "confidence", "prediction"))` generates fits to the least-squares coefficients. For example with a quadratic equation,

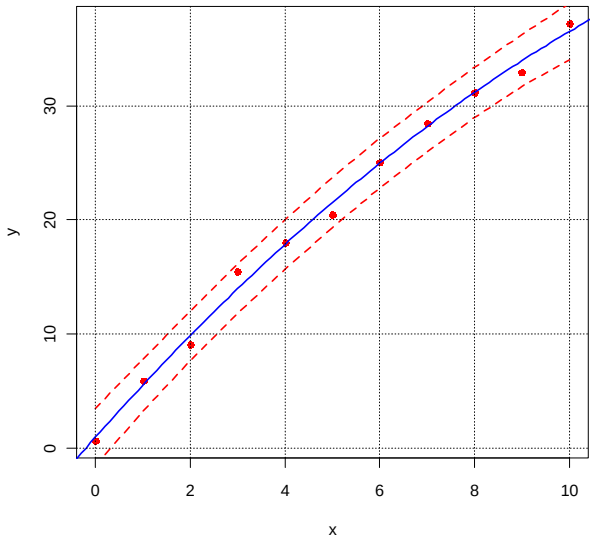
```
fit <- lm(y ~ x + x2)
plot(x,y)
lines(x,predict(fit))
```

creates a plot of (x,y) points along with a regression line that extends over the range of the x-variable. If a regression line is needed that has a finer resolution than the input x-spacing and/or extends beyond the range of the x-variable (0:10 in this case) use code like the following.

```
fit <- lm(y ~ x + x2)
plot(x,y ,pch=16,col='red')
z <- seq(-1,11,0.1)
z2 <- z^2
lines(z, predict(fit,newdata=data.frame(x=z,x2=z2)),
     col='blue',lwd=2)
```

If a regression line plus 95% prediction limits (would contain 95% of any future y-values) are required, add two lines to the above code fragment.

```
lines(x,predict(fit,interval='prediction')[,'lwr'],
     col='red',lwd=2,lty=2)
lines(x,predict(fit,interval='prediction')[,'upr'] ,
     col='red',lwd=2,lty=2)
```



Associated R Code

See the R file, “Using the R function lm.R” to see this function in action.