```
> resin <- read.table("http://www.stat.umn.edu/~gary/book/fcdae.data/exmpl3.2",
header=TRUE)
```

These are the log lifetime data from Table 3.1.

```
> resin
```

Again, it's a data frame. R numbers the rows and labels the columns (variables). In this file, treatments are just numbered 1 through 5.

```
   temp    y
1     1 2.04
2     1 1.91
3     1 2.00
4     1 1.92
...
```

```
> library(oehlert)
```

Could also load the R package with the book data.

```
> emp03.2
```

Note that temp in this data set has the actual temperature rather than a coding as 1, 2, , and the response has a different name.

```
   temp logTime
1   175    2.04
2   175    1.91
3   175    2.00
4   175    1.92
...
```

```
> temp <- as.factor(resin$temp); temp2 <- as.factor(emp03.2[,"temp"]);
temp3 <- with(emp03.2,as.factor(temp))
```

All of these create new variables for the temperature treatment variable that are factors, that is, a grouping variable with levels rather than quantitative. None of these changed the `temp` inside of `resin` or `emp03.2`.

```
> resin<-within(resin,{temp <- as.factor(temp)});resin$temp<-as.factor(resin[,"temp"])
```

Both of these make the `temp` inside of `resin` into a factor. Note that you need to save the output of `within`.

```
> emp03.2 <- within(emp03.2, {ftemp <- as.factor(temp)});emp03.2
```

In fact, we can even add new columns using the `within` function. Note: the curly brackets around the assignment are really only needed when we make multiple assignments in one statement.

```
   temp logTime ftemp
1   175    2.04   175
2   175    1.91   175
3   175    2.00   175
```

```
> #
```

`with(foo,x+y)` tries to do `x+y` by first looking for `x` and `y` inside of `foo`. It uses them if it finds them, otherwise it looks for `x` and `y` as generic variables. `with` returns the result of the command.

`within(foo,assignment)` is related. It tries to make the assignment within a copy of `foo`. This could either add a new variable or change a variable in `foo`. It the returns the modified data frame.

```
> #
```

For your own sanity, you need to keep track of whether you are creating/modifing variables within a data frame or outside of a data frame. You can wind up with similarly named variables inside and out, and then things get confusing, because you might think you are using one but you get the other.

```
>>>> summary(resin)
```

The material with multiple > signs at the lead is good, interesting stuff that you might, or perhaps should, do in an analysis, but it is not in the main line of exposition.
Here we get some summary information about the contents of `resin`. The information about temp is not very useful, but the information on y (the log lifetimes) is what we really want.

```
      temp             y
 Min.   :1.000   Min.   :0.830
 1st Qu.:2.000   1st Qu.:1.210
 Median :3.000   Median :1.380
 Mean   :2.865   Mean   :1.465
 3rd Qu.:4.000   3rd Qu.:1.710
 Max.   :5.000   Max.   :2.040
```

```
>>>> summary(resin[,2])
```

We can get a summary just for y by selecting the second variable in resin (treated like the second column of a matrix).

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.830   1.210   1.380   1.465   1.710   2.040
```

```
>>>> summary(resin$y);with(resin,summary(y));with(emp03.2,summary(logTime))
```

All three of these do the same thing (output only shown once).

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.830   1.210   1.380   1.465   1.710   2.040
```

```
>>>> stem(resin$y)
```

The stem and leaf plot looks roughly bimodal.

```

  The decimal point is 1 digit(s) to the left of the |

   8 | 3
  10 | 26895677
  12 | 1266781588
  14 | 2345
  16 | 166616
  18 | 580126
  20 | 04
```

```
>>>> with(resin,stem(y))
```

Gives the same thing (output not shown). If you used `within(resin,stem(y))` it would print the stem and leaf plot and then return a copy of `resin`.

```
>>>> stem(resin$y,2)
```

We can ask for about twice as many stems, and now it looks like it might be trimodal.

```
  The decimal point is 1 digit(s) to the left of the |

   8 | 3
   9 |
  10 | 2689
  11 | 5677
  12 | 126678
  13 | 1588
  14 | 2
  15 | 345
  16 | 1666
  17 | 16
  18 | 58
  19 | 0126
  20 | 04
```

```
>>>> resin$temp
```

temp does not look any different after having been made a factor.

```
 [1] 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 5 5 5 5 5 5
Levels: 1 2 3 4 5
```

```
>>>> junk <- as.factor(c(22,22,33,33,15,26));junk
```

But if you make something a little messier into a factor, R knows enough to choose the levels in ascending order.

```
[1] 22 22 33 33 15 26
Levels: 15 22 26 33
```

```
>>>> as.numeric(junk);as.numeric(junk)+10
```

If you want to do arithmetic with a factor variable, you must first convert it to a numeric variable. The numeric version is just 1, 2, ... in the order of the levels.

```
[1] 2 2 4 4 1 3
[1] 12 12 14 14 11 13
```

```
>>>> levels(junk)
```

You can retrieve the levels of a factor, but they show up as character data.

```
[1] "15" "22" "26" "33"
```

```
>>>> as.numeric(levels(junk))[as.numeric(junk)]
```

We retrieved the levels of the factor (which show up as character), and then we converted them to numeric. We then subscript with the numeric version of the factor to recover the original numeric data. So you can get the original back, but it’s often easier to just have two variables, one a factor and one not a factor.

```
[1] 22 22 33 33 15 26
```

```
>>>> junk2 <- factor(c(22,22,33,33,15,26),levels=c("22","33","26","15"))
```

Using `factor` instead of `as.factor`, we can also specify labels (in character form), and R will set the numeric levels to be in order that levels are specified. Note that 15 is now level 4.

```
>>>> junk2;as.numeric(junk2)
```

Here you can see the reordering of levels.

```
[1] 22 22 33 33 15 26
Levels: 22 33 26 15
[1] 1 1 2 2 4 3
```

```
>>>> factor(c("red","blue","yellow","Red","white","blue"))
```

You can also turn character strings into a factor, and R will sort the levels alphabetically (unless you specify some other order as we just showed). Note that levels are case sensitive. Most of our data will be set up with numbers rather than labels for levels.

```
[1] red    blue   yellow Red    white  blue
Levels: blue red Red white yellow
```

```
>>>> tb <- factor(rep(1:5,c(8,8,8,7,6)))
```

If you need to type in factor levels, you can save a lot of time by learning to use the `rep` function (repeat function). Here 1 gets repeated 8 times, 2 gets repeated 8 times, and finally 5 gets repeated six times.

```
>>>> rep(1:4,each=3,length=24)
```

For more regular patterns, you can use `rep` with the `each` and `length` parameters.

```
 [1] 1 1 1 2 2 2 3 3 3 4 4 4 1 1 1 2 2 2 3 3 3 4 4 4
```

```
>>>> with(emp03.2,split(logTime,ftemp))
```

You can split a data vector by a factor, and the result will be a list with named components corresponding to the different levels of the factor. Each element of the list will be a vector of data with the corresponding level of the factor.

```
$`175`
[1] 2.04 1.91 2.00 1.92 1.85 1.96 1.88 1.90

$`194`
[1] 1.66 1.71 1.42 1.76 1.66 1.61 1.55 1.66

$`213`
[1] 1.53 1.54 1.38 1.31 1.35 1.27 1.26 1.38

$`231`
[1] 1.15 1.22 1.17 1.16 1.21 1.28 1.17

$`250`
[1] 1.26 0.83 1.08 1.02 1.09 1.06
```

```
>>>> lapply(with(emp03.2,split(logTime,ftemp)), summary)
```

Splitting is useful, because you can use `lapply` to apply a function separately to each element of the list. In this case, we get summary statistics for the data in each treatment group.

```
$`175`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.850   1.895   1.915   1.932   1.970   2.040

$`194`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.420   1.595   1.660   1.629   1.672   1.760

$`213`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.260   1.300   1.365   1.378   1.418   1.540

$`231`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.150   1.165   1.170   1.194   1.215   1.280

$`250`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.830   1.030   1.070   1.057   1.088   1.260
```

```
>>>> sapply(with(emp03.2,split(logTime,ftemp)), summary)
```

When the output of each function application is the same shape, `sapply` will simplify the output into a compact matrix form. This would not work if we were applying the `sort()` function, because the different groups would have different lengths after sorting and would not fit into a matrix

```
          175   194   213   231   250
Min.    1.850 1.420 1.260 1.150 0.830
1st Qu. 1.895 1.595 1.300 1.165 1.030
Median  1.915 1.660 1.365 1.170 1.070
Mean    1.932 1.629 1.378 1.194 1.057
3rd Qu. 1.970 1.672 1.418 1.215 1.088
Max.    2.040 1.760 1.540 1.280 1.260
```

```
>>>> lapply(split(resin$y,resin$temp),length)
```

Another way to get the treatment group counts.

```
1 2 3 4 5
8 8 8 7 6
```

```
>>>> attach(resin)
```

I'm getting tired of typing all those dollar signs. If we `attach` a data frame, then we can refer to the elements of the data frame by name without all of the `resin$` hassle.

```
>>>> summary(y)
```

Just like magic.

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.830   1.210   1.380   1.465   1.710   2.040
```

```
>>>> y;y <- 17;y;resin$y
```

Remember my warning about creating variables similarly named to items in data frames? You have been warned again.

```
 [1] 2.04 1.91 2.00 1.92 1.85 1.96 1.88 1.90 ...
[1] 17
 [1] 2.04 1.91 2.00 1.92 1.85 1.96 1.88 1.90 ...
```

```
>>>> y[temp==1]
```

Just a reminder about subscripting.

```
[1] 2.04 1.91 2.00 1.92 1.85 1.96 1.88 1.90
```

```
>>>> replications(~temp,resin)
```

Here is another way to get counts (replications) in different treatment groups.

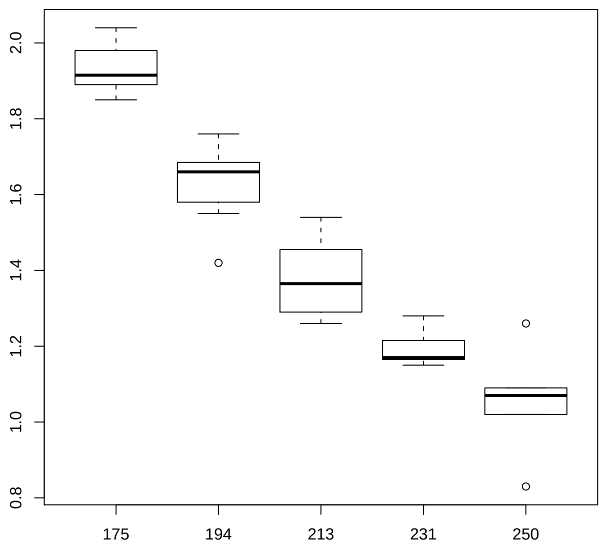
```
$temp
temp
1 2 3 4 5
8 8 8 7 6
```

```
>>>> detach(resin)
```

Let's detach the resin data for now just to keep things cleaner.

```
> boxplot(logTime~ftemp,data=emp03.2)
```

There are many ways to get boxplots of multiple groups of data. In this approach, we use a model-like approach where we model logTime by ftemp. This gives us a boxplot of logTime data separately by the levels of ftemp.



```
> library(Stat5303libs);library(cfcdae)
```

These are R libraries with some additional functions for the class. In the future, I'll just load them in by default.

```
> out <- lm(logTime ~ ftemp,data=emp03.2)
```

OK, it's time to start fitting some models to data. At this stage of the game there are two model fitting functions to use: `lm()` and/or `aov()`. I just use `lm`.

The basic arguments to `lm` are a model and an optional data frame. The form of the model is response variable, then a tilde (which we interpret as "is modeled by") and then the explanatory variable(s). In our case, we have just the single factor variable ftemp representing the different temperature treatments.

It is usually best to save the output of `lm` into a variable. We then use that variable in various ways to extract information that we need.

```
> anova(out)
```

You get analysis of variance by using the `anova()` command with the model output as the argument.

We see that temperature was highly significant, but that was obvious in the box plot.

```
Analysis of Variance Table

Response: logTime
          Df Sum Sq Mean Sq F value    Pr(>F)
ftemp      4 3.5376 0.88441  96.363 < 2.2e-16 ***
Residuals 32 0.2937 0.00918
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

```
> summary(out)
```

`summary()` gives a regression-like output that shows results for each parameter used in the model.

R's default parameterization for factors is that the first treatment effect is 0. However, when you load the cfcdae package, it changes the default parameterization for factors to the sum of the effects is zero. We can thus figure out the coefficient for treatment 5 as minus the sum of the other four coefficients.

Also note that some of the individual coefficients can be "non-significant" even within a highly significant term.

```
Call:
lm.default(formula = logTime ~ ftemp, data = emp03.2)

Residuals:
     Min       1Q   Median       3Q      Max
-0.22667 -0.03667  0.00250  0.03125  0.20333

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.43794    0.01585  90.708  < 2e-16 ***
ftemp1       0.49456    0.03065  16.134  < 2e-16 ***
ftemp2       0.19081    0.03065   6.225 5.67e-07 ***
ftemp3      -0.06044    0.03065  -1.972   0.0573 .
ftemp4      -0.24365    0.03222  -7.563 1.30e-08 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.0958 on 32 degrees of freedom
Multiple R-squared:  0.9233,Adjusted R-squared:  0.9138
F-statistic: 96.36 on 4 and 32 DF,  p-value: < 2.2e-16
```

```
> model.effects(out,"ftemp")
```

The `model.effects` function in cfcdae makes finding all the effects a little easier.

```
        175         194         213         231         250
 0.49455952  0.19080952 -0.06044048 -0.24365476 -0.38127381
```

```
> out2 <- lm(logTime~ftemp,data=emp03.2,contrasts=list(ftemp=contr.treatment))
```

R's default is to use the "first alpha is zero" parameterization, and we can force R to fit with that parameterization by using the contrasts option in lm.

```
> summary(out2)
```

Note that we now get coefficients for the last four levels of temp, and in this equivalent parameterization, the individual coefficients are all significant.

```
...
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.93250    0.03387  57.055  < 2e-16 ***
ftemp2      -0.30375    0.04790  -6.341 4.06e-07 ***
ftemp3      -0.55500    0.04790 -11.586 5.49e-13 ***
ftemp4      -0.73821    0.04958 -14.889 6.13e-16 ***
ftemp5      -0.87583    0.05174 -16.928  < 2e-16 ***
...
```

```
> predict(out)
```

If you just ask for predicted values, you get the predicted values. In this model, the predicted values are the treatment means.

```
       1        2              11        12        13        14
1.932500 1.932500 . . . 1.628750 1.628750 1.628750 1.628750 . . .
      29       30       31       32       33       34       35       36       37
1.194286 1.194286 1.194286 1.056667 1.056667 1.056667 1.056667 1.056667 1.056667
```

```
> predict(out,interval="confidence",level=.95)
```

We can ask for confidence intervals for the predicted values. The default level is 95%, so we did not really need to specify it.

```
        fit       lwr      upr
1  1.932500 1.8635073 2.001493
2  1.932500 1.8635073 2.001493
. . .
36 1.056667 0.9770008 1.136333
37 1.056667 0.9770008 1.136333
```

```
> predict(out,interval="prediction",level=.95)
```

We can also get prediction intervals for future data. Recall that prediction intervals take into account the variability of data around the mean. Thus they will be wider than confidence intervals which only try to capture the mean itself.

```
        fit       lwr      upr
1  1.932500 1.7255219 2.139478
2  1.932500 1.7255219 2.139478
. . .
36 1.056667 0.8458905 1.267443
37 1.056667 0.8458905 1.267443
```

```
>>>> predframe <-data.frame(ftemp=factor(c(175,194,213,231)));predframe
```

We can can also specify where we want to predict. We do it by having a data frame with named elements that specify the variables in the model.
Here we only need ftemp, and we'll predict at the first four levels.

```
  ftemp
1   175
2   194
3   213
4   231
```

```
>>>> predict(out,predframe)
```

Now do the prediction.

```
       1        2        3        4
1.932500 1.628750 1.377500 1.194286
```

```
> p4 <- lm(logTime~temp+I(temp^2)+I(temp^3)+I(temp^4),data=emp03.2)
```

The variable temp is a quantitative variable, and we can fit up to a quartic because there are five different levels of temp.

The form I() says to take whatever is between the parentheses and evaluate that operation. Then use it as a term in the model.

```
> p3 <- lm(logTime~temp+I(temp^2)+I(temp^3),data=emp03.2)
```

Let's fit some lower order models as well.

```
> p2 <- lm(logTime~temp+I(temp^2),data=emp03.2)

> p1 <- lm(logTime~temp,data=emp03.2)

> p0 <- lm(logTime~1,data=emp03.2)
```

This just fits an overall constant (same as single mean model).

```
> anova(p0,p4)
```

Another form of `anova` lets us directly compare two models. Even though we built p0 and p4 as functional/quantitative variable/regression type models, the full model p4 fits the same as the separate means model, and the model p0 is just the same as the single mean model. This comparison looks different, but it has the same results as the `anova(out)` we looked at above.

```
Analysis of Variance Table

Model 1: logTime ~ 1
Model 2: logTime ~ temp + I(temp^2) + I(temp^3) + I(temp^4)
  Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1     36 3.8313
2     32 0.2937  4    3.5376 96.363 < 2.2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

```
> anova(p0)
```

`anova()` applied to a single model shows you the improvement (reduction in error SS) as you add each term to the model beyond the overall constant. In a model with only the overall constant, you don't get to see much.

```
Analysis of Variance Table

Response: logTime
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals 36 3.8313 0.10643
```

```
> anova(p1)
```

The SS for temp is the reduction in residual SS going from a model of a constant mean to a model where the mean varies linearly with temp. Note that the SS for temp and Residuals in this ANOVA sum to the residual SS in the anova for p0.

```
Analysis of Variance Table

Response: logTime
          Df Sum Sq Mean Sq F value    Pr(>F)
temp       1 3.4593  3.4593  325.41 < 2.2e-16 ***
Residuals 35 0.3721  0.0106
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

```
> anova(p4)
```

With multiple terms in the model, the SS are always the reduction in residual SS when adding that term to a model that includes the preceding terms. That is, we see linear improving constant; quadratic improving linear; cubic improving quadratic; and quartic improving cubic.

Here we see that quartic does not improve over cubic, and cubic does not improve over quadratic, but quadratic does improve over linear. Thus an acceptable simple model would include linear and quadratic terms, i.e., model p2.

```
Analysis of Variance Table

Response: logTime
          Df Sum Sq Mean Sq  F value    Pr(>F)
temp       1 3.4593  3.4593 376.9128 < 2.2e-16 ***
I(temp^2)  1 0.0783  0.0783   8.5361  0.006338 **
I(temp^3)  1 0.0000  0.0000   0.0020  0.964399
I(temp^4)  1 0.0000  0.0000   0.0009  0.976258
Residuals 32 0.2937  0.0092
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

```
> summary(p4)
```

The regression-like summary shows no terms significant! These t-tests are for testing whether each term is needed when all of the other terms are already included. That should match for the quartic term (and it does), because the quartic term was the last term into the model and that is how `anova()` is looking at it.

The reason no individual term is significant in this summary is colinearity. No term is needed if you have the other three. But if we maintain hierarchy, then the only p-value we would look at in this output would be that of the quartic.

```
Call:
lm.default(formula = logTime ~ temp + I(temp^2) + I(temp^3) +
    I(temp^4), data = emp03.2)

Residuals:
     Min       1Q   Median       3Q      Max
-0.22667 -0.03667  0.00250  0.03125  0.20333

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  9.699e-01  1.957e+02   0.005    0.996
temp         7.573e-02  3.750e+00   0.020    0.984
I(temp^2)   -7.649e-04  2.679e-02  -0.029    0.977
I(temp^3)    2.600e-06  8.459e-05   0.031    0.976
I(temp^4)   -2.988e-09  9.962e-08  -0.030    0.976

Residual standard error: 0.0958 on 32 degrees of freedom
Multiple R-squared:  0.9233,Adjusted R-squared:  0.9138
F-statistic: 96.36 on 4 and 32 DF,  p-value: < 2.2e-16
```

```
> anova(p1,p2,p3,p4)
```

We can also use `anova` to directly compare a sequence of models to get improvement sums of squares. Note that these match with the anova for the terms in model p4 above.

```
Analysis of Variance Table

Model 1: logTime ~ temp
Model 2: logTime ~ temp + I(temp^2)
Model 3: logTime ~ temp + I(temp^2) + I(temp^3)
Model 4: logTime ~ temp + I(temp^2) + I(temp^3) + I(temp^4)
  Res.Df     RSS Df Sum of Sq      F   Pr(>F)
1     35 0.37206
2     34 0.29372  1  0.078343 8.5361 0.006338 **
3     33 0.29370  1  0.000019 0.0020 0.964399
4     32 0.29369  1  0.000008 0.0009 0.976258
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

```
> op4 <- lm(logTime~poly(temp,degree=4),data=emp03.2)
```

The poly() form generates orthogonal polynomials. Difficult to interpret (just use predict and be done with it), but not subject to colinearity.

```
> op3 <- lm(logTime~poly(temp,degree=3),data=emp03.2)
```

Might as well fit the lower orders, too.

```
> op2 <- lm(logTime~poly(temp,degree=2),data=emp03.2)
> op1 <- lm(logTime~poly(temp,degree=1),data=emp03.2)
```

```
> anova(op1,op2,op3,op4)
```

The model comparisons with orthogonal polynomials are exactly the same as with the non-orthogonal monomials.

```
Analysis of Variance Table

Model 1: logTime ~ poly(temp, degree = 1)
Model 2: logTime ~ poly(temp, degree = 2)
Model 3: logTime ~ poly(temp, degree = 3)
Model 4: logTime ~ poly(temp, degree = 4)
  Res.Df     RSS Df Sum of Sq      F   Pr(>F)
1     35 0.37206
2     34 0.29372  1  0.078343 8.5361 0.006338 **
3     33 0.29370  1  0.000019 0.0020 0.964399
4     32 0.29369  1  0.000008 0.0009 0.976258
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

```
> anova(op3,p4)
```

We can even mix and match and get the same results.

```
Model 1: logTime ~ poly(temp, degree = 3)
Model 2: logTime ~ temp + I(temp^2) + I(temp^3) + I(temp^4)
  Res.Df     RSS Df  Sum of Sq     F Pr(>F)
1     33 0.29370
2     32 0.29369  1 8.2568e-06 9e-04 0.9763
```

```
> anova(op4)
```

The `anova()` function considers the poly term to have all the degrees of freedom, so it does not split out the anova in 1 df pieces.

```
Analysis of Variance Table

Response: logTime
                       Df Sum Sq Mean Sq F value    Pr(>F)    
poly(temp, degree = 4)  4 3.5376 0.88441  96.363 < 2.2e-16 ***
Residuals              32 0.2937 0.00918                      
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

```
> summary(op4)
```

The p-values for individual coefficients the the orthogonal polynomial model match the p-values for the sequential anova of model p4. That is the claim to fame of orthogonal polynomials.

```
Call:
lm.default(formula = logTime ~ poly(temp, degree = 4), data = emp03.2)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.22667 -0.03667  0.00250  0.03125  0.20333 

Coefficients:
                         Estimate Std. Error t value Pr(>|t|)    
(Intercept)              1.465135   0.015750  93.027  < 2e-16 ***
poly(temp, degree = 4)1 -1.859909   0.095801 -19.414  < 2e-16 ***
poly(temp, degree = 4)2  0.279899   0.095801   2.922  0.00634 ** 
poly(temp, degree = 4)3  0.004310   0.095801   0.045  0.96440    
poly(temp, degree = 4)4 -0.002873   0.095801  -0.030  0.97626    
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.0958 on 32 degrees of freedom
Multiple R-squared:  0.9233,Adjusted R-squared:  0.9138 
F-statistic: 96.36 on 4 and 32 DF,  p-value: < 2.2e-16
```

```
> op3
```

A second claim to fame for orthogonal polynomials is that the coefficients do not change if you change the order of the polynomial. That is most definitely not true for the monomials in models p1, p2, p3, p4.

```
Call:
lm.default(formula = logTime ~ poly(temp, degree = 3), data = emp03.2)

Coefficients:
            (Intercept)  poly(temp, degree = 3)1  poly(temp, degree = 3)2  
                1.46514                 -1.85991                  0.27990  
poly(temp, degree = 3)3  
                0.00431  
```

> **predict(p4,interval="confidence")**

The full polynomial model is equivalent to the ftemp model and gives the same predictions.

```
        fit       lwr      upr
1  1.932500 1.8635073 2.001493
2  1.932500 1.8635073 2.001493
. . .
36 1.056667 0.9770008 1.136333
37 1.056667 0.9770008 1.136333
```

> **predict(op4,interval="confidence")**

Ditto for the orthogonal polynomial version.

```
        fit       lwr      upr
1  1.932500 1.8635073 2.001493
2  1.932500 1.8635073 2.001493
. . .
36 1.056667 0.9770008 1.136333
37 1.056667 0.9770008 1.136333
```

> **pframe <- data.frame(temp=c(175,194,213,231,250))**

We want to do some prediction, so let's make a data frame to predict at the observed temperatures.

> **predict(p2,pframe)**

Here are the predictions for our selected model.

```
       1        2        3        4        5
1.933083 1.627312 1.378293 1.194731 1.056229
```

> **predict(p4,pframe)**

Here are the predictions for the full model. They are basically identical, which is why the reduced model still works well.

```
       1        2        3        4        5
1.932500 1.628750 1.377500 1.194286 1.056667
```

> **predict(op4,pframe)**

Try prediction now with the full orthogonal polynomial model. The coefficients and so forth in the model are different, but the predicted values are the same.

```
       1        2        3        4        5
1.932500 1.628750 1.377500 1.194286 1.056667
```

```
> pframe2 <- data.frame(temp=175:250)
```

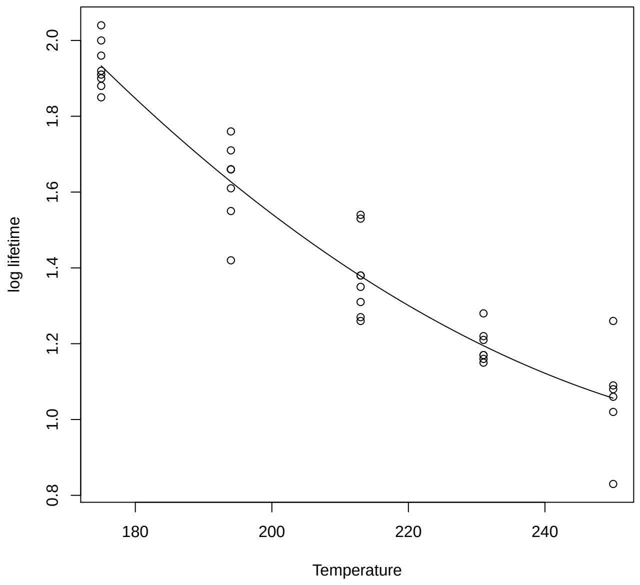
Now lets do some prediction across the entire range. We'll need a new data frame of locations to predict.

```
> plot(temp,logTime,xlab='Temperature',ylab='log lifetime')
```

Plot the observed log lifetimes against temperature.

```
> lines(pframe2$temp,predict(p2,pframe2))
```

Now add lines that link our predictions.
We see that the predicted response is not linear, but it's not very curved either. And it goes nicely through the pattern of points.



```
> AIC(p1,p2,p3,p4)
```

AIC for the four models prefers the quadratic, followed by cubic, quartic, and linear. The column for df counts parameters. Note that the quadratic model has four parameters: β_0, β_1, β_2, and σ^2.

```
   df       AIC
p1  3 -59.18418
p2  4 -65.93237
p3  5 -63.93471
p4  6 -61.93575
```

```
> BIC(p1,p2,p3,p4,k=log(37))
```

BIC also says that quadratic is best, but it prefers linear to quartic due to the harsh penalty on additional parameters.

```
   df       AIC
p1  3 -54.35142
p2  4 -59.48870
p3  5 -55.88012
p4  6 -52.27025
```

```
>>>> p5 <- lm(logTime ~ temp + I(temp^2) + I(temp^3) + I(temp^4) + ftemp,data=emp03.2)
```

OK, now we're going to do a whole bunch of peculiar, odd things that don't necessarily make much sense, but will illustrate what goes on when you specify certain kinds of non-standard models.
In this model, we've fit all four model degrees of freedom using polynomials, and then we've tried to fit the factor variable as well.

```
>>>> summary(p5)
```

I say tried to fit, because we are unable to fit any more degrees of freedom. Those first four polynomial terms (one less than the number of groups) have swallowed up all the between groups degrees of freedom and variability.

```
Call:
lm(formula = logTime ~ temp + I(temp^2) + I(temp^3) + I(temp^4) +
    ftemp)

Residuals:
     Min       1Q   Median       3Q      Max
-0.22667 -0.03667  0.00250  0.03125  0.20333

Coefficients: (4 not defined because of singularities)
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  9.699e-01  1.957e+02   0.005    0.996
temp         7.573e-02  3.750e+00   0.020    0.984
I(temp^2)   -7.649e-04  2.679e-02  -0.029    0.977
I(temp^3)    2.600e-06  8.459e-05   0.031    0.976
I(temp^4)   -2.988e-09  9.962e-08  -0.030    0.976
ftemp1              NA         NA      NA       NA
ftemp2              NA         NA      NA       NA
ftemp3              NA         NA      NA       NA
ftemp4              NA         NA      NA       NA

Residual standard error: 0.0958 on 32 degrees of freedom
Multiple R-squared: 0.9233,Adjusted R-squared: 0.9138
F-statistic: 96.36 on 4 and 32 DF,  p-value: < 2.2e-16
```

```
>>>> p6 <- lm(logTime ~ temp + I(temp^2)  + ftemp,data=emp03.2)
```

If we only do a couple of polynomial terms, then we can fit the remaining two degrees of freedom using temp. This is a bit unusual, but it has a good use shown below.

```
>>>> anova(p6)
```

What temp does is suck up all the remaining SS and DF between the treatments that had not be accounted for by linear and quadratic terms. Here it is practically nothing and not significant; this is another way to determine you do not need the higher order terms.

```
Analysis of Variance Table

Response: logTime
          Df Sum Sq Mean Sq  F value    Pr(>F)
temp       1 3.4593  3.4593 376.9128 < 2.2e-16 ***
I(temp^2)  1 0.0783  0.0783   8.5361  0.006338 **
ftemp      2 0.0000  0.0000   0.0015  0.998540
Residuals 32 0.2937  0.0092
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

```
>>>> anova(p2,p6)
```

Another way to do the same thing. If you compare this to the model with just the polynomial terms you get another way to compare the reduced polynomial model (quadratic in this case) to the full model.
We would get the same anova if we compared p2 to p4 or to out.

```
Analysis of Variance Table

Model 1: logTime ~ temp + I(temp^2)
Model 2: logTime ~ temp + I(temp^2) + ftemp
  Res.Df     RSS Df  Sum of Sq      F Pr(>F)
1     34 0.29372
2     32 0.29369  2 2.6829e-05 0.0015 0.9985
```

```
>>>> summary(p6)
```

For the love of heaven, don't try to make sense out of the coefficients when you have mixed polynomial and factor types of predictors. It can be done, but there lies madness. In particular, when you have over parameterized and used too many predictors, there are infinitely many different sets of coefficients that will lead to the same set of fitted values.

```
Call:
lm(formula = logTime ~ temp + I(temp^2) + ftemp)

Residuals:
     Min       1Q   Median       3Q      Max
-0.22667 -0.03667  0.00250  0.03125  0.20333

Coefficients: (2 not defined because of singularities)
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  7.470e+00  1.738e+00   4.298 0.000151 ***
temp        -4.559e-02  1.770e-02  -2.575 0.014843 *
I(temp^2)    7.975e-05  4.459e-05   1.789 0.083159 .
ftemp1      -1.499e-03  6.590e-02  -0.023 0.981993
ftemp2       1.791e-03  5.927e-02   0.030 0.976079
ftemp3              NA         NA      NA       NA
ftemp4              NA         NA      NA       NA
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.0958 on 32 degrees of freedom
Multiple R-squared: 0.9233,Adjusted R-squared: 0.9138
F-statistic: 96.36 on 4 and 32 DF,  p-value: < 2.2e-16
```

```
> p7 <- lm(logTime ~ ftemp + temp + I(temp^2),data=emp03.2)
```

What if we do it the other way around? That is, what if we put the factor in first and then try additional polynomial terms?

```
>>>> summary(p7)
```

The factor term has soaked up all the sums of squares and degrees of freedom between treatments, so there is nothing left for the polynomials to do; we just can't fit them now. The results for the factor coefficients look like what we saw in `summary(out)`, the model with the factor but no polynomial terms.

```
Call:
lm(formula = logTime ~ ftemp + temp + I(temp^2))

Residuals:
     Min       1Q   Median       3Q      Max
-0.22667 -0.03667  0.00250  0.03125  0.20333

Coefficients: (2 not defined because of singularities)
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.43794    0.01585  90.708  < 2e-16 ***
ftemp1       0.49456    0.03065  16.134  < 2e-16 ***
ftemp2       0.19081    0.03065   6.225 5.67e-07 ***
ftemp3      -0.06044    0.03065  -1.972   0.0573 .
ftemp4      -0.24365    0.03222  -7.563 1.30e-08 ***
temp              NA         NA      NA       NA
I(temp^2)         NA         NA      NA       NA
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.0958 on 32 degrees of freedom
Multiple R-squared: 0.9233,Adjusted R-squared: 0.9138
F-statistic: 96.36 on 4 and 32 DF,  p-value: < 2.2e-16
```

```
> anova(p2,p7)
```

All the comparisons of p2 to p4 or p7 or p6 or out give us the same results. We have four different ways of (over) parameterizing the full model, but it's still just the full model fit when all is said and done. We've just achieved that full model fit with a different set of predictors.

```
Analysis of Variance Table

Model 1: logTime ~ temp + I(temp^2)
Model 2: logTime ~ ftemp + temp + I(temp^2)
  Res.Df     RSS Df  Sum of Sq      F Pr(>F)
1     34 0.29372
2     32 0.29369  2 2.6829e-05 0.0015 0.9985
```

```
> linear.contrast(out,ftemp,c(-1,1,0,0,0))
```

The `linear.contrast()` function computes linear contrasts of treatment effects. Its first argument is the linear model object, the second is the variable for which you want to make a contrast, and the last is the contrast coefficients. There must be one coefficient for every level of the term and the coefficients must add to 0.

The value of `linear.contrast()` is an estimate of the contrast, its standard error, t-value, p-value for testing the null hypothesis that the contrast is zero with two-sided alternative, and a confidence interval.

In this example, we are comparing the first two levels of `ftemp`. The estimate 0.3038 is .4946-.1908, the difference of the first two treatment effects as found in `summary(out)` above. The se for the difference is .0479; note that this is not equal to $\sqrt{.03065^2 + .03065^2} = .0433$ (se's for the individual coefficients combined as if the estimated effects were independent); this is because coefficient estimates are (usually) dependent. Also, compare this to the output of `summary(out2)`; what do you see?

You must use contr.sum style parameterization to use this function. This is not mathematically required, but using contr.sum makes the programming easier and the results more easily understood.

```
  estimates         se   t-value      p-value   lower-ci   upper-ci
1  -0.30375 0.04790063 -6.341252 4.056402e-07 -0.4013204 -0.2061796

> summary(out2)

Call:
lm.default(formula = logTime ~ ftemp, data = emp03.2, contrasts = list(ftemp = contr.treatm

Residuals:
     Min       1Q   Median       3Q      Max 
-0.22667 -0.03667  0.00250  0.03125  0.20333 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  1.93250    0.03387  57.055  < 2e-16 ***
ftemp2      -0.30375    0.04790  -6.341 4.06e-07 ***
ftemp3      -0.55500    0.04790 -11.586 5.49e-13 ***
ftemp4      -0.73821    0.04958 -14.889 6.13e-16 ***
ftemp5      -0.87583    0.05174 -16.928  < 2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

> linear.contrast(out,ftemp,c(1/3,-1/2,1/3,-1/2,1/3))
```

Here is a contrast to compare the average of levels 2 and 4 with the average of levels 1, 3, and 5. These averages are not significantly different.

```
  estimates         se  t-value  p-value    lower-ci  upper-ci
1 0.0440377 0.03224116 1.365884 0.181499 -0.02163540 0.1097108

> linear.contrast(out,ftemp,c(-1,1.5,-1,1.5,-1))
```

This contrast is still comparing levels 2 and 4 to levels 1,3,5, but this contrast is just -3 times the previous one. Note that the p-value is the same.

```
   estimates         se   t-value  p-value   lower-ci   upper-ci
1 -0.1321131 0.09672348 -1.365884 0.181499 -0.3291324 0.06490619
```

```
> cs <- matrix(c(1,0,-2,0,1,0,1,-2,1,0),5);cs
```

We can estimate more than one contrast at a time if we use a matrix of coefficients with contrasts given in the columns of the matrix. This matrix will compare treatments 1 and 5 to treatment 3, and also compare treatments 2 and 4 to treatment 3.

```
     [,1] [,2]
[1,]    1    0
[2,]    0    1
[3,]   -2   -2
[4,]    0    1
[5,]    1    0
```

```
> linear.contrast(out,ftemp,cs)
```

OK, let's give those two contrasts a try. The first one is reasonably significant, but the second one is not. Why might we expect this given what we know about an appropriate polynomial model?

```
   estimates         se   t-value     p-value    lower-ci  upper-ci
1 0.23416667 0.08523980 2.7471516 0.009789368  0.06053887 0.4077945
2 0.06803571 0.08394822 0.8104486 0.423668953 -0.10296121 0.2390326
```

```
> linear.contrast(out,ftemp,cs,jointF=TRUE)
```

There is an additional option called `jointF`. It's false by default, but if you set it to true you will also get an F-test of the null hypothesis that all of the contrasts are zero. The numerator degrees of freedom will be at least one and no more than the number of contrasts; it could be in between these values if some of the contrasts are linearly dependent.

```
$estimates
   estimates         se   t-value     p-value    lower-ci  upper-ci
1 0.23416667 0.08523980 2.7471516 0.009789368  0.06053887 0.4077945
2 0.06803571 0.08394822 0.8104486 0.423668953 -0.10296121 0.2390326

$Ftest
        F df1 df2    p-value
 4.541971   2  32 0.01835042
```

```
> #
```

The linear.contrast function has a number of other arguments that you might find useful. For example, if you give the optional argument `allpairs=TRUE`, then you don't need to give your own coefficients, and the function will take differences of all pairs. If you want to compare all treatments to a particular treatment, say treatment k, then you can use the optional argument `controlpairs=k`, and again you don't need to specify your own contrasts. You can also change the confidence level on the intervals with the `confidence=x` optional argument.

```
> linear.contrast(out,ftemp,allpairs=TRUE,confidence=.99)
```

Here we compare all pairs of treatments using t-intervals with confidence level .99.

```
      estimates         se   t-value      p-value     lower-ci  upper-ci
1 - 2 0.3037500 0.04790063  6.341252 4.056402e-07  0.172574999 0.4349250
1 - 3 0.5550000 0.04790063 11.586485 5.492304e-13  0.423824999 0.6861750
1 - 4 0.7382143 0.04958187 14.888796 6.128444e-16  0.602435260 0.8739933
1 - 5 0.8758333 0.05173860 16.928045 1.580931e-17  0.734148139 1.0175185
2 - 3 0.2512500 0.04790063  5.245233 9.735655e-06  0.120074999 0.3824250
2 - 4 0.4344643 0.04958187  8.762564 5.174824e-10  0.298685260 0.5702433
2 - 5 0.5720833 0.05173860 11.057186 1.834240e-12  0.430398139 0.7137685
3 - 4 0.1832143 0.04958187  3.695187 8.176935e-04  0.047435260 0.3189933
3 - 5 0.3208333 0.05173860  6.201044 6.072828e-07  0.179148139 0.4625185
4 - 5 0.1376190 0.05329891  2.582024 1.460493e-02 -0.008339019 0.2835771
```

```
> linear.contrast(out,ftemp,controlpairs=5,confidence=.999)
```

Here we compare all treatments to treatment 5 at the .999 confidence level. At that level, treatment 4 does not differ from treatment 5 (just barely), and the other treatments all differ from treatment 5.

```
      estimates         se   t-value      p-value    lower-ci  upper-ci
1 - 5 0.8758333 0.05173860 16.928045 1.580931e-17  0.68844636 1.0632203
2 - 5 0.5720833 0.05173860 11.057186 1.834240e-12  0.38469636 0.7594703
3 - 5 0.3208333 0.05173860  6.201044 6.072828e-07  0.13344636 0.5082203
4 - 5 0.1376190 0.05329891  2.582024 1.460493e-02 -0.05541905 0.3306571
```

```
> pr03.5
```

This is from the oehlert library and gives strength of concrete with different percentages of polypropylene fibers embedded.

```
   pctFiber strength
1      0.00      7.8
2      0.25      7.9
3      0.50      7.4
4      0.75      7.0
5      1.00      5.9
. . .
```

```
> newpr03.5 <- within(pr03.5,fFiber <- as.factor(pctFiber))
```

Make a treatment factor.

```
> out <- lm(strength~fFiber,data=newpr03.5)
```

Do the basic model. The anova shows that we can reject the null of all means equal.

```
> anova(out)
Analysis of Variance Table

Response: strength
          Df Sum Sq Mean Sq F value    Pr(>F)
fFiber     4 6.2627 1.56567  12.975 0.0005713 ***
Residuals 10 1.2067 0.12067
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

```
> model.effects(out,"fFiber")
```

Here are the model effects.

```
          0         0.25          0.5         0.75            1 
 0.593333333  0.693333333 -0.006666667 -0.173333333 -1.106666667 
```

```
> linear.contrast(out,fFiber,c(.25,.25,.25,.25,-1))
```

Having snooped in the data, I think that I just want to compare the last treatment to the average of the first four.

```
  estimates        se  t-value      p-value  lower-ci upper-ci
1  1.383333 0.2242271 6.169341 0.0001056076 0.8837243 1.882942
```

```
> 6.169^2*.12067
```

The SS for the contrast is the square of the t times the MSE. This contrast is about 75% of the total treatment SS.

```
[1] 4.592285
```

```
> 6.169^2/4
```

The Scheffé F is the square of the t divided by g-1.

```
[1] 9.51414
```

```
> pf(9.514,4,10,lower=FALSE)
```

Get the p-value. We see that it is significant even after adjusting for data snooping.

```
[1] 0.001935102
```

```
> matcfs <- matrix(c(1,-1,0,0,0,  0,1,-1,0,0,  0,0,1,-1,0,  0,0,0,1,-1),nr=5)
```

Suppose instead that I just wanted to do four contrasts: trt 1 vs 2, trt 2 vs 3, trt 3 vs 4, and trt 4 vs 5. Here is a matrix of coefficients for the four contrasts.

```
> matcfs
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]   -1    1    0    0
[3,]    0   -1    1    0
[4,]    0    0   -1    1
[5,]    0    0    0   -1
```

```
> linear.contrast(out,fFiber,matcfs)
```

Do the contrasts and get the p-values.

```
   estimates        se    t-value     p-value   lower-ci  upper-ci
1 -0.1000000 0.2836273 -0.3525754 0.731724790 -0.7319610 0.5319610
2  0.7000000 0.2836273  2.4680276 0.033216842  0.0680390 1.3319610
3  0.1666667 0.2836273  0.5876256 0.569813979 -0.4652943 0.7986277
4  0.9333333 0.2836273  3.2907035 0.008140343  0.3013723 1.5652943
```

```
> .05/4
```

To use Bonferroni at the 5% level, each test should be run at the 1.25% level. We see that the fourth comparison is significant, so we can reject the null that all the nulls are true.

```
[1] 0.0125
```

```
> cheese <- read.table("http://www.stat.umn.edu/~gary/book/fcdae.data/exmpl5.5",
header=TRUE)
```

Let's use some different data for a while. This is the free amino acids in cheese data from example 5.5 of the text.

```
> names(cheese)
```

Two variables, y and trt.

```
[1] "trt" "y"
```

```
> cheese$trt <- factor(cheese$trt)
```

Make trt a factor.

```
> cout <- lm(y~trt,data=cheese)
```

Fit the model.

```
> anova(cout)
```

There is a reasonably significant difference between the treatments.

```
Analysis of Variance Table

Response: y
          Df Sum Sq Mean Sq F value  Pr(>F)
trt        3 5.6279  1.8760  11.932 0.01830 *
Residuals  4 0.6289  0.1572
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

```
> summary(cout)
```

Basic summary information.

```
Call:
lm(formula = y ~ trt, data = cheese)

Residuals:
      1       2       3       4       5       6       7       8
 0.0100 -0.3050 -0.4400 -0.1665 -0.0100  0.3050  0.4400  0.1665

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   5.0604     0.1402  36.097 3.52e-06 ***
trt1         -0.8754     0.2428  -3.605   0.0227 *
trt2         -0.6304     0.2428  -2.596   0.0603 .
trt3          0.2446     0.2428   1.007   0.3707
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.3965 on 4 degrees of freedom
Multiple R-squared: 0.8995,Adjusted R-squared: 0.8241
F-statistic: 11.93 on 3 and 4 DF,  p-value: 0.01830
```

```
> pairwise(cout,trt)
```

The pairwise function does all pairwise comparisons, and then prints confidence intervals based on various techniques. By default, it uses HSD and .95 coverage. Significant differences are marked with a star.

```
Pairwise comparisons ( hsd ) of trt
        estimate signif diff     lower      upper
  1 - 2  -0.2450    1.614152 -1.859152  1.3691524
  1 - 3  -1.1200    1.614152 -2.734152  0.4941524
* 1 - 4  -2.1365    1.614152 -3.750652 -0.5223476
  2 - 3  -0.8750    1.614152 -2.489152  0.7391524
* 2 - 4  -1.8915    1.614152 -3.505652 -0.2773476
  3 - 4  -1.0165    1.614152 -2.630652  0.5976524
```

```
> pairwise(cout,trt,confidence=.9)
```

Here we use confidence .9 to match the example in the text.

```
Pairwise comparisons ( hsd ) of trt
        estimate signif diff     lower      upper
  1 - 2  -0.2450    1.285889 -1.530889  1.0408889
  1 - 3  -1.1200    1.285889 -2.405889  0.1658889
* 1 - 4  -2.1365    1.285889 -3.422389 -0.8506111
  2 - 3  -0.8750    1.285889 -2.160889  0.4108889
* 2 - 4  -1.8915    1.285889 -3.177389 -0.6056111
  3 - 4  -1.0165    1.285889 -2.302389  0.2693889
```

```
> pairwise(cout,trt,confidence=.9,type="regwr")
```

Now with regwr.

```
Pairwise comparisons ( regwr ) of trt
        estimate signif diff     lower        upper
  1 - 2  -0.2450    1.100905 -1.345905  0.855905084
* 1 - 3  -1.1200    1.114670 -2.234670 -0.005330492
* 1 - 4  -2.1365    1.285889 -3.422389 -0.850611111
  2 - 3  -0.8750    1.100905 -1.975905  0.225905084
* 2 - 4  -1.8915    1.114670 -3.006170 -0.776830492
  3 - 4  -1.0165    1.100905 -2.117405  0.084405084
```

```
> pairwise(cout,trt,confidence=.9,type="snk")
```

Now with snk.

```
Pairwise comparisons ( snk ) of trt
        estimate signif diff     lower        upper
  1 - 2  -0.2450   0.8453095 -1.090309  0.600309491
* 1 - 3  -1.1200   1.1146695 -2.234670 -0.005330492
* 1 - 4  -2.1365   1.2858889 -3.422389 -0.850611111
* 2 - 3  -0.8750   0.8453095 -1.720309 -0.029690509
* 2 - 4  -1.8915   1.1146695 -3.006170 -0.776830492
* 3 - 4  -1.0165   0.8453095 -1.861809 -0.171190509
```

```
> pairwise(cout,trt,confidence=.9,type="lsd")
```

Now with lsd, which is basically the same output as linear.contrast with the allpairs=TRUE option. In this case, it's the same pattern as snk.

```
Pairwise comparisons ( lsd ) of trt
        estimate signif diff     lower       upper
  1 - 2  -0.2450   0.8453078 -1.090308  0.60030783
* 1 - 3  -1.1200   0.8453078 -1.965308 -0.27469217
* 1 - 4  -2.1365   0.8453078 -2.981808 -1.29119217
* 2 - 3  -0.8750   0.8453078 -1.720308 -0.02969217
* 2 - 4  -1.8915   0.8453078 -2.736808 -1.04619217
* 3 - 4  -1.0165   0.8453078 -1.861808 -0.17119217
```

```
> lines(pairwise(cout,trt,confidence=.9,type="snk"))
```

An alternative presentation is to use "underline" diagrams, although as done here the diagram is turned on its side so that the underlines are actually sidelines. There will be a line covering a pair of treatments if they are not significantly different.

```
1 -0.875 |
2 -0.630 |
3  0.245
4  1.261
```

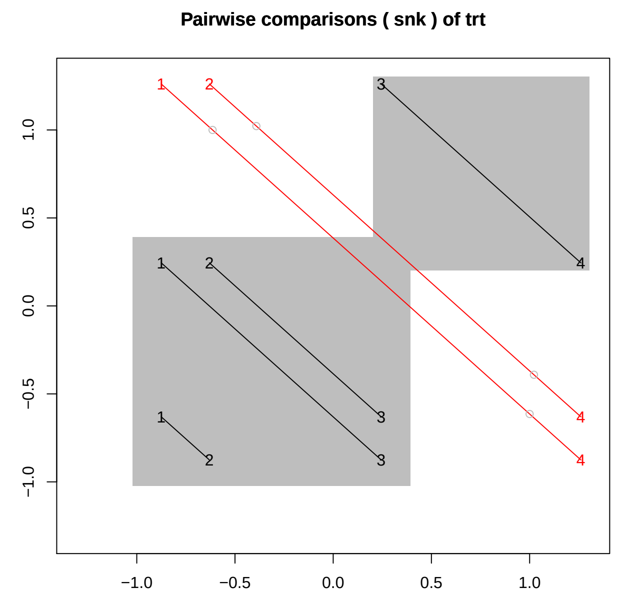
```
> lines(pairwise(cout,trt,confidence=.9,type="regwr"))
```

regwr has a more complicated set of lines.

```
1 -0.875 |
2 -0.630 | |
3  0.245   | |
4  1.261     |
```

```
> plot(pairwise(cout,trt,type="snk"))
```

We can also plot pairwise differences. Significant differences are shown in red; the limits of the cutoffs are shown as light gray circles. Nonsignificant differences are shown in black, and there is a gray zone of insignificance shown. In this case, treatments 1 and 3 not being significantly different implies that we will call the 1&2 and 2&3 pairs also nonsignificant. Those other two pairs are shown in the nonsignificance box generated by the 1&3 comparison.



```
> compare.to.control(cout,trt,control=1)
```

Treatment 1 is actually a control treatment, so we might like to compare the other treatments to it. The compare.to.control function does pairwise differences with the control treatment (which is indicated by the control=k argument), and uses a Dunnett correction on the comparisons. By default, it uses confidence .95, but you can change that as usual. (This is like the controlpairs option in linear.contrast, but it uses the Dunnett adjustment.)

```
        difference      lower    upper
  2 - 1     0.2450 -1.1883313 1.678331
  3 - 1     1.1200 -0.3133313 2.553331
* 4 - 1     2.1365  0.7031687 3.569831
```

```
> soybeans <- read.table("http://www.stat.umn.edu/~gary/book/fcdae.data/exmpl5.10",
header=TRUE)
```

Let's look at the soybean data from Example 5.10.

```
> names(soybeans)


[1] "trt" "y"
> sout <- lm(sqrt(100-y)~as.factor(trt),data=soybeans)
```

Here is something new. We can transform the response and make treatments into factors from within the model statement.

```
> anova(sout)
```

Basic anova is highly significant.

```
Analysis of Variance Table

Response: y
          Df  Sum Sq Mean Sq F value    Pr(>F)
trt       13 105.960   8.151  14.907 9.383e-12 ***
Residuals 42  22.965   0.547
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

> pairwise(sout,as.factor(trt))
```

Holy cow! Fourteen different treatments! There are 91 different pairwise comparisons. This goes on forever. Note how we had to "name" the factor.

```
  1 - 2   -1.615710e+00    1.859434 -3.47514427  0.24372390
  1 - 3   -1.680337e+00    1.859434 -3.53977142  0.17909674
  1 - 4   -1.543076e+00    1.859434 -3.40251047  0.31635769
* 1 - 5   -1.941476e+00    1.859434 -3.80091047 -0.08204231
...
  12 - 13 -8.735121e-01    1.859434 -2.73294620  0.98592196
  12 - 14 -1.640346e+00    1.859434 -3.49978004  0.21908812
  13 - 14 -7.668338e-01    1.859434 -2.62626792  1.09260024

> lines(pairwise(sout,as.factor(trt)))
```

Instead of looking at all those confidence intervals, let's go straight to the line diagram.

```

1  -1.68951 |
11 -1.68951 |
6  -1.27694 | |
10 -1.07148 | |
7  -1.07148 | |
8  -0.17022 | | |
4  -0.14644 | | |
2  -0.07380 | | |
3  -0.00918 | | |
9   0.15775 | | |
5   0.25196   | |
12  1.42500     | |
13  2.29851       |
14  3.06534       |
> compare.to.best(sout,as.factor(trt),conf=.99)
```

You know, with 14 different treatments it would be useful to figure out which ones are equivalent to the best. This uses the one-sided Dunnett allowance and marks those treatments that are significantly different with a star.

```
           difference allowance
best is 14  0.0000000        NA
  13 - 14  -0.7668338 -1.715237
  12 - 14  -1.6403460 -1.715237
* 5 - 14   -2.8133798 -1.715237
* 9 - 14   -2.9075936 -1.715237
* 3 - 14   -3.0745188 -1.715237
* 2 - 14   -3.1391460 -1.715237
* 4 - 14   -3.2117798 -1.715237
* 8 - 14   -3.2355636 -1.715237
* 7 - 14   -4.1368222 -1.715237
* 10 - 14  -4.1368222 -1.715237
* 6 - 14   -4.3422858 -1.715237
* 11 - 14  -4.7548562 -1.715237
* 1 - 14   -4.7548562 -1.715237
```

```
> compare.to.best(sout,as.factor(trt),conf=.99,lowisbest=TRUE)
```

Hmm, what do we do if the smallest response is best? Simply use the lowisbest=TRUE optional argument.

```
            difference allowance
* 14 - 1  4.754856e+00   1.71744
* 13 - 1  3.988022e+00   1.71744
* 12 - 1  3.114510e+00   1.71744
* 5 - 1   1.941476e+00   1.71744
* 9 - 1   1.847263e+00   1.71744
  3 - 1   1.680337e+00   1.71744
  2 - 1   1.615710e+00   1.71744
  4 - 1   1.543076e+00   1.71744
  8 - 1   1.519293e+00   1.71744
  7 - 1   6.180340e-01   1.71744
  10 - 1  6.180340e-01   1.71744
  6 - 1   4.125704e-01   1.71744
  11 - 1  1.554312e-15   1.71744
best is 1 0.000000e+00        NA
```