Contrasts in R

Rose Maier

March 13, 2015

A (sort of) Complete Guide to Contrasts in R

Sherlock Holmes and the Case of the Inconsistent Contrasts

So far, we've been using the contrasts() command to set contrasts in R, but you may have noticed that sometimes the results don't look quite right. If you set your contrast weights to sum to 1 and -1 on each side of the contrast, the resulting contrast estimate should equal the difference bewteen the means (if your contrast weights sum to 2 and -2, then it should equal twice the mean difference, etc.). Sometimes it appears to work just fine, but sometimes it doesn't! And if you throw in a nonorthogonal comparison, things get even more mysterious! What to make of this, Watson?

It turns out the problem is that the contrasts() command isn't actually expecting contrast weights, like you might think. It's built for coding schemes (like traditional dummy coding), so it's actually expecting *the inverse* of the matrix of desired contrast weights, which can be thought of as the contrast coding scheme rather than the contrast weights themselves (eye roll). This is very frustratingly not at all clear in the help documentation for the function, or in the many webpages providing examples and tutorials on how to use it. We'll walk through examples of how to do this below (see the section on DIY contrasts).

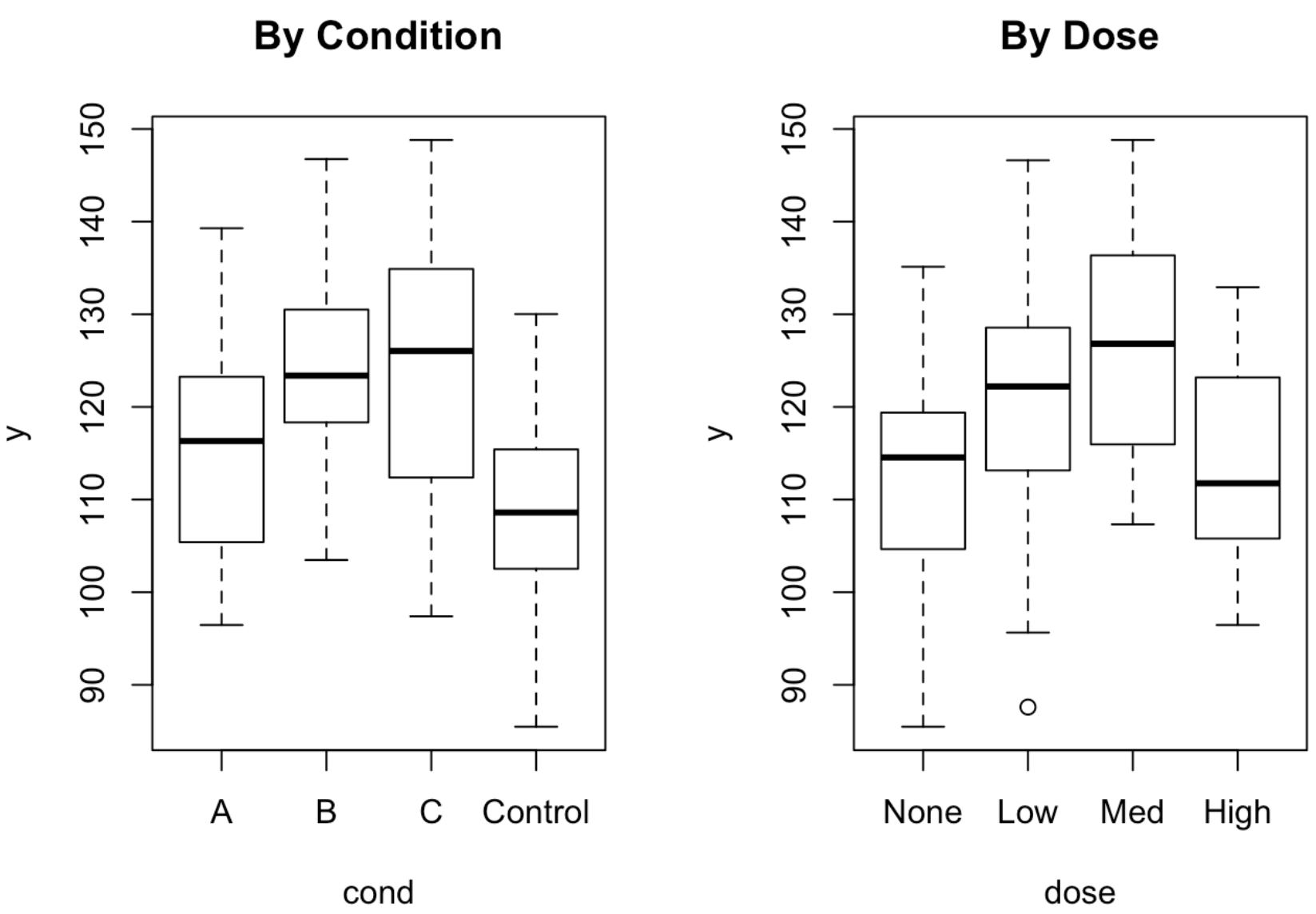
The goal of this tutorial is to give you a better idea of how contrasts work in R, and what you need to do to get R to run the comparisons you want.

```
# first, generate some random data
n <- 128
cond <- gl(4, n/4, labels=c("A", "B", "C","Control"))
dose <- gl(4, 4, length=n, labels=c("None", "Low", "Med", "High"), ordered=TRUE)
data <- data.frame(cond=cond, dose=dose)
data$y <- ifelse(data$cond=="Control", rnorm(n, mean=100, sd=10),
                 ifelse(data$cond=="A", rnorm(n, mean=110, sd=10),
                        rnorm(n, mean=120, sd=10)))
data$y <- ifelse(data$dose=="Low", data$y + 10,
                 ifelse(data$dose=="Med", data$y + 15,
                        data$y))

str(data)
```

```
## 'data.frame':    128 obs. of  3 variables:
##  $ cond: Factor w/ 4 levels "A","B","C","Control": 1 1 1 1 1 1 1 1 1 1 ...
##  $ dose: Ord.factor w/ 4 levels "None"<"Low"<"Med"<..: 1 1 1 1 2 2 2 2 3 3 ...
##  $ y   : num  105 116 105 104 124 ...
```

```
par(mfrow=c(1,2)) # put the next two plots side by side
plot(y ~ cond, data=data, main="By Condition")
plot(y ~ dose, data=data, main="By Dose")
```




```
par(mfrow=c(1,1)) # return to the default of one plot at a time
```

Two Ways to Apply Contrasts

You can apply contrasts either as a feature of the variable itself, as part of the dataframe, or specify it in the model. If you add your contrasts to the dataframe, then those contrasts will get used every time that variable gets called in any model (unless you overwrite them with new contrasts). If you specify contrasts in the model, then they'll only get used in that model. So choose whichever strategy is best for your situation (there will be more examples of this below).

Default Contrasts: Treatment Contrasts (Traditional Dummy Coding)

By default, R uses traditional dummy coding (called “treatment contrasts” in R) for any non-ordered factors, and polynomial trend contrasts for any ordered factors. That works out well if you intend to look at regression coefficients, with lm() for example:

```
model1 <- lm(y ~ cond, data=data)
summary(model1)
```

```
##
## Call:
## lm(formula = y ~ cond, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -27.3811  -8.6832  -0.2297   7.5401  24.0410
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  116.017      2.077  55.864  < 2e-16 ***
## condB          8.609      2.937   2.931  0.00402 **
## condC          8.758      2.937   2.982  0.00345 **
## condControl   -7.123      2.937  -2.425  0.01674 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.75 on 124 degrees of freedom
## Multiple R-squared:  0.2469, Adjusted R-squared:  0.2287
## F-statistic: 13.55 on 3 and 124 DF,  p-value: 1.056e-07
```

It automatically uses the first level as the reference group. In some cases, there’s another level you would prefer to use as the reference group (for example, in these data, the control condition would probably be a better choice than condition A). You can change the reference group with the relevel() command:

```
levels(data$cond)
```

```
## [1] "A"       "B"       "C"       "Control"
```

```
data$cond <- relevel(data$cond, ref="Control")
levels(data$cond) # Note that the order of the levels has changed
```

```
## [1] "Control" "A"       "B"       "C"
```

```
model1.contrref <- lm(y ~ cond, data=data)
summary(model1.contrref) # Now control is being used as the refernce group, since it is
now the first level
```

```
##
## Call:
## lm(formula = y ~ cond, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -27.3811  -8.6832  -0.2297   7.5401  24.0410
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  108.894      2.077  52.434  < 2e-16 ***
## condA          7.123      2.937   2.425   0.0167 *
## condB         15.731      2.937   5.356 3.98e-07 ***
## condC         15.881      2.937   5.407 3.16e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.75 on 124 degrees of freedom
## Multiple R-squared:  0.2469, Adjusted R-squared:  0.2287
## F-statistic: 13.55 on 3 and 124 DF,  p-value: 1.056e-07
```

I chose to relevel condition and save the new levels to the dataframe. That means every time I use that variable from now on (models, plotting, etc.), the order of the levels will be with “control” first. It’s also possible to relevel just inside the model, without saving it to the dataframe. For example, if I want to use condition C as the reference group for this model, but I don’t want to save that new ordering to the datframe:

```
model1.condCref <- lm(y ~ relevel(cond, ref="C"), data=data)
levels(data$cond) # note that it is not changed
```

```
## [1] "Control" "A"       "B"       "C"
```

```
summary(model1.condCref)
```

```
##
## Call:
## lm(formula = y ~ relevel(cond, ref = "C"), data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -27.3811  -8.6832  -0.2297   7.5401  24.0410
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    124.7750     2.0768  60.081  < 2e-16 ***
## relevel(cond, ref = "C")Control -15.8805     2.9370  -5.407 3.16e-07 ***
## relevel(cond, ref = "C")A        -8.7579     2.9370  -2.982  0.00345 **
## relevel(cond, ref = "C")B        -0.1491     2.9370  -0.051  0.95960
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.75 on 124 degrees of freedom
## Multiple R-squared:  0.2469, Adjusted R-squared:  0.2287
## F-statistic: 13.55 on 3 and 124 DF,  p-value: 1.056e-07
```

Note that traditional dummy coding is fine for regression coefficients, but since traditional dummy codes aren't orthogonal, it messes things up when you're just trying to partition variance (i.e. an ANOVA). (Also remember that the default R anova functions use type 1 sums of squares, which is generally not what you want. To get type 3 sums of squares, use the Anova() function from the car package.)

```
model2 <- lm(y ~ cond * dose, data=data) # factorial ANOVA

library(car)
Anova(model2, type="III")
```

```
## Anova Table (Type III tests)
##
## Response: y
##             Sum Sq  Df   F value    Pr(>F)
## (Intercept) 379456   1 3633.5971 < 2.2e-16 ***
## cond          5610   3   17.9064 1.458e-09 ***
## dose           817   3    2.6073   0.05518 .
## cond:dose     1075   9    1.1441   0.33818
## Residuals    11696 112
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

R is still using traditional dummy coding (treatment contrasts) behind the scenes here, unlike other stats software (like SPSS) that would switch to effects coding for an ANOVA. You can see this because if we relevel condition, the sums of squares will change!

```
model2.relevel <- lm(y ~ relevel(cond, ref="C") * dose, data=data)

Anova(model2.relevel, type="III")
```

```
## Anova Table (Type III tests)
##
## Response: y
##                              Sum Sq  Df   F value    Pr(>F)
## (Intercept)                  498202   1 4770.6789 < 2.2e-16 ***
## relevel(cond, ref = "C")       5610   3   17.9064 1.458e-09 ***
## dose                           2858   3    9.1225 1.879e-05 ***
## relevel(cond, ref = "C"):dose  1075   9    1.1441    0.3382
## Residuals                     11696 112
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note the mysterious changes in the intercept and effect of dose, Watson. Most irksome.

Thankfully, you can override the default contrasts (treatment contrasts for non-ordered factors and polynomial trends for ordered factors) with whatever you like.

Setting up for an ANOVA: Sum-to-zero Contrasts (Effects Coding)

For an ANOVA, you should set your factors to use effects coding, rather than relying on the default treatment codes. You can do that with the contr.sum() function:

```
# this overrides the default contrasts and tells it to use the contr.sum() function to m
ake contrasts instead
contrasts(data$cond) <- contr.sum
contrasts(data$dose) <- contr.sum

model3 <- lm(y ~ cond * dose, data=data)

Anova(model3, type="III")
```

```
## Anova Table (Type III tests)
##
## Response: y
##               Sum Sq  Df    F value    Pr(>F)
## (Intercept) 1799778   1 17234.3192 < 2.2e-16 ***
## cond           5610   3    17.9064 1.458e-09 ***
## dose           4342   3    13.8609 9.477e-08 ***
## cond:dose      1075   9     1.1441    0.3382
## Residuals     11696 112
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

What if you have some actual comparisons you'd like to run? For example, maybe you want to run Helmert contrasts on condition, and polynomial trend contrasts on dose? As long as they're orthogonal, they'll work fine in an ANOVA:

```
contrasts(data$cond) <- contr.helmert
contrasts(data$dose) <- contr.poly

model4 <- lm(y ~ cond * dose, data=data)

Anova(model4, type="III")
```

```
## Anova Table (Type III tests)
##
## Response: y
##               Sum Sq  Df    F value    Pr(>F)
## (Intercept) 1799778   1 17234.3192 < 2.2e-16 ***
## cond           5610   3    17.9064 1.458e-09 ***
## dose           4342   3    13.8609 9.477e-08 ***
## cond:dose      1075   9     1.1441    0.3382
## Residuals     11696 112
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Again, you could choose to alter the contrasts in the dataframe itself (as I did above), or you could specify it just within the model call. You can see that the above contrasts have been saved in the dataframe:

```
str(data)
```

```
## 'data.frame':    128 obs. of  3 variables:
##  $ cond: Factor w/ 4 levels "Control","A",..: 2 2 2 2 2 2 2 2 2 2 ...
##   ..- attr(*, "contrasts")= num [1:4, 1:3] -1 1 0 0 -1 -1 2 0 -1 -1 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr  "Control" "A" "B" "C"
##   .. .. ..$ : NULL
##  $ dose: Ord.factor w/ 4 levels "None"<"Low"<"Med"<..: 1 1 1 1 2 2 2 2 3 3 ...
##   ..- attr(*, "contrasts")= num [1:4, 1:3] -0.671 -0.224 0.224 0.671 0.5 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr  "None" "Low" "Med" "High"
##   .. .. ..$ : chr  ".L" ".Q" ".C"
##  $ y   : num  105 116 105 104 124 ...
```

The moral of the story: For ANOVAs, effects coding works great, orthogonal contrast coding works great, and traditional dummy coding not so much.

Getting to Actually See the Results of Your Contrasts

Anova() won't show you the individual contrast results, just the overall effect of each factor. You can see the results of each contrast by using the summary() function on the model object:

```
summary(model4)
```

```
##
## Call:
## lm(formula = y ~ cond * dose, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -23.805  -5.759  -1.246   6.148  19.469
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  118.5781     0.9032 131.280  < 2e-16 ***
## cond1          3.5613     1.2774   2.788 0.006233 **
## cond2          4.0567     0.7375   5.501 2.42e-07 ***
## cond3          2.0656     0.5215   3.961 0.000132 ***
## dose.L         1.8356     1.8065   1.016 0.311776
## dose.Q       -10.9778     1.8065  -6.077 1.73e-08 ***
## dose.C        -3.4383     1.8065  -1.903 0.059566 .
## cond1:dose.L  -1.0675     2.5548  -0.418 0.676848
## cond2:dose.L  -1.0571     1.4750  -0.717 0.475077
## cond3:dose.L  -1.1785     1.0430  -1.130 0.260910
## cond1:dose.Q  -2.9255     2.5548  -1.145 0.254611
## cond2:dose.Q   1.0866     1.4750   0.737 0.462843
## cond3:dose.Q  -2.3773     1.0430  -2.279 0.024544 *
## cond1:dose.C   1.3695     2.5548   0.536 0.592993
## cond2:dose.C   1.2342     1.4750   0.837 0.404516
## cond3:dose.C  -0.5667     1.0430  -0.543 0.587978
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.22 on 112 degrees of freedom
## Multiple R-squared:  0.4853, Adjusted R-squared:  0.4164
## F-statistic:  7.04 on 15 and 112 DF,  p-value: 1.361e-10
```

If you use aov() instead of lm() to specify the original model, then you'll need to add a split argument to the summary() call to see the contrast results:

```
model5 <- aov(y ~ cond * dose, data=data)
summary(model5) # no contrast results displayed
```

```
##              Df Sum Sq Mean Sq F value   Pr(>F)
## cond          3   5610  1870.0  17.906 1.46e-09 ***
## dose          3   4342  1447.5  13.861 9.48e-08 ***
## cond:dose     9   1075   119.5   1.144    0.338
## Residuals   112  11696   104.4
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(model5, split=list( cond=list(AvB=1, CvAB=2, CtrlvElse=3),
                            dose=list(Linear=1, Quad=2, Cubic=3))) # there are my contras
ts!
```

```
##                                Df Sum Sq Mean Sq F value   Pr(>F)
## cond                            3   5610    1870  17.906 1.46e-09 ***
##   cond: AvB                     1    812     812   7.773 0.006233 **
##   cond: CvAB                    1   3160    3160  30.257 2.42e-07 ***
##   cond: CtrlvElse               1   1638    1638  15.690 0.000132 ***
## dose                            3   4342    1447  13.861 9.48e-08 ***
##   dose: Linear                  1    108     108   1.032 0.311776
##   dose: Quad                    1   3856    3856  36.928 1.73e-08 ***
##   dose: Cubic                   1    378     378   3.623 0.059566 .
## cond:dose                       9   1075     119   1.144 0.338175
##   cond:dose: AvB.Linear         1     18      18   0.175 0.676848
##   cond:dose: CvAB.Linear        1     54      54   0.514 0.475077
##   cond:dose: CtrlvElse.Linear   1    133     133   1.277 0.260910
##   cond:dose: AvB.Quad           1    137     137   1.311 0.254611
##   cond:dose: CvAB.Quad          1     57      57   0.543 0.462843
##   cond:dose: CtrlvElse.Quad     1    543     543   5.195 0.024544 *
##   cond:dose: AvB.Cubic          1     30      30   0.287 0.592993
##   cond:dose: CvAB.Cubic         1     73      73   0.700 0.404516
##   cond:dose: CtrlvElse.Cubic    1     31      31   0.295 0.587978
## Residuals                     112  11696     104
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Note that this is just controlling what output gets displayed. If you want to show few
er contrasts, you can just not include them in the split list. For example, I can leave
out all the stuff about cubic trends in dose:
summary(model5, split=list( cond=list(AvB=1, CvAB=2, CtrlvElse=3),
                            dose=list(Linear=1, Quad=2)))
```

```
##                                Df Sum Sq Mean Sq F value   Pr(>F)
## cond                            3   5610    1870  17.906 1.46e-09 ***
##   cond: AvB                     1    812     812   7.773 0.006233 **
##   cond: CvAB                    1   3160    3160  30.257 2.42e-07 ***
##   cond: CtrlvElse               1   1638    1638  15.690 0.000132 ***
## dose                            3   4342    1447  13.861 9.48e-08 ***
##   dose: Linear                  1    108     108   1.032 0.311776
##   dose: Quad                    1   3856    3856  36.928 1.73e-08 ***
## cond:dose                       9   1075     119   1.144 0.338175
##   cond:dose: AvB.Linear         1     18      18   0.175 0.676848
##   cond:dose: CvAB.Linear        1     54      54   0.514 0.475077
##   cond:dose: CtrlvElse.Linear   1    133     133   1.277 0.260910
##   cond:dose: AvB.Quad           1    137     137   1.311 0.254611
##   cond:dose: CvAB.Quad          1     57      57   0.543 0.462843
##   cond:dose: CtrlvElse.Quad     1    543     543   5.195 0.024544 *
## Residuals                     112  11696     104
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

You can see the contrasts you ran in the model object saved from lm(). This is especially helpful if you're using automatic contrast functions, like contr.helmert(), and you want to check that the function is really running the comparisons you think it is. For example, you can see here that what R uses when you call contr.helmert() is actually what would be called "reverse" Helmert contrasts in other software.

```
attributes(model4$qr$qr)$contrasts
```

```
## $cond
##         [,1] [,2] [,3]
## Control   -1   -1   -1
## A          1   -1   -1
## B          0    2   -1
## C          0    0    3
##
## $dose
##               .L   .Q         .C
## None -0.6708204  0.5 -0.2236068
## Low  -0.2236068 -0.5  0.6708204
## Med   0.2236068 -0.5 -0.6708204
## High  0.6708204  0.5  0.2236068
```

How did I figure out how to pull this information out of the model object? Lots of time playing around with str(model4).

DIY Contrasts

While there are a handful of automatic contrast functions in R (what I've been using so far), you will sometimes find yourself wanting to run comparisons that are not included there. When that happens, you can specify them yourself. You need to be careful, though, because the contrasts() function is a sneaky little bastard, as noted above. To apply contrast weights, you'll need to give it the inverse of your matrix of weights.

For example, let's say we wanted to compare Control to A, B, and C (contrast 1), and then compare A to B (contrast 2), and then A to C (contrast 3). Note that these are not orthogonal.
1. Specfiy the weights for your contrasts (and be sure to check the order of the levels of the factor, so your weights will line up properly)
2. Create a temporary matrix with each contrast as one row. The top row (for the constant) should be 1/j for j groups.
3. Get the inverse of that temporary matrix.
4. The first column of the inverse will be all 1's. Drop that first column. The remaining columns are your contrast matrix.

You can then apply that matrix to the factor itself in the dataframe using contrasts(data$factor) <- mat, or you can just specify it as part of the model using the contrasts argument in lm().

```
# specify contrast weights
levels(data$cond)
```

```
## [1] "Control" "A"       "B"       "C"
```

```
c1 <- c(-1 , 1/3, 1/3,  1/3)
c2 <- c( 0,   1,  -1,    0 )
c3 <- c( 0,   1,   0,   -1 )

# create temporary matrix
mat.temp <- rbind(constant=1/4, c1, c2, c3)
mat.temp
```

```
##           [,1]      [,2]       [,3]       [,4]
## constant  0.25 0.2500000  0.2500000  0.2500000
## c1       -1.00 0.3333333  0.3333333  0.3333333
## c2        0.00 1.0000000 -1.0000000  0.0000000
## c3        0.00 1.0000000  0.0000000 -1.0000000
```

```
# get the inverse of that matrix
mat <- solve(mat.temp)
mat
```

```
##      constant    c1            c2            c3
## [1,]        1 -0.75 -5.551115e-17  2.775558e-17
## [2,]        1  0.25  3.333333e-01  3.333333e-01
## [3,]        1  0.25 -6.666667e-01  3.333333e-01
## [4,]        1  0.25  3.333333e-01 -6.666667e-01
```

```
# drop the first column
mat <- mat[ , -1]

model6 <- lm(y ~ cond, data=data, contrasts=list(cond = mat))
summary(model6)
```

```
##
## Call:
## lm(formula = y ~ cond, data = data, contrasts = list(cond = mat))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -27.3811  -8.6832  -0.2297   7.5401  24.0410
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  118.578      1.038 114.195  < 2e-16 ***
## condc1        12.912      2.398   5.384 3.51e-07 ***
## condc2        -8.609      2.937  -2.931  0.00402 **
## condc3        -8.758      2.937  -2.982  0.00345 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.75 on 124 degrees of freedom
## Multiple R-squared:  0.2469, Adjusted R-squared:  0.2287
## F-statistic: 13.55 on 3 and 124 DF,  p-value: 1.056e-07
```

```
# double check your contrasts
attributes(model6$qr$qr)$contrasts
```

```
## $cond
##            c1            c2            c3
## Control -0.75 -5.551115e-17  2.775558e-17
## A        0.25  3.333333e-01  3.333333e-01
## B        0.25 -6.666667e-01  3.333333e-01
## C        0.25  3.333333e-01 -6.666667e-01
```

Remember c2 and c3 are nonorthogonal contrasts (which is why the weights look so messed up), and what we see in the regression coefficients is always the *unique* effect of each predictor. So we're seeing the difference between A and B controlling for the difference between A and C, and vice versa. Both c2

and c3 are orthogonal to c1, so it doesn't affect our interpretation of the others.

If you specify orthogonal contrasts, the regression coefficients for each contrast should just equal the difference between those group means. For example, let's run a new set of orthogonal contrasts on dose:

```
# specify contrast weights
levels(data$dose)
```

```
## [1] "None" "Low"  "Med"  "High"
```

```
NLvMH <- c(-1/2, -1/2, 1/2, 1/2)
NvL <- c( 1,  -1,   0,    0 )
MvH <- c( 0,   0,   1,   -1 )

# create temporary matrix
mat.temp <- rbind(constant=1/4, NLvMH, NvL, MvH)
mat.temp
```

```
##           [,1]  [,2] [,3]  [,4]
## constant  0.25  0.25 0.25  0.25
## NLvMH    -0.50 -0.50 0.50  0.50
## NvL       1.00 -1.00 0.00  0.00
## MvH       0.00  0.00 1.00 -1.00
```

```
# get the inverse of that matrix
mat <- solve(mat.temp)
mat
```

```
##      constant NLvMH  NvL  MvH
## [1,]        1  -0.5  0.5  0.0
## [2,]        1  -0.5 -0.5  0.0
## [3,]        1   0.5  0.0  0.5
## [4,]        1   0.5  0.0 -0.5
```

```
# drop the first column
mat <- mat[ , -1]

model7 <- lm(y ~ dose, data=data, contrasts=list(dose=mat) )
summary(model7)
```

```
##
## Call:
## lm(formula = y ~ dose, data = data, contrasts = list(dose = mat))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -33.751  -8.190   0.040   7.813  25.270
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  118.578      1.076 110.187  < 2e-16 ***
## doseNLvMH      3.179      2.152   1.477  0.14215
## doseNvL       -8.723      3.044  -2.866  0.00489 **
## doseMvH       13.232      3.044   4.347 2.84e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.18 on 124 degrees of freedom
## Multiple R-squared:  0.1911, Adjusted R-squared:  0.1715
## F-statistic: 9.765 on 3 and 124 DF,  p-value: 7.859e-06
```

```
# double check your contrasts
attributes(model7$qr$qr)$contrasts
```

```
## $dose
##      NLvMH  NvL  MvH
## None  -0.5  0.5  0.0
## Low   -0.5 -0.5  0.0
## Med    0.5  0.0  0.5
## High   0.5  0.0 -0.5
```

You can check your results against the group means to verify that your contrast estimates are operating as expected:

```
library(dplyr)
dose.means <- summarize(group_by(data, dose), y.mean=mean(y))
dose.means
```

```
## Source: local data frame [4 x 2]
##
##   dose   y.mean
## 1 None 112.6267
## 2  Low 121.3500
## 3  Med 126.7839
## 4 High 113.5517
```

Happy day! Our coefficient estimate for the first contrast (3.18) equals the average of the last two groups (126.78 + 113.55 /2 = 120.17) minus the average of the first two groups (112.63 + 121.35 /2 = 116.99).

So what happens if you try to run your own contrasts without doing the weird inverse thing? It depends. If your contrasts are orthogonal, then the t-tests and p-values you get for the regression coefficients will all be fine, but your contrast estimates (and corresponding SEs) might not match the difference between group means you expected. If your contrasts are nonorthogonal, then failing to do the weird inverse thing can result in totally garbage estimates and useless t-tests. So you MUST do this inverse thing if you specify nonorthogonal contrasts, but you should probably get in the habbit of doing it anyway for orthogonal ones as well.

Running Fewer than J-1 Contrasts for J Groups

As you know, you can get j-1 orthogonal contrasts out of a factor with j levels. What if you have lots of levels, and you really only have a couple contrasts you care about? Is it okay to run fewer than j-1 contrasts?

Yep. If you want to save time and only specify the contrast(s) you care about, you can do that, and R will come up with some orthogonal contrasts to fill in the rest. What you won't be able to do is take the inverse of your contrast weights; you can only take the inverse of a square matrix, and if you have fewer than j-1 contrasts, your temporary matrix won't be square. But remember: as long as your contrasts are orthogonal, your t-tests will all be fine even if you don't do the inverse thing. So go ahead and just use the contrasts you want directly with the contrasts() function, and be aware that your contrast estimates may not accurately reflect the differences between group means.

```
# A v B is the only contrast I care about
AvB <- c(0, 1, -1, 0)
mat <- cbind(AvB)
mat
```

```
##      AvB
## [1,]   0
## [2,]   1
## [3,]  -1
## [4,]   0
```

```
contrasts(data$cond) <- mat

model8 <- lm( y ~ cond, data=data)
summary(model8)
```

```
##
## Call:
## lm(formula = y ~ cond, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -27.3811  -8.6832  -0.2297   7.5401  24.0410
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  118.578      1.038 114.195  < 2e-16 ***
## condAvB       -4.304      1.469  -2.931 0.004022 **
## cond           7.030      2.077   3.385 0.000953 ***
## cond           9.425      2.077   4.538 1.32e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.75 on 124 degrees of freedom
## Multiple R-squared:  0.2469, Adjusted R-squared:  0.2287
## F-statistic: 13.55 on 3 and 124 DF,  p-value: 1.056e-07
```

```
attributes(model8$qr$qr)$contrasts
```

```
## $cond
##         AvB
## Control   0 -0.7071068 -0.5000000
## A         1  0.4714045 -0.1666667
## B        -1  0.4714045 -0.1666667
## C         0 -0.2357023  0.8333333
```

Note that if you add fewer than j-1 contrasts to the contrasts argument in lm(), it will NOT fill out the remaining contrasts for you. Rather, any group differences other than those represented in your contrast will get lumped into the error term!

```
model9 <- lm( y ~ cond, data=data, contrasts=list(cond=mat))
summary(model9)
```

```
##
## Call:
## lm(formula = y ~ cond, data = data, contrasts = list(cond = mat))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -33.101  -9.173  -0.838   8.166  30.238
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  118.578      1.156 102.610   <2e-16 ***
## condAvB       -4.304      1.634  -2.634   0.0095 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.07 on 126 degrees of freedom
## Multiple R-squared:  0.05218,    Adjusted R-squared:  0.04466
## F-statistic: 6.937 on 1 and 126 DF,  p-value: 0.009501
```

```
Anova(model9, type="III")
```

```
## Anova Table (Type III tests)
##
## Response: y
##              Sum Sq  Df   F value    Pr(>F)
## (Intercept) 1799778   1 10528.911 < 2.2e-16 ***
## cond           1186   1     6.937  0.009501 **
## Residuals     21538 126
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# compare those results with an ANOVA with effects coding:
model10 <- lm( y ~ cond, data=data, contrasts=list(cond=contr.sum))
Anova(model10, type="III")
```

```
## Anova Table (Type III tests)
##
## Response: y
##              Sum Sq  Df   F value    Pr(>F)
## (Intercept) 1799778   1 13040.391 < 2.2e-16 ***
## cond           5610   3    13.549 1.056e-07 ***
## Residuals     17114 124
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```