Using rockchalk for Regression Analysis*

Paul E. Johnson

January 28, 2015

The rockchalk package includes functions for estimating regressions, extracting diagnostic information from them, preparing presentable tables, and creating helpful two and three dimensional plots. It is primarily intended to facilitate teachers and students who are conducting ordinary least squares and elementary generalized models. Compared to other packages that help with regression analysis, this one offers more-than-the-usual amount of emphasis on the analysis of regression models that include interaction terms. It includes functions that can receive a fitted model and then return standardized, mean-centered, and residual-centered regression results. The plotting functions address some limitations of R's termplot function. Version 1.8 introduces a consistent framework for the creation of "newdata" objects and the calculation of predicted values so that the marginal effects of separate predictors can be easily explored.

1 Introduction

This is the Spring, 2013 update of the rockchalk package. I offer a course in regression analysis for social and behavioral scientists every year. As the course goes on, I keep track of the difficulties that the students experience with R and I craft functions that facilitate their work on particular assignments. In this edition, I offer several enhancements in the calculation of predicted values, regression plots, and regression tables.

The core of the effort this time has been development of an internally coherent framework that allows students to create "newdata" objects in a flexible and convenient way. Users are welcome to use the newdata() function directly, but I offer a convenience called predictOMatic() that allows one to specify a regression model and then receive predicted values for a range of various input values. The default output will be similar to other regression software tools that provide "marginal effect" estimates for regression analysis.

In this edition of rockchalk, I have exerted a great deal of effort to improve the consistency of predicted value plots for regression analysis. The plotSlopes() function is a plotter for linear models with interaction terms. It draws a plot on the screen, but it also creates an output object that can be subjected to a follow-up hypothesis test (by the testSlopes() function). In version 1.8, there is a new plot method for testSlopes objects that employ interactions of numeric predictors. The plots for testSlopes objects will be the most useful new feature in this version of the package.

For people who want to plot nonlinear regressions, the function plotCurves() is a replacement for plotSlopes. It can handle any sort of nonlinearity and interaction on the right hand side.

*In case you want to be involved in development, the source code is available on GitHub. Please browse http://github.com/pauljohn32. The read-only git archive address is `git://github.com/pauljohn32/rockchalk.git`. Once you learn how to "clone" the repository with a git client, then we can discuss ways for you to contribute.

Where some regression formula can cause R's termplot() function to fail, plotCurves() will succeed.

The core functionality of many functions has not been altered. Functions to provide standardized regression coefficients, mean-centered, or residual-centered regression coefficients have not changed much, although the plots of them are improved.

2 Facilitating Collection of Summary Information

2.1 summarize: A replacement for summary

When an R function provides output that is not suitable for a term paper, the student must cobble together some code and assemble a customized table. In my opinion, R's summary() function for data frames is not adequate. It lacks summaries for diversity of observations. In addition, the output is not formatted in a way that is conducive to the creation of plots or tables. As a result, I offer the function summarize(), which separates the numeric from factor variables and provides an alphabetized summary.

Consider the results of applying summarize() to Fox's Chile data set from the car package:

```
library(car)
data(Chile)
(summChile <- summarize(Chile))
```

```
$numerics
            age        income    population statusquo
0%      18.0000 2.500000e+03 3.750000e+03   -1.8030
25%     26.0000 7.500000e+03 2.500000e+04   -1.0022
50%     36.0000 1.500000e+04 1.750000e+05   -0.0456
75%     49.0000 3.500000e+04 2.500000e+05    0.9686
100%    70.0000 2.000000e+05 2.500000e+05    2.0486
mean    38.5487 3.387586e+04 1.522222e+05    0.0000
sd      14.7564 3.950287e+04 1.021980e+05    1.0002
var    217.7518 1.560477e+09 1.044444e+10    1.0004
NA's     1.0000 9.800000e+01 0.000000e+00   17.0000
N     2700.0000 2.700000e+03 2.700000e+03 2700.0000

$factors
         education                    region                       sex
 S             :1120.0000   SA            : 960.0000   F             :1379.0000
 P             :1107.0000   S             : 718.0000   M             :1321.0000
 PS            : 462.0000   C             : 600.0000   NA's          :   0.0000
 NA's          :  11.0000   N             : 322.0000   entropy       :   0.9997
 entropy       :   1.4900   M             : 100.0000   normedEntropy:   0.9997
 normedEntropy:   0.9401   NA's          :   0.0000   N             :2700.0000
 N             :2700.0000   entropy       :   2.0628
                            normedEntropy:   0.8884
                            N             :2700.0000
            vote
 N             : 889.0000
 Y             : 868.0000
 U             : 588.0000
 A             : 187.0000
 NA's          : 168.0000
 entropy       :   1.8264
 normedEntropy:   0.9132
 N             :2700.0000
```

The result object is a list that includes a data frame for the numeric variables, with named rows and (optionally) alphabetized columns, as well as a separate report for each factor variable. Users who wish to summarize only the numeric variables can run summarizeNumerics() instead, while others who want to summarize only factors can run summarizeFactors(). The output from summarizeFactors() is a list of factor summaries.

A companion function is centralValues(), which will provide only one number for each variable in a data frame. For numerics, it returns the mean, while for factor variables, it returns the mode.

```
centralValues(Chile)
```

```
  region population sex      age education   income     statusquo vote
1     SA   152222.2   F 38.54872         S 33875.86 -1.118151e-08    N
```

2.2 Easier predictions and newdata objects

Students struggle with R's predict() methods. The primary challenge is in the creation of a newdata object of interesting values of the predictors. If the newdata argument, here myNewDF in predict(m1, newdata = myNewDF) is not exactly right, the command will fail. If the regression formula uses functions like "as.numeric()" or "as.factor()", its almost impossible for first-time R users to get this right. In version 1.8, I believe I have solved the problem entirely with the functions newdata() and predictOMatic().

Let's fit an example regression with the Chile data set from the car package.

```
m1 <- lm(statusquo ~ age + income + population + region + sex, data = Chile)
```

The default output of predictOMatic() will cycle through the predictors, one by one.

```
m1pred <- predictOMatic(m1)
m1pred
```

```
$age
  age   income population region sex         fit
1  18 33868.73     151750     SA   F -0.11305577
2  26 33868.73     151750     SA   F -0.04562173
3  36 33868.73     151750     SA   F  0.03867081
4  49 33868.73     151750     SA   F  0.14825112
5  70 33868.73     151750     SA   F  0.32526546

$income
       age income population region sex          fit
1 38.60116   2500     151750     SA   F -0.018233384
2 38.60116   7500     151750     SA   F -0.005668318
3 38.60116  15000     151750     SA   F  0.013179282
4 38.60116  35000     151750     SA   F  0.063439546
5 38.60116 200000     151750     SA   F  0.478086727

$population
       age   income population region sex        fit
1 38.60116 33868.73       3750     SA   F  0.3814125
2 38.60116 33868.73      25000     SA   F  0.3353494
3 38.60116 33868.73     175000     SA   F  0.0101982
4 38.60116 33868.73     250000     SA   F -0.1523774
5 38.60116 33868.73     250000     SA   F -0.1523774

$region
       age   income population region sex         fit
1 38.60116 33868.73     151750     SA   F  0.06059664
2 38.60116 33868.73     151750      S   F  0.13954025
3 38.60116 33868.73     151750      C   F -0.05674648
4 38.60116 33868.73     151750      N   F  0.16233093
5 38.60116 33868.73     151750      M   F  0.18826330

$sex
       age   income population region sex         fit
1 38.60116 33868.73     151750     SA   F  0.06059664
2 38.60116 33868.73     151750     SA   M -0.10488518
```

The newdata() and predictOMatic() functions handle the details and allow the user to adjust their requests to allow for very fine grained control. Here are the key arguments for these functions.

1. divider. The name of an algorithm to select among observed values for which predictions are to be calculated. These are the possibilities.

   "seq" an evenly spaced sequence of values from low to high across the variable.

"quantile" quantile values that eminate from the center of the variable "outward".

"std.dev." the mean plus or minus the standard deviation, or 2 standard deviations, and so forth.

"table" when a variable has only a small set of possible values, this selects the most frequently observed values.

2. n. The number of values to be selected.

3. predVals. Where the divider argument sets the default algorithm to be used, the predVals argument can choose variables for focus and select divider algorithms separately for the variables. It is also allowed to declare particular values.

The user can request that particular values of the predictors are used, or can declare one of several algorithms for selection of focal values. All of these details are managed by the argument called predVals, which is described in the help pages (with plenty of examples).

```
mypred2 <- predictOMatic(m1, predVals = c("age", "region"), n = 3)
mypred2
```

	age	income	population	region	sex	fit
1	26	33868.73	151750	SA	F	−0.04562173
2	36	33868.73	151750	SA	F	0.03867081
3	49	33868.73	151750	SA	F	0.14825112
4	26	33868.73	151750	S	F	0.03332188
5	36	33868.73	151750	S	F	0.11761443
6	49	33868.73	151750	S	F	0.22719473
7	26	33868.73	151750	C	F	−0.16296485
8	36	33868.73	151750	C	F	−0.07867231
9	49	33868.73	151750	C	F	0.03090800

```
mypred3 <- predictOMatic(m1, predVals = c(age = "std.dev.", region = "table"), n = 3)
mypred3
```

	age	income	population	region	sex	fit
1	23.9	33868.73	151750	SA	F	−0.06332316
2	38.6	33868.73	151750	SA	F	0.06058688
3	53.3	33868.73	151750	SA	F	0.18449692
4	23.9	33868.73	151750	S	F	0.01562045
5	38.6	33868.73	151750	S	F	0.13953049
6	53.3	33868.73	151750	S	F	0.26344053
7	23.9	33868.73	151750	C	F	−0.18066629
8	38.6	33868.73	151750	C	F	−0.05675625
9	53.3	33868.73	151750	C	F	0.06715379

```
mypred4 <- predictOMatic(m1, predVals = list(age = summChile$numerics[2:4, "age"],
    region = c("SA", "C","N")), n = 3)
mypred4
```

	age	income	population	region	sex	fit
1	26	33868.73	151750	SA	F	−0.04562173
2	36	33868.73	151750	SA	F	0.03867081
3	49	33868.73	151750	SA	F	0.14825112
4	26	33868.73	151750	C	F	−0.16296485
5	36	33868.73	151750	C	F	−0.07867231
6	49	33868.73	151750	C	F	0.03090800
7	26	33868.73	151750	N	F	0.05611256
8	36	33868.73	151750	N	F	0.14040510
9	49	33868.73	151750	N	F	0.24998541

I've invested quite a bit of effort to make sure this works dependably with complicated regression formulae. All formulae, such as y ~ x1 + log(x2 + alpha) + ploy(x3, d) will just work. (In case one is interested to know *why* this works, the secret recipe is a new function called model.data(). Under the hood, this required some hard work, a frustrating chain of trial and error that is discussed in the vignette *Rchaeology*, which is distributed with this package).

The predictOMatic() function will work when the regression model provides a predict function that can handle the newdata argument. If the regression package does not provide such a function, the user should step back and use the newdata function to create the examplar predictor data frames and then calculate predicted values manually. The newdata() function will set all variables to center values and then it will create a "mix and match" combination of the ones that the user asks for. This sequence shows ways to ask for various "mix and match" combinations of values from age and region.

```
mynewdf <- newdata(m1, predVals = c("age","region"), n = 3)
mynewdf
```

```
  age   income population region sex
1  26 33868.73     151750     SA   F
2  36 33868.73     151750     SA   F
3  49 33868.73     151750     SA   F
4  26 33868.73     151750      S   F
5  36 33868.73     151750      S   F
6  49 33868.73     151750      S   F
7  26 33868.73     151750      C   F
8  36 33868.73     151750      C   F
9  49 33868.73     151750      C   F
```

```
mynewdf2 <- newdata(m1, predVals = list(age = "std.dev.", region = c("SA", "C","N")))
mynewdf2
```

```
   age   income population region sex
1 23.9 33868.73     151750     SA   F
2 38.6 33868.73     151750     SA   F
3 53.3 33868.73     151750     SA   F
4 23.9 33868.73     151750      C   F
5 38.6 33868.73     151750      C   F
6 53.3 33868.73     151750      C   F
7 23.9 33868.73     151750      N   F
8 38.6 33868.73     151750      N   F
9 53.3 33868.73     151750      N   F
```

```
mynewdf3 <- newdata(m1, predVals = list(age = c(20, 30, 40), region = c("SA", "C","N"))
    )
mynewdf3
```

```
  age   income population region sex
1  20 33868.73     151750     SA   F
2  30 33868.73     151750     SA   F
3  40 33868.73     151750     SA   F
4  20 33868.73     151750      C   F
5  30 33868.73     151750      C   F
6  40 33868.73     151750      C   F
7  20 33868.73     151750      N   F
8  30 33868.73     151750      N   F
9  40 33868.73     151750      N   F
```

Of course, functions from rockchalk or any other R package can be placed into the process for choosing focal values.

```
mynewdf <- newdata(m1, predVals = list(age = getFocal(Chile$age, n = 3), region =
    getFocal(Chile$region, n = 3)))
mynewdf
```

```
  age   income population region sex
1  26 33868.73     151750     SA   F
2  36 33868.73     151750     SA   F
3  49 33868.73     151750     SA   F
4  26 33868.73     151750      S   F
5  36 33868.73     151750      S   F
6  49 33868.73     151750      S   F
7  26 33868.73     151750      C   F
8  36 33868.73     151750      C   F
9  49 33868.73     151750      C   F
```

The function getFocal() is a generic function; it will receive variables of different types and "do the right thing." By default, focal values of numeric variables are quantile values. For factor values, the most frequently observed values are selected. These are customizable, as explained in the documentation. The newdata() output can be used in a predict() function call as demonstrated above.

It would be nice if every regression model's predicted values were accompanied by 95% confidence intervals. Models fit by lm() can supply confidence intervals, but not glm(). At the current time, there are many competing methods that might be used to calculate those intervals; predict.glm() in R's stats package avoids the issue entirely by not calculating intervals. In rockchalk-1.8, I crafted some code to calculate confidence intervals for glm objects using the (admittedly crude) Wald-based approximation. In the scale of the linear predictor, we calculate a 95% confidence interval, and then use the inverse link function to transform that onto the space of the observed response. In this example, I replicate an example that is an R classic, from the help page of predict.glm. The reader will note that the output includes a warning about the construction of the confidence interval.

```
df <- data.frame(ldose = rep(0:5, 2), sex = factor(rep(c("M", "F"), c(6, 6))),
   SF.numdead = c(1, 4, 9, 13, 18, 20, 0, 2, 6, 10, 12, 16))
df$SF.numalive <-  20 - df$SF.numdead
budworm.lg <- glm(cbind(SF.numdead, SF.numalive) ~ sex*ldose, data = df,  family =
    binomial)
predictOMatic(budworm.lg, predVals = c(ldose = "std.dev.", sex = "table"), interval = "
    confidence")
```

```
rockchalk:::predCI: model's predict method does not return an interval. We will
   improvize with a Wald type approximation to the confidence interval
   sex ldose        fit        lwr        upr
1    F -1.06 0.01881807 0.004677914 0.07258339
2    F  0.72 0.08776815 0.038579388 0.18744514
3    F  2.50 0.32553481 0.229753335 0.43851358
4    F  4.28 0.70771129 0.566928719 0.81746365
5    F  6.06 0.92393399 0.802113370 0.97326083
6    M -1.06 0.01547336 0.003572797 0.06444937
7    M  0.72 0.12874384 0.061009646 0.25153364
8    M  2.50 0.58147189 0.454577322 0.69842946
9    M  4.28 0.92888909 0.831913678 0.97181146
10   M  6.06 0.99192343 0.959388591 0.99843626
```

3 Better Regression Tables: Some outreg Examples.

On May 8, 2006, Dave Armstrong, who was a political science PhD student at University of Maryland, posted a code snippet in r-help that demonstrated one way to use the "cat" function from R to write LaTeX markup. That gave me the idea to write a LaTeX output scheme that would help create some nice looking term and research papers. I wanted "just enough" information, but not too much.

Since 2006, many new R packages have been introduced for the creation of regression tables, but I still prefer to maintain outreg. I fight to keep this simple, but have added some features in response to user requests. In rockchalk-1.8, I have compromised with users who want "more stars." Personally, I think the one-two-three stars for p-values are silly, but they are used so widely that I now have incorporated a user option to request various alphas. I've also made it easier for users to request additional summary statistics in the bottom half of the table.

In the following, I will demonstrate some tables for lm and glm fits on a simulated data set. This new simulation function, genCorrelatedData2(), is a convenient way to create multiple-predictor regression data sets of arbitary size and complexity, allowing for interactions and nonlinearity.

```
set.seed(1234)
dat <- genCorrelatedData2(N = 100, means = c(0, 10, 0), sds = c(1, 2, 1), rho = c(0, 0,
    0),
```

Table 1: Default Outreg Table

	M1 Estimate (S.E.)	M2 Estimate (S.E.)	M3 Estimate (S.E.)
(Intercept)	-1.542 (5.865)	-3.193 (6.137)	-1.740 (5.948)
x1	-3.238** (0.974)	. 	-3.187** (1.000)
x2	4.147*** (0.584)	4.320*** (0.611)	4.171*** (0.594)
x3	. 	. 	-0.274 (1.115)
N	100	100	100
RMSE	10.603	11.134	10.655
R^2	0.406	0.338	0.406
adj R^2	0.394	0.331	0.388

$*p \leq 0.05 ** p \leq 0.01 *** p \leq 0.001$

```
    stde = 10, beta = c(0, -3, 4, 0), verbose = FALSE)
```

That creates a new matrix with variables $x1$, $x2$, $x3$, and y. We run some linear regressions and then create a categorical output variable for a logistic regression.

```
m1 <- lm(y ~ x1 + x2, data = dat)
m2 <- lm(y ~ x2, data = dat)
m3 <- lm(y ~ x1 + x2 + x3, data = dat)
## Create categorical variant
myilogit <- function(x) exp(x)/(1 + exp(x))
dat$y3 <- rbinom(100, size = 1, p = myilogit(scale(dat$y)))
gm1 <- glm(y3 ~ x1 + x2, data = dat)
```

The outreg examples are offered in Tables 1 through 3. Table 1 is the default output for three models, obtained from outreg(list (m1, m2, m3)). On the other hand, users can pack output estimates into a single column and ask for more stars, as illustrated in Table 2. In Table 3, observe that the linear and generalized linear model output peacefully co-exist, side-by-side. This output is, in my opinion, completely acceptable for inclusion in a presentation or conference paper. There are some warts in this output. Because the model labels are not equal in length, the columns are not equally sized. Because many LaTeX distributions do not include the dcolumns package, I've not aligned the coefficient estimates vertically as some users might prefer. It seems to me that, if the vertical alignment of estimates along the decimal is required, most users will install dcolumn and make a few obvious changes in the markup that I provide.

In rockchalk 1.8, a translator from LaTeX markup to simple HTML was introduced. The function outreg2HTML() can write an HTML code file that a word processor program such as Libre Office or Microsoft Word is able to import without a struggle.

4 Plotting Regressions with Interactions

4.1 Interaction in Linear Regression.

One of the most fundamental skills in regression analysis is the interpretation of interactive predictors. It is much easier for students to understand the effect of an interaction if they can create a nice plot to show how the predicted value depends on various values of a predictor. The plotSlopes() function was introduced in 2010 when I was teaching a large first-year graduate course (more than 50 students) and it became apparent that about 20 percent of them would not

Table 2: My Spread Out Regressions

	The First Model with a Long Title		Another Model	
	Estimate	(S.E.)	Estimate	(S.E.)
(Intercept)	-1.542	(5.865)	-1.740	(5.948)
x1	-3.238**	(0.974)	-3.187**	(1.000)
x2	4.147***	(0.584)	4.171***	(0.594)
x3	.		-0.274	(1.115)
N	100		100	
RMSE	10.603		10.655	
R^2	0.406		0.406	
adj R^2	0.394		0.388	

$*p \leq 0.05** p \leq 0.01***p \leq 0.001$

Table 3: Combined OLS and GLM Estimates

	OLS:y	GLM: Categorized y
	Estimate	Estimate
	(S.E.)	(S.E.)
(Intercept)	-1.542	0.063
	(5.865)	(0.275)
x1	-3.238**	-0.041
	(0.974)	(0.046)
x2	4.147***	0.040
	(0.584)	(0.027)
N	100	100
RMSE	10.603	
R^2	0.406	
adj R^2	0.394	
Deviance		24.051
$-2LLR(Modelchi^2)$		0.789

$*p \leq 0.05** p \leq 0.01***p \leq 0.001$

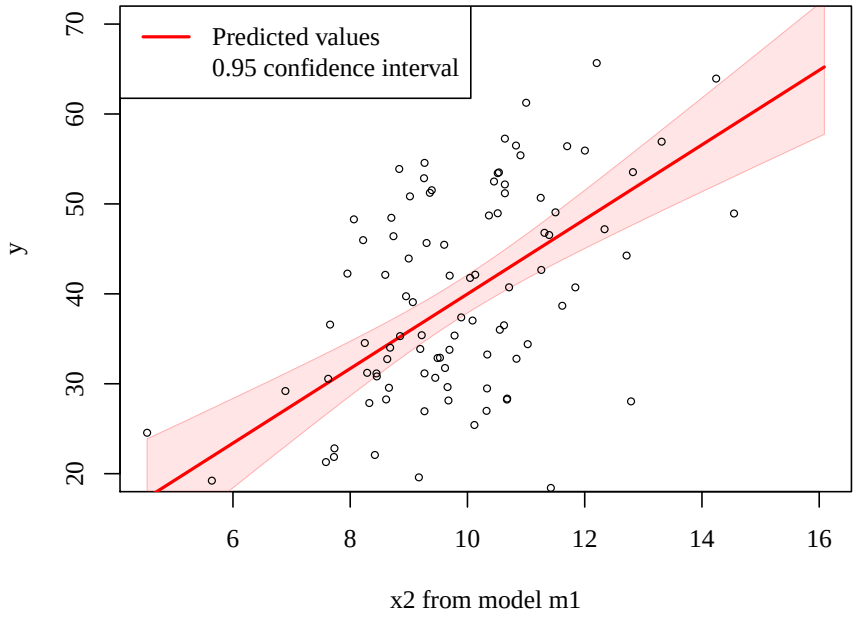

Figure 1: plotSlopes: Linear Model with Confidence Interval

be able to manage the R coding required to draw several lines on a single plot. Unfortunately, R's termplot() function will not draw regressions involving interactions.

The rockchalk package has two functions to help with this, plotSlopes() and plotCurves(). plotCurves() is more general, it can handle any kind of formula that the user estimates. plotSlopes() is more limited, it is only for lm objects. In return for that limitation, plotSlopes() creates an output object which can be used to conduct post-hoc hypothesis tests.

At its most elementary level, plotSlopes() is a "one step" regression line plotter. If the regression model includes more than one predictor, then a single predictor is displayed on the horizontal axis and the other predictors are set on their central values. A plot for the model m1, that was illustrated above, is presented in Figure 1. In rockchalk-1.8, new arguments were added to allow the "see though" confidence region. The command to generate Figure 1 was

```
m1ps <- plotSlopes(m1, plotx = "x2", xlab = "x2 from model m1", interval = "confidence"
    , opacity = 80, col = "red", ylim = c(20, 70))
```

I've adjusted the color and opacity to illustrate the usage of those arguments. The y range is adjusted to make a little extra room for the legend. The plotSlopes() function is very flexible. All of the label, color, and scale arguments of a plot function are also available. The plotSlopes function also works well if the moderator is a categorical variable.

It is important to note that the output object, m1ps, has the information necessary to recreate the plotted line in the form of a newdata data frame. The first few lines in the newdata object are

```
m1ps$newdata[1:3, ]
```

```
           x1       x2       fit       lwr       upr
1 -0.01591005 4.535561 17.32046 10.77693 23.86398
2 -0.01591005 4.831765 18.54895 12.32943 24.76848
3 -0.01591005 5.127970 19.77744 13.87975 25.67514
```

This is more interesting if we fit a regression with an interaction term, such as

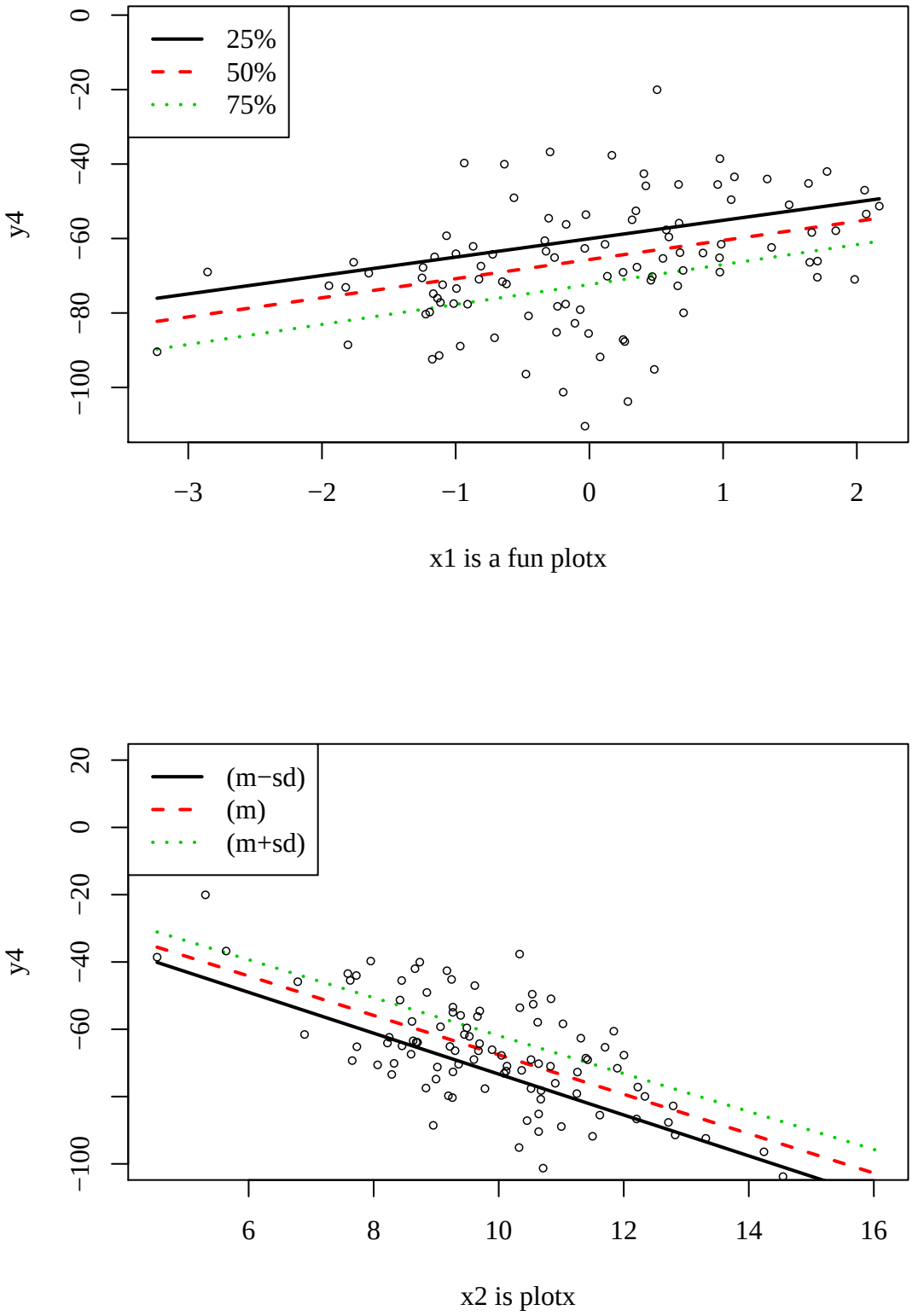

Figure 2: plotSlopes Illustrated

```
m4 <- lm(y4 ~ x1*x2 + x3, data = dat)
```

We then ask plotSlopes to draw the predicted values using one numeric variable as the horizontal axis and values of another variable (a moderator) are set at particular values. Either x1 or x2 can be viewed as the "moderator" predictor, the one on which the effect of the other depends. In rockchalk version 1.8, the selection of values of the moderator was generalized, so that the user can specify either a function that selects values, or a vector of values, or the name of an algorithm. The default algorithm will choose quantile values, but Figure 2 demonstrates also the "std.dev." divider algorithm. The code to produce that figure was

```
par(mfcol=c(2,1))
m4psa <- plotSlopes(m4, plotx = "x1", modx = "x2", xlab = "x1 is a fun plotx")
m4psb <- plotSlopes(m4, plotx = "x2", modx = "x1", modxVals = "std.dev.", xlab = "x2 is
    plotx", ylim = c(-100, 20))
par(mfcol=c(1,1))
```

When modx is a numeric variable, then some particular values must be selected for calculation of predicted value lines. The modxVals argument is used to either specify moderator values or an algorithm to select focal values. By default, three hypothetical values of plotx are selected (the quantiles 25%, 50%, and 75%).

If modx is a factor variable, then the most frequently observed scores will be selected for consideration. The default display will include the regression line as well as color-coded points for the subgroups represented by values of the moderator.

Suppose we have a four-valued categorical variable, "West","Midwest", "South", and "East". If that variable is used in an interaction in the regression model, then the plotSlopes output will include four lines, one for each region. For example, consider Figure 3, which is created by

```
m5psa <- plotSlopes(m5, plotx = "x1", modx = "x4", xlab = "x1 is a Continuous Predictor
    ", xlim = magRange(dat$x1, c(1.2,1)))
```

The categorical variable is x4.

It is possible to superimpose confidence intervals for many subgroups, but sometimes these plots start to look a little bit "busy". The mixing of shades in overlapping intervals may help with that problem. A plot that focuses on just two subgroups is presented in Figure 4, which is produced by

```
m5psb <- plotSlopes(m5, plotx = "x1", modx = "x4", modxVals = c("West","East"), xlab =
    "x1 is a Continuous Predictor", xlim=magRange(dat$x1, c(1.2,1)), interval = "conf")
```

In rockchalk version 1.8, I've exerted quite a bit of effort to make sure that colors are chosen consistently when users remove or insert groups in these plots. The same value of the moderator should always be plotted in the same way–the line, points, and interval colors should not change. Note, for example, in Figures 3 and 4, the line for East is black in both plots, while the line for West is red in both.

4.2 testSlopes, a companion of plotSlopes

The students in psychology and political science are usually interested in conducting some diagnostic analysis of the interactive terms. Aiken and West (1991) (and later Cohen et al., 2002Cohen, Cohen, West, and Aiken) propose using the t test to find out if the effect of the "plotx" variable is statistically significantly different from zero for each particular value of the moderator variable. The new version of rockchalk declares a method plot.testSlopes that handles the work of plotting the interaction.

The usual case would be the following. First, carry out the regression analysis. Then run plotSlopes, and run testSlopes, and then pass that result object to the plot method.

```
m4 <- lm(y ~ x1*x2 + x3, data = dat)
m4ps <- plotSlopes(m4, plotx = "x1", modx ="x2", xlab = "x1 is a
   Continuous Predictor")
```

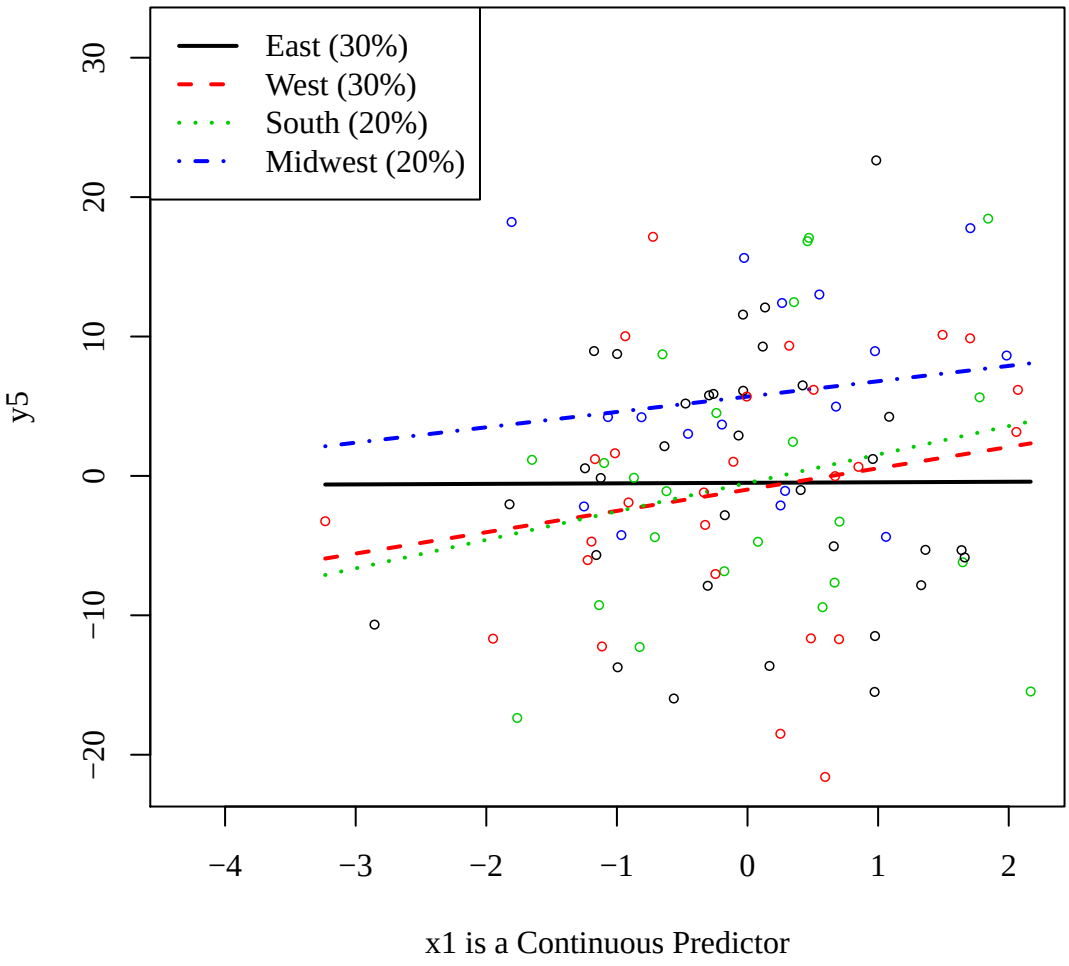

Figure 3: plotSlopes with a Categorical Moderator

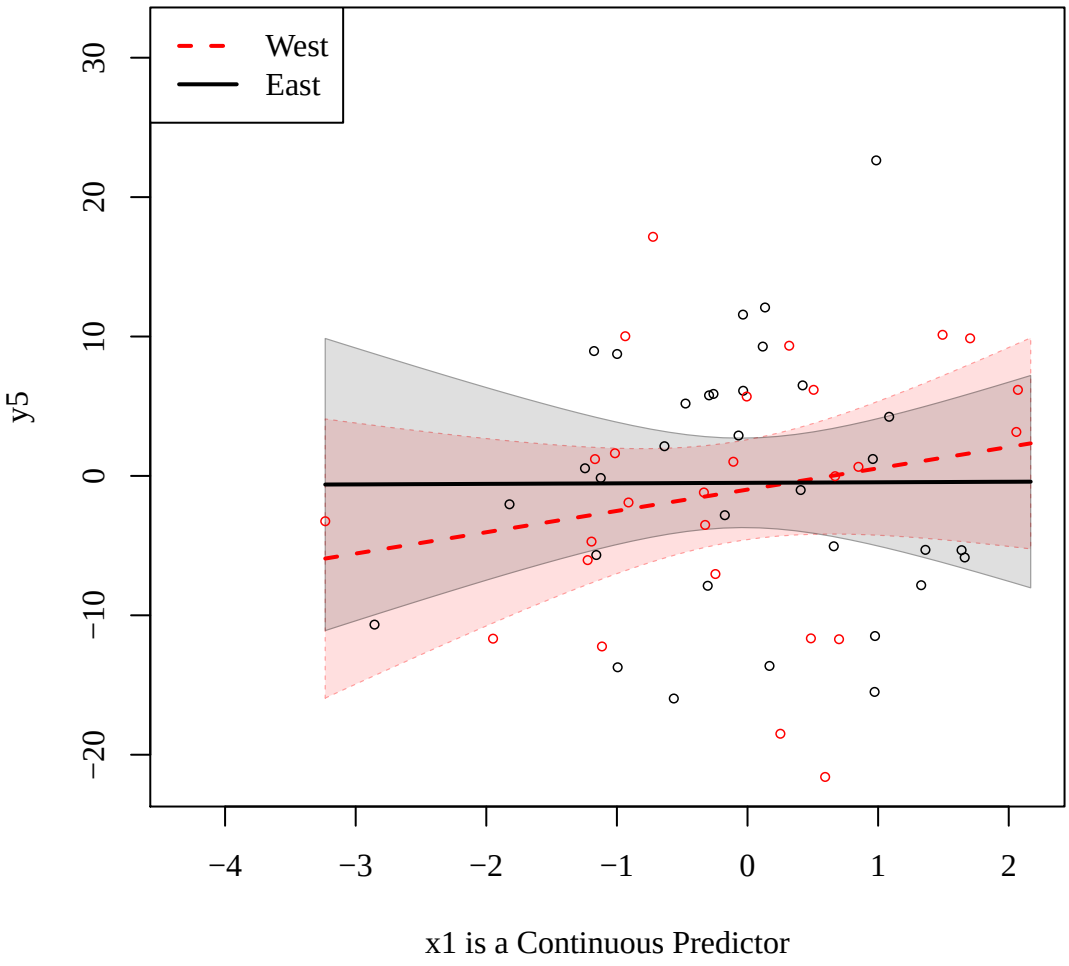

Figure 4: plotSlopes: the interval argument

```
Values of x2 INSIDE this interval:
       lo        hi
 6.987883 13.129219
cause the slope of (b1 + b2*x2)x1 to be statistically significant
```

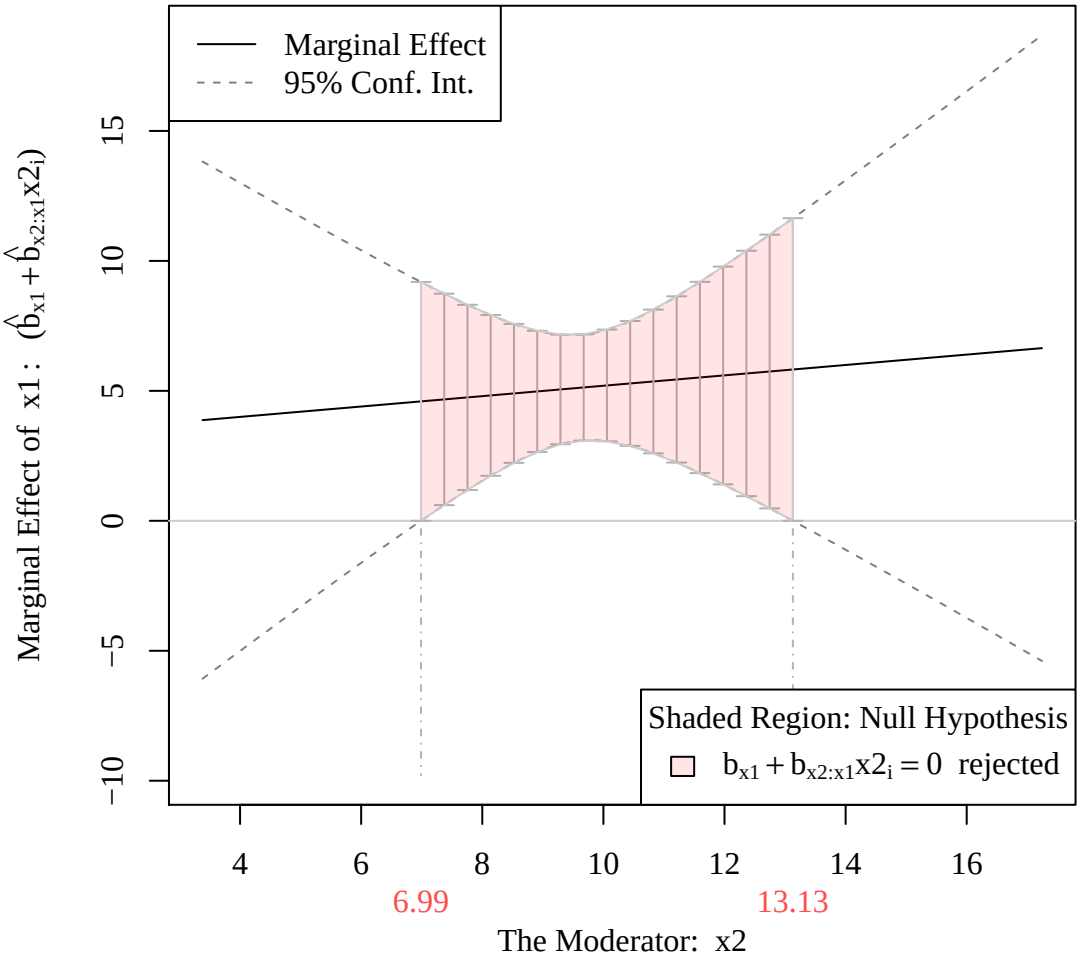

Figure 5: testSlopes for an Interactive Model

```
m4psats <- testSlopes(m4ps)
plot(m4psats)
```

The output from testSlopes will differ, depending on whether modx is numeric or a factor. If it is a factor, then the slope of the lines and the associated t-test for each will be presented. My psychology students call these "simple-slopes", following the terminology of Aiken and West. The general idea is that we want to know if the "combined" effect of plotx is not zero. For a model stated with predictors $plotx_i$ and $modx_i$as

$$y_i = b_0 + b_{plotx} plotx_i + b_{modx} modx_i + b_{plotx:modx} plotx_i \cdot modx_i + \ldots + e_i \tag{1}$$

the null hypothesis would be

$$H_0 : 0 = \hat{b}_{simple\,slope} = \hat{b}_{plotx} + \hat{b}_{plotx:modx} modx \tag{2}$$

If modx is a factor, then we simply calculate the slope of each line and the test is straight-forward. If modx is a numeric variable, then we confront a problem that is a bit more interesting. We don't really want to say that the simple slope is different from 0 for particular values of $modx$, but instead we want to answer the question, "for which values of the moderator would the effect of the plotx variable be statistically significant?". This necessitates the calculation of the so-called Johnson-Neyman interval (1936), a plot of which is presented in Figure 5.

The method of calculation is outlined in Preacher et al. (2006). The values of $modx$ associated with a statistically significant effect of $plotx$ on the outcome is determined from the computation of a T statistic for $\hat{b}_{simple\,slope}$. The J-N interval is the set of values of $modx$ for which the following (quadratic equation) holds:

$$\hat{t} = \frac{\hat{b}_{simple\,slope}}{std.err(\hat{b}_{simple\,slope})} = \frac{\hat{b}_{simple\,slope}}{\sqrt{\widehat{Var(\hat{b}_{plotx})} + modx^2 \widehat{Var(\hat{b}_{plotx \cdot modx})} + 2modx \widehat{Cov(\hat{b}_{plotx}, \hat{b}_{plotx \cdot modx})}}} \geq T_{\frac{\alpha}{2}, df} \tag{3}$$

Suppose there are two real roots, $root1$ and $root2$. The values of $modx$ for which the slope is statistically significant may be a compact interval, $[root1, root2]$, as demonstrated in Figure 5, or it may two open intervals, $(-\infty, root1]$ and $[root2, \infty)$. I had expected almost all applications to result in that latter case, but the somewhat surprising case illustrated in Figure 5 is not too infrequently observed.

4.3 plotCurves for nonlinear predictor formulae

plotCurves() generalizes the plotting capability of plotSlopes. plotCurves() should be able to handle any regression formulas that include nonlinear transformations. Models that have polynomials or terms that are logged (or otherwise transformed) can be plotted. In that sense, plotCurves() is rather similar to R's own termplot() function, but plotCurves() has two major advantages. First, it allows interactions, and second, it handles some complicated formulae that termplot() is not able to manage.

Suppose a dependent variable y5 is created according to a nonlinear process.

$$y5_i = -3x1_i + 7 * log(x2) + 1.1x2_i + 2.2x1_i \times x2_i + e_i \tag{4}$$

We fit a model with an elaborate specification and create the plot in Figure 6 with these commands:

```
m5 <- lm(y5 ~ log(x2) + x1 * x2, data = dat)
m5pc <- plotCurves(m5, plotx = "x2", modx = "x1")
```

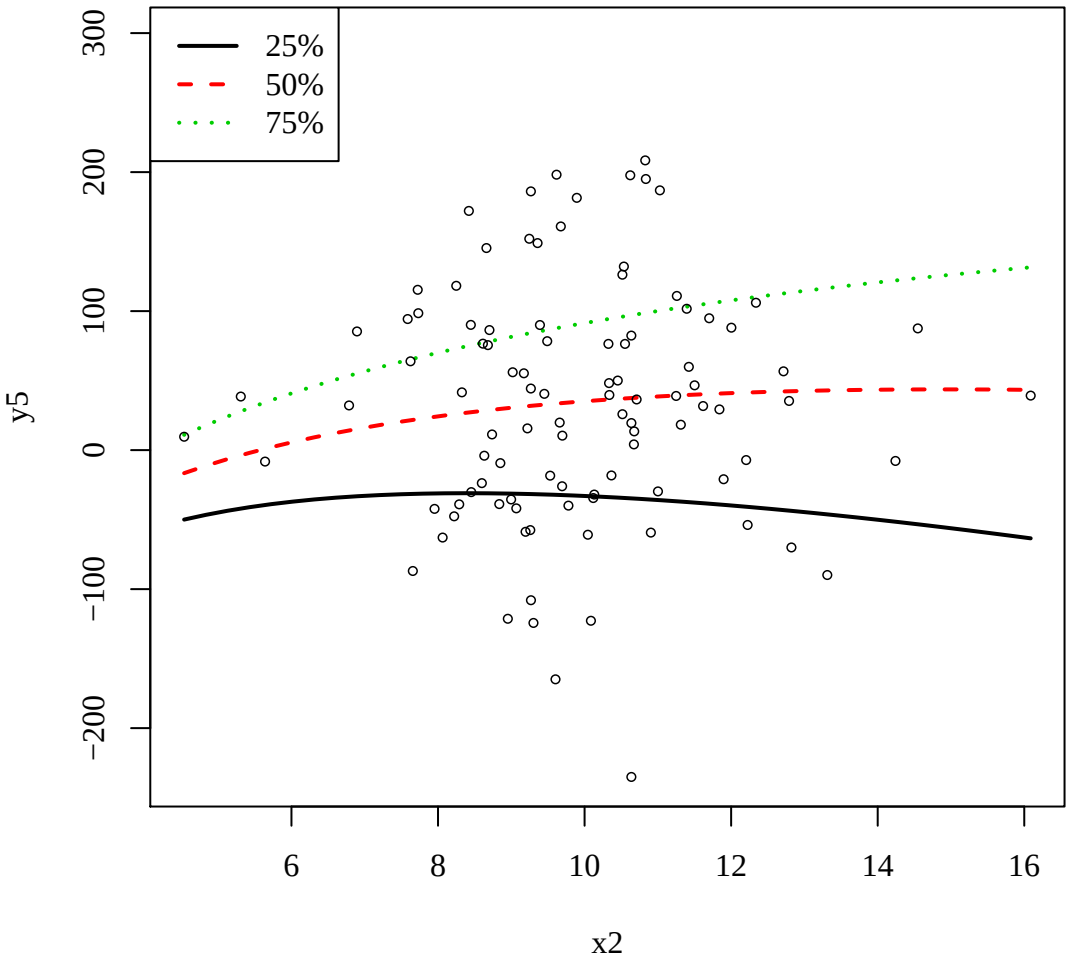

Figure 6: plotCurves

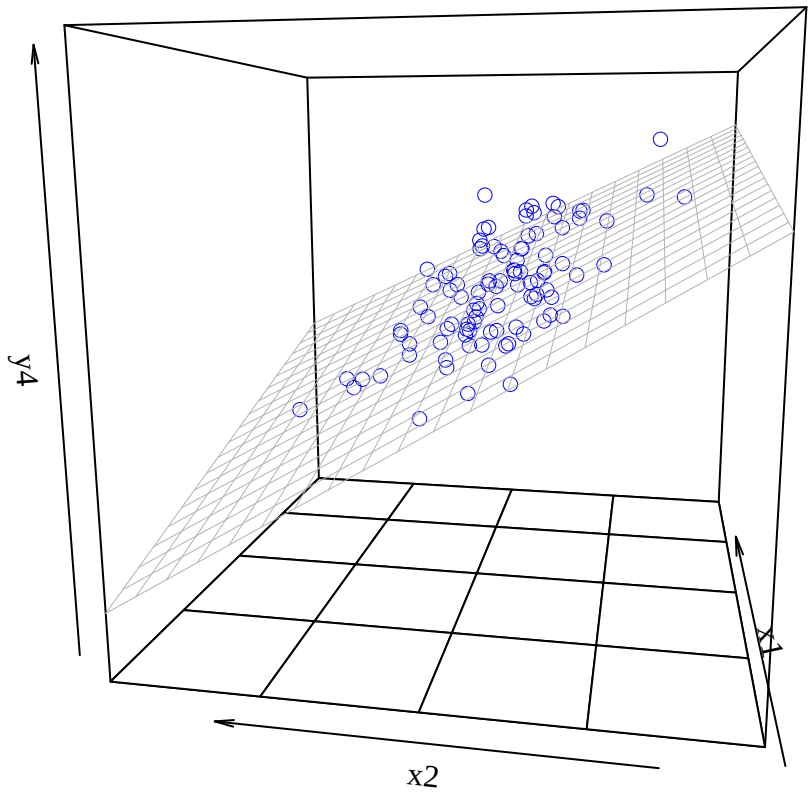

Figure 7: plotPlane for the Interactive Model m4

4.4 plotPlane

The persp function in R works well, but its interface is too complicated for most elementary and intermediate R users. To facilitate its use for regression users, the plotPlane() function is offered.

The plotPlane function offers a visualization of the mutual effect of two predictors, whether or not the regression model is linear. plotPlane() is designed to work like plotCurves(), to tolerate nonlinear components in the regression formula. plotPlane() allows the depiction of a 3 dimensional curving plane that "sits" in the cloud of data points. The variables that are not explicitly pictured in the plotPlane() figure are set to central reference values. Recall model m4, which used the formula y4 ~x1*x2 + x3. As illustrated in Figure 7, plotCurves() presents a reasonable view of the predicted values.

Because plotPlane() is a simple convenience wrapper for R's persp() function, it responds to the same customizing arguments that perp would allow. The arguments phi and theta will rotate the figure, for example. The output in Figure 7 is produced by the following.

```
p100 <- plotPlane(m4, plotx1 = "x1", plotx2 = "x2", phi = 10, theta = -80, lcol = gray(
    .70))
```

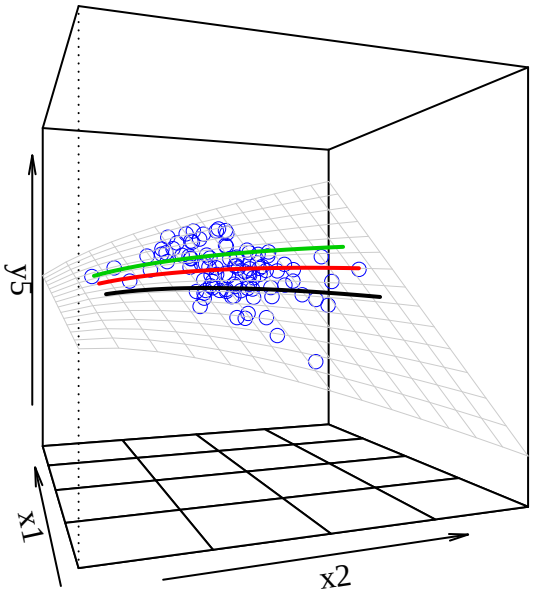

Figure 8: Making plotSlopes and plotPlane Work Together

One of the major educational benefits of the 3-D figure is that students can easily see that a model with a simple interaction effect is *not a linear model any more.* We will return to that point in the discussion of mean centering in regression analysis.

Recall that model m5 is the rather complicated nonlinear formula log(x2∗x2) + x1 ∗ x2. The plotCurves() output for that was already presented in Figure 6. The three dimensional view of the same is presented in Figure 8, but with an added twist. The twist is that the predicted value lines from the 2-D plot functions can be superimposed on the plane. The function addLines() does the work for translating 2-D plot object onto the regression plane.

5 Standardized, Mean-Centered, and Residual-Centered Regressions

5.1 Standardized regression

Many of us learned to conduct regression analysis with SPSS, which reported both the ordinary (unstandardized) regression coefficients as well as a column of "beta weights", the output of a "standardized" regression analysis. Each variable, for example $x1_i$, was replaced by an estimated $Z - score : (x1_i - \overline{x1})/std.dev.(x1_i)$. Some people think these coefficients are easier to interpret (but, for a strong cautionary argument against them, see King, 1986). R offers no such thing as standardized regression, probably because this practice is practice is thought to be mistaken. The automatic standardization of all predictors, no matter whether they are categorical, interaction terms, or transformed values (such as logs) is dangerous.

To illustrate that, the rockchalk introduces a function called standardize(). Each column of the design matrix is scaled to a new variable with mean 0 and standard deviation 1. The result from standardize() will be an R lm object, which will respond to any follow-up analysis commands. For example:

```
m4 <- lm (y4 ~ x1 * x2, data = dat)
m4s <- standardize(m4)
```

I doubt that a reasonable person would actually want a standardized regression and have tried to warn users in the output.

```
summary(m4s)
```

```
All variables in the model matrix and the dependent variable
were centered. The centered variables have the letter "s" appended to their
non-centered counterparts, even constructed
variables like `x1:x2` and poly(x1,2). We agree, that's probably
ill-advised, but you asked for it by running standardize().

The rockchalk function meanCenter is a smarter option, probably.

The summary statistics of the variables in the design matrix.
         mean std.dev.
y4s         0        1
x1s         0        1
x2s         0        1
`x1:x2s`    0        1

Call:
lm(formula = y4s ~ -1 + x1s + x2s + `x1:x2s`, data = stddat)

Residuals:
     Min       1Q   Median       3Q      Max
-1.72507 -0.46354 -0.03264  0.40507  1.87269

Coefficients:
         Estimate Std. Error t value Pr(>|t|)
x1s       0.17451    0.50032   0.349    0.728
x2s      -0.65681    0.06654  -9.871 2.55e-16 ***
`x1:x2s`  0.15602    0.49986   0.312    0.756
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6575 on 97 degrees of freedom
Multiple R^2:  0.5764, Adjusted R^2:  0.5633
F-statistic: 44.01 on 3 and 97 DF,  p-value: < 2.2e-16
```

5.2 Mean-centered Interaction Models

Sometimes people will fit a model like this

$$y_i = b_o + b_1 x1_i + b_2 x2_i + e_i \tag{5}$$

and then wonder, "is there an interaction between $x1_i$ and $x2_i$?" The natural inclination is to run this model,

```
m1 <- lm(y ~ x1*x2)
```

or its equivalent

```
m2 <- lm(y ~ x1 + x2 + x1:x2)
```

Researchers have been advised that they should not run the ordinary interaction model without "mean-centering" the predictors (Aiken and West, 1991). They are advised to replace $x1_i$ with $(x1_i - \overline{x1})$ and $x2_i$ with $(x2_i - \overline{x2})$, so that the fitted model will

$$y_i = b_o + b_1(x1_i - \overline{x1}) + b_2(x2_i - \overline{x2}) + b_3(x1_i - \overline{x1})(x2_i - \overline{x2}) + e_i \tag{6}$$

Table 4: Comparing Ordinary and Standardized Regression

	Not Standardized		Standardized	
	Estimate	(S.E.)	Estimate	(S.E.)
(Intercept)	-8.211	(6.077)	.	
x1	2.627	(7.572)	.	
x2	-5.929***	(0.604)	.	
x1:x2	0.242	(0.781)	.	
x1s	.		0.175	(0.500)
x2s	.		-0.657***	(0.067)
‘x1:x2s‘	.		0.156	(0.500)
N	100		100	
RMSE	10.934		0.657	
R^2	0.576		0.576	
adj R^2	0.563		0.563	

$*p \leq 0.05 ** p \leq 0.01 *** p \leq 0.001$

This is a little tedious to do in R, so I provide a function meanCenter() that can handle the details. meanCenter() will receive a model, scan it for interaction terms, and then center the variables that are involved in interactions. We previously fit the model m4, and now we center it.

```
m4mc <- meanCenter(m4)
summary(m4mc)
```

```
These variables were mean-centered before any transformations were made on the design
    matrix.
[1] "x1c" "x2c"
The centers and scale factors were
              x1c      x2c
mean  -0.01591005 9.882781
scale  1.00000000 1.000000
The summary statistics of the variables in the design matrix (after centering).
             mean std.dev.
y4       -66.93072 16.54351
x1c        0.00000  1.09885
x2c        0.00000  1.83261
x1c:x2c   -0.17691  1.44789

The following results were produced from:
meanCenter.default(model = m4)

Call:
lm(formula = y4 ~ x1c * x2c, data = stddat)

Residuals:
     Min       1Q   Median       3Q      Max
-28.5387  -7.6686  -0.5399   6.7013  30.9809

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -66.8878     1.1021 -60.694  < 2e-16 ***
x1c           5.0238     1.0263   4.895 3.97e-06 ***
x2c          -5.9331     0.6049  -9.809 3.83e-16 ***
x1c:x2c       0.2425     0.7809   0.311    0.757
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.93 on 96 degrees of freedom
Multiple R^2:  0.5764, Adjusted R^2:  0.5632
F-statistic: 43.55 on 3 and 96 DF,  p-value: < 2.2e-16
```

By default, meanCenter() will only center the variables involved in an interaction, and it leaves the others unchanged. The user can request a different treatment of the variables. Version 1.8

introduces the argument "terms", which allows the user to list the names of the predictors that should be centered. If the user wants all of the numeric predictors to be mean-centered, the usage of the argument centerOnlyInteractors would be appropriate:

```
m4mc <- meanCenter(m4, centerOnlyInteractors = FALSE)
```

By default, it does not standardize while centering (but the user can request standardization with the argument standardize = TRUE. The option centerDV causes the dependent variable to be centered as well.

5.3 Residual-centered Models

Residual-centering (Little et al., 2006)is another adjustment that has been recommended for models that include interactions or squared terms. Like mean-centering, it is often recommended as a way to obtain smaller standard errors or to make estimates more numerically stable. Like mean centering, it causes a completely superficial change in the estimated coefficients. Nothing of substance is altered.

The residualCenter() function is used in the same manner as meanCenter(). The user fits an interactive model and the result object is passed to residualCenter() like so:

```
m4rc <- residualCenter(m4)
summary(m4rc)
```

```
Call:
lm(formula = y4 ~ x1 + x2 + x1.X.x2, data = mfnew)

Residuals:
     Min       1Q   Median       3Q      Max
-28.5387  -7.6686  -0.5399   6.7013  30.9809

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -8.3976     6.0476  -1.389    0.168
x1            4.9576     1.0040   4.938 3.33e-06 ***
x2           -5.9148     0.6020  -9.825 3.53e-16 ***
x1.X.x2       0.2425     0.7809   0.311    0.757
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.93 on 96 degrees of freedom
Multiple R^2:  0.5764,  Adjusted R^2:  0.5632
F-statistic: 43.55 on 3 and 96 DF,  p-value: < 2.2e-16
```

I would explain residual-centering as follows. Suppose we fit the linear model, with no interaction (note, I'm calling the coefficients c_j, not b_j as usual):

$$y = c_0 + c_1 x1 + c_2 x2 + e_i. \tag{7}$$

Let's proceed as if those parameter estimates, $\hat{c}_1$, $\hat{c}_2$, are the "right ones" for our analytical purpose. We'd like to fit an interactive model, but protect the linear parts from fluctuation. In R, when we run th lm(y ~x1 * x2), we are allowing all of the coefficients to fluctuate

$$y_i = b_o + b_1 x1_i + b_2 x2_i + b_3 x1_i \times x2_i + e_i \tag{8}$$

Residual centering is one way to stabilize the estimation by assuring that $\hat{b}_1 = \hat{c}_1$ and $\hat{b}_2 = \hat{c}_2$. Only the coefficient $\hat{b}_3$ floats freely.

One of the reasons that residual-centering is so appealing is that its stabilizing benefit is obtained almost by accident. Here is the gist of the calculations. First, estimate a regression in which the left hand side is the interaction product term:

$$(x1_i \times x2_i) = d_0 + d_1 x1_i + d_2 x2 + u_i \tag{9}$$

The residuals from that regression are, by definition, orthogonal to both $x1$ and $x2$. Call those fitted residuals $\widehat{u}_i$. The we run the interactive regression, replacing the column of the predictor $x1_i \times x2_i$, with $\widehat{u}_i$. That is to say, the model we want, equation (8), is estimated as:

$$y_i = b_0 + b_1 x1_i + b_2 x2_i + b3\widehat{u}_i + e_i, \tag{10}$$

In essence, we have taken the interaction $(x1_i \times x2_i)$, and purged it of its parts that are linearly related to $x1_i$ and $x2_i$.

rockchalk 1.6 included summary, print, and predict methods for the residual-centered regression objects. It is worth mentioning that the code can handle interactions of arbitrarily many predictors. If the formula has lm(y ∼x1 ∗ x2 ∗ x3 ∗ x4), for example, this implies many separate interactions must be calculated. We need to calculate residual-centered residuals for $x1 \cdot x2$, $x1 \cdot x3$, $x1 \cdot x4$, $x2 \cdot x3$ and so forth, and then use them as predictors to get centered estimates of terms $x1 \cdot x2 \cdot x3$, and then their centered values are predictors four term interactions. Aspiring R programmers who want to learn about programming with R formula objects might benefit from the study of the function residualCenter.R in the rockchalk source code.

6 A Brief Analysis of Mean-Centering

We can put the tools together by making a little detour into the question that seems to plague every regression analyst at one time or another: What does that interaction term *really mean*? Along the way, we will try to dispel the idea that centering somehow makes estimates "better" or more numerically precise. The primary advocates of centering as a way to deal with numerical instability are Aiken and West (1991), who integrated that advice into the very widely used regression textbook, *Applied Multiple Regression/Correlation for the Behavioral Sciences* (Cohen et al., 2002). They claim that the inclusion of interactions causes "inessential multicollinearity" that is alleviated by centering. The advice is widely followed. One statistics book intended for biologists observed, for example, "We support the recommendation of Aiken & West (1991) and others that multiple regression with interaction terms should be fitted to data with centered predictor values" (Quinn, 2002, Chapter 6).

Technical rebuttals have been published already (Kromrey, 1998), but the matter still seems not widely understood. The argument is not that mean-centering (or residual-centering) is wrong, but rather that it is unnecessary. It is irrelevant, and possibly, misleading.

At the core of the matter is the fact that our uncertainty about regression estimates depends on our point of view. Please review the confidence interval in Figure 1. The y axis is not even "in the picture." Would one's appreciation of the regression's predictive line be enhanced if the y axis were moved into the picture?

We can move the y axis by centering the predictor variable. Suppose we replace x_i with $x_i - 5$ and then re-estimate the model. That has the simple effect of moving the y axis 5 units to the right. The slope is unchanged, and the reported intercept is changed in a completely superficial way. The predicted value line, the slope estimates, the residual standard error, and so forth, are not changed in any substantial way. This is simply a matter of user convenience. I believe that no reasonable person can say the regression is "better" after centering x_i.

However, there is a superficial difference that has deceived many authors. Notice that the confidence interval is hourglass shaped. *At the new y axis*, our prediction is more precise. If we move the y axis further to the right, into the center of the data, say by mean-centering $(x_i - 10)$, we move to the position that allows an even more precise prediction. The estimate of the intercept's standard error will be smaller for the obvious reason. We are not actually gaining certainty, we are simply reporting our most favorable "snapshot" of it. The predicted value, and the confidence interval for any observed value of x_i is completely unchanged by centering.

It should not surprise the reader to learn that mean-centering interactive predictors enhances the standard errors in the same illusory way. Let's work though an example to see why this is

Table 5: Comparing Regressions

	Linear		Interaction		Mean-centered		Residual-centered	
	Estimate	(S.E.)	Estimate	(S.E.)	Estimate	(S.E.)	Estimate	(S.E.)
(Intercept)	-466.657**	(92.426)	229.946	(351.536)	519.565**	(17.086)	-466.657**	(92.054)
x1	9.471**	(1.838)	-4.720	(7.150)	.		9.471**	(1.831)
x2	10.400**	(1.878)	-4.102	(7.306)	.		10.400**	(1.871)
x1:x2	.		0.289*	(0.141)	.		.	
x1c	.		.		9.794**	(1.838)	.	
x2c	.		.		10.528**	(1.872)	.	
x1c:x2c	.		.		0.289*	(0.141)	.	
x1.X.x2	.		.		.		0.289*	(0.141)
N	400		400		400		400	
RMSE	311.193		309.940		309.940		309.940	
R^2	0.233		0.241		0.241		0.241	
adj R^2	0.229		0.236		0.236		0.236	

$*p \leq 0.05** p \leq 0.01$

so tempting. Suppose the true data generating mechanism is an interaction like so

$$y_i = 2 + 0.1x1_i + 0.1x2_i + 0.2 \cdot (x1_i \times x2_i) + e_i, \tag{11}$$

where $e_i \sim N(0, 300^2)$ and $\rho_{x1,x2} = 0.4$.

A regression analysis that ignores the interaction,

```
lm(y ~ x1 + x2, data = dat2)
```

is reported in the first column of Table 5. I've used outreg's new alpha argument to emphasize the "really good" estimates with more stars. Notice that everything is "statistically significant!"

Unable to leave well enough alone, the researcher wonders, "is there an interaction between $x1$ and $x2$?" Run

```
lm(y ~ x1 * x2, data = dat2)
```

The second column in Table 5 summarizes that regression. Be prepared for a shock when you scan the estimates. Almost everybody I know has said "what the heck?" or "Holy Cow!" or "Oh My God, my great result went to Hell, I'll never get tenure!" Neither of the key variables is "statistically significant" any more.

Cohen, et al. claim that the apparent instability of the coefficients is a reflection of "inessential collinearity," due to the fact that $x1$ and $x2$ are correlated with the new term, $x1 \times x2$. They advised their readers to "mean-center" their predictors and run the regression again.

Remember the hourglass shape of the confidence interval. By mean-centering, we are repositioning ourself for a much more favorable snapshot. The welcoming effect of the centered estimates is found in the third column of Table 5. The point estimates in that snapshot are "significant again." It appears we have "solved" the problem of inessential collinearity.

The first hint of trouble is in the fact that the coefficients of the interactive effects in columns 2 and 3 are identical. Those coefficients are the same because they are estimates of the cross partial derivative $\partial^2 y/\partial x1 \partial x2$. That particular value is the same, no matter where we position the y axis, as it should be. Note as well that the root mean square and R^2 estimates are identical. Everything that we expect to remain the same is the same. Except for the fact that the slopes and their hypothesis tests seem better in the centered model, we would think there is nothing interesting here.

Here's a puzzle for you. Consider Figure 9, which shows the predicted values and confidence intervals from the centered and uncentered regressions. Is there any substantively important difference between these two regressions?

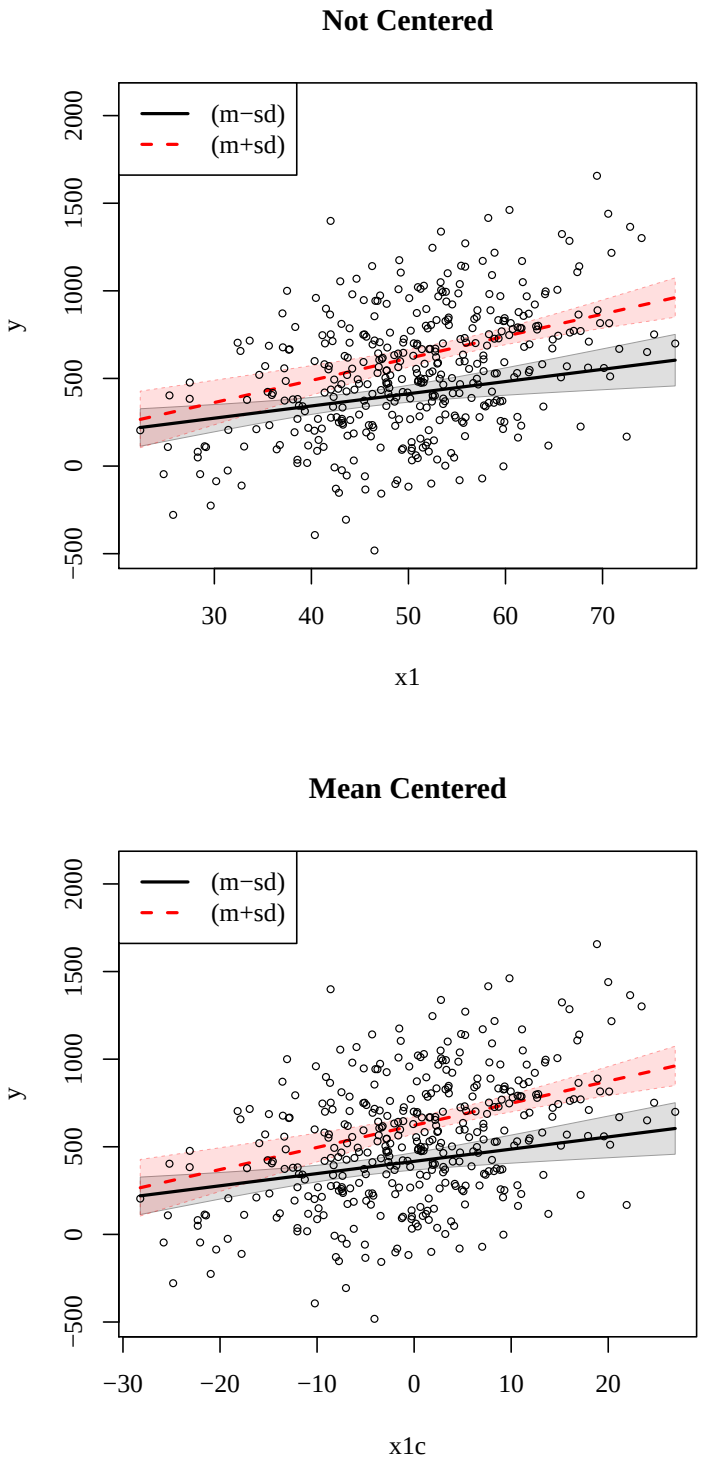

Figure 9: plotSlopes for the centered and non-centered regressions

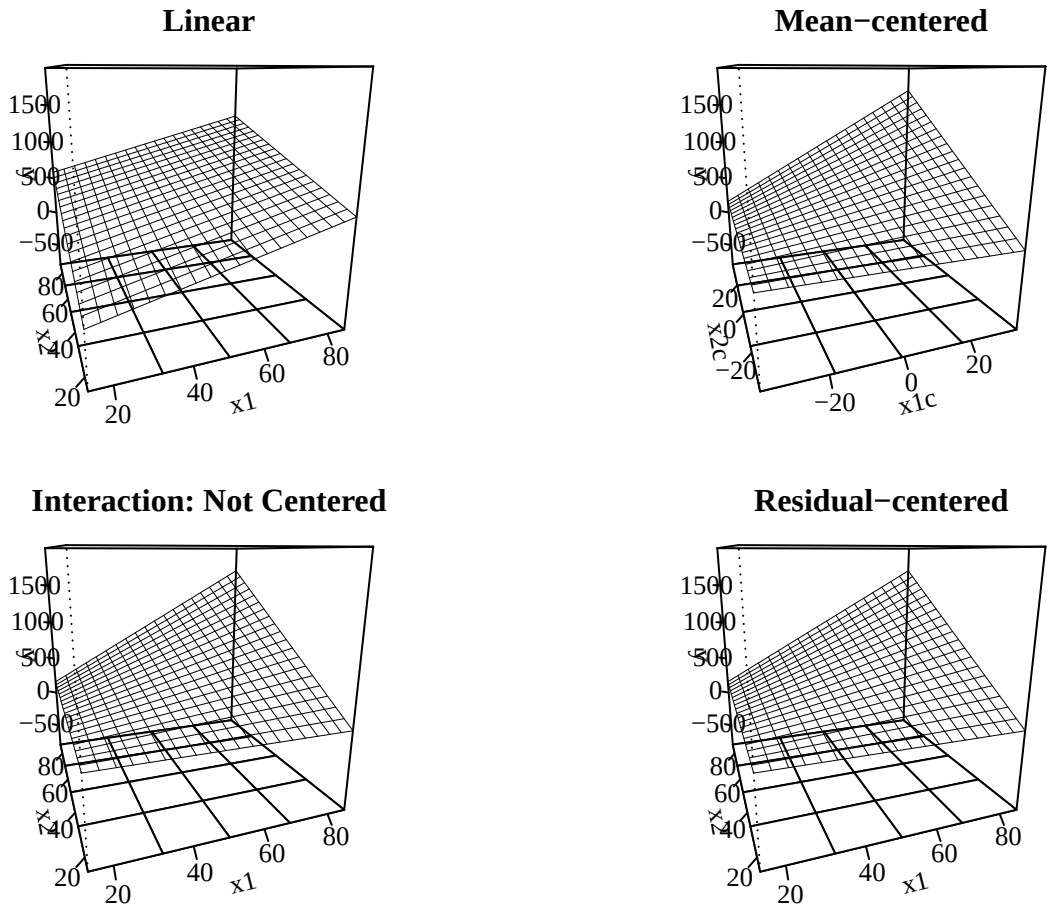

Figure 10: Predicted Planes from Centered and Uncentered Fits Identical

Perhaps the 2-D plots are not persuasive. Don't stop yet. In the 3-D plots will help quite a bit. We have not yet grasped the most fundamental changed caused by the insertion of the interaction term. When we insert $x1_i \times x2_i$, we change the fudamental nature of the regression surface. The surface of the fitted model is no longer a "flat" plane, but rather it is a curving surface.

I've assembled 3-D plots in Figure 10. We see that the ordinary interaction, mean-centered, and residual-centered models produce identical predicted values! The only difference in the figures is that the axes of the predictors have been re-scaled, so that the y axis is (implicitly) re-positioned. Now that we understand the situation, we could play around with the data and move the axis back and forth until we arrive at a position that is most favorable to our interpretation.

The regression coefficient estimates are snapshots, each summarizing the curvature at one particular point in a curving surface. It seems quite apparent in Figure 10 that the models are identical, and yet we receive different regression reports from different spots. The non-centered data offers us the slope estimate from the "front-left" part of the graph. Mean-centered estimates report on the slope in the middle of the graph. In the rockchalk examples folder, one can find a file called "centeredRegression.R" that walks through this argument step by step.

What about residual-centering? Because the transformation that it employs is more abstract, I initially thought it was actually a different model. And yet it is not. The residual-centered model is completely equivalent to the ordinary interaction model and the mean-centered model. For a given combination of the input values, the predicted values and confidence intervals are

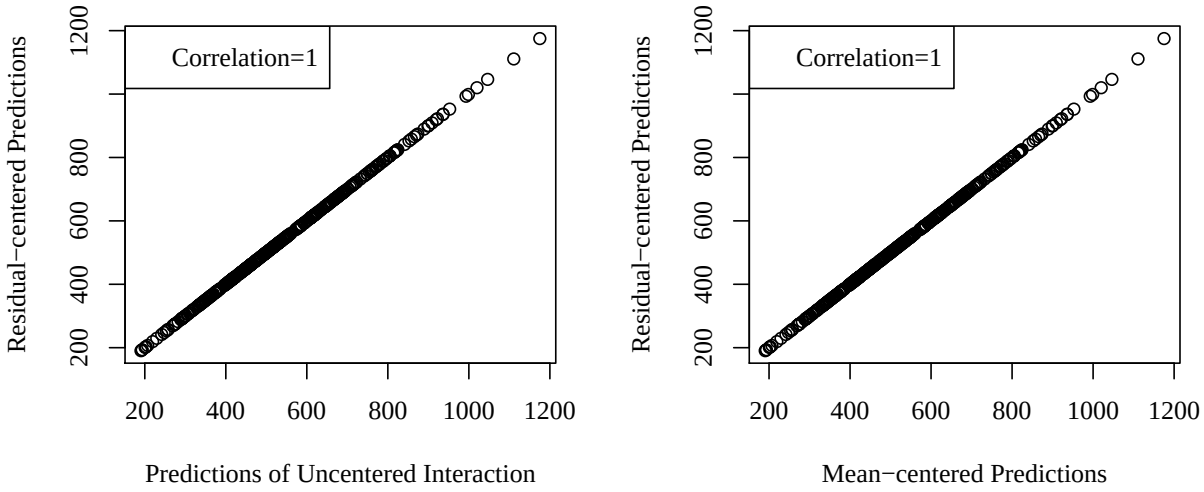

Figure 11: Predicted Values of Mean and Residual-centered Models

the same. The predicted values of the ordinary interactive model, the mean-centered model, and the residual-centered models are illustrated in Figure 11.

The take-away point from this is that there is no free lunch in regression analysis. If re-scaling a variable by adding or subtracting a constant seems to change a result, one should be cautious and suspect an error.

In order to drive the point home, I'd like to show that it is possible to translate between the estimates of any one of these fitted models and the estimates of the others. The ordinary model is

$$y_i = b_0 + b_1 x1_i + b_2 x2_i + b_3 (x1_i \times x2_i) + e1_i \tag{12}$$

The mean-centered model is

$$y_i = c_0 + c_1 (x1_i - \overline{x1}) + c_2 (x2_i - \overline{x2}) + c_3 (x1_i - \overline{x1}) \cdot (x2_i - \overline{x2}) + e2_i \tag{13}$$

In order to compare with equation 12, we would re-arrange like so

$$y_i = c_0 + c_1 (x1_i) - c_1 \overline{x1} + c_2 (x2_i) - c_2 \overline{x2} + c_3 (x1_i x2_i + \overline{x1 x2} - \overline{x1} x2_i - \overline{x2} x1_i) + e2_i \tag{14}$$

$$y_i = c_0 + c_1 (x1_i) - c_1 \overline{x1} + c_2 (x2_i) - c_2 \overline{x2} + c_3 (x1_i x2_i) + c_3 \overline{x1 x2} - c_3 \overline{x1} x2_i - c_3 \overline{x2} x1_i) + e2_i \tag{15}$$

$$y_i = \{c_0 - c_1 \overline{x1} - c_2 \overline{x2} + c_3 \overline{x1 x2}\} + \{c_1 - c_3 \overline{x2}\} x1_i + \{c_2 - c_3 \overline{x1}\} x2_i + c_3 (x1_i x2_i) + e2_i \tag{16}$$

One can then compare the parameter estimates from equations 12 and 16. Both 12 and 16 include a single parameter times $(x1_i x2_i)$, leading one to expect that the estimate $\hat{b}_3$ should be equal to the estimate of $\hat{c}_3$, and they are! Less obviously, one can use the fitted coefficients from either model to deduce the fitted coefficients from the other. The following equalities describe

that relationship.

$$\begin{aligned}
\hat{b}_0 &= \hat{c}_0 - \hat{c}_1\overline{x1} - \hat{c}_2\overline{x2} + \hat{c}_3\overline{x1x2} && (17)\\
\hat{b}_1 &= \hat{c}_1 - \hat{c}_3\overline{x2} && (18)\\
\hat{b}_2 &= \hat{c}_2 - \hat{c}_3\overline{x1} && (19)\\
\hat{b}_3 &= \hat{c}_3 && (20)
\end{aligned}$$

The estimated fit of equation 13 would provide estimated coefficients $\hat{c}_j$, $j = 0, ..., 3$, which would then be used to calculate the estimates from the non-centered model.

The estimation of the residual-centered model requires two steps. The residual from this regression model

$$(x1_i \widehat{\times} x2_i) = \hat{d}_0 + \hat{d}_1 x1_i + \hat{d}_2 x2_i. \quad (21)$$

is

$$\widehat{u}_i = (x1_i \times x2_i) - (x1_i \widehat{\times} x2_i), \quad (22)$$

which is inserted into equation 12.

$$y_i = h_0 + h_1 x1_i + h_2 x2_i + h_3\{x1_i \times x2_i - x1_i \widehat{\times} x2_i\} + e3_i \quad (23)$$

Replacing $x1_i \widehat{\times} x2_i$ with $\hat{d}_0 + \hat{d}_1 x1_i + \hat{d}_2 x2_i$, 23 becomes

$$\begin{aligned}
y_i &= h_0 + h_1 x1_i + h_2 x2_i + h_3\{x1_i \times x2_i - \hat{d}_0 - \hat{d}_1 x1_i - \hat{d}_2 x2_i\} + e3_i && (24)\\
&= h_0 + h_1 x1_i + h_2 x2_i + h_3\{x1_i \times x2_i\} - h_3\hat{d}_0 - h_3\hat{d}_1 x1_i - h_3\hat{d}_2 x2_i\} + e3_i && (25)\\
& \{h_0 - h_3\hat{d}_0\} + \{h_1 - h_3\hat{d}_1\}x1_i + \{h_2 - h_3\hat{d}_2\}x2_i + h_3\{x1_i \times x2_i\} + e3_i && (26)
\end{aligned}$$

As in the previous comparison, we can translate coefficient estimates between the ordinary specification and the residual-centered model. The coefficient estimated for the product term, $\hat{h}_3$, should be equal to $\hat{b}_3$ and $\hat{c}_3$ (and it is!). If we fit the residual centered model, 23, we can re-generate the coefficients of the other models like so:

$$\begin{aligned}
\hat{b}_0 &= \hat{c}_0 - \hat{c}_1\overline{x1} - \hat{c}_2\overline{x2} + \hat{c}_3\overline{x1x2} = && h_0 - h_3\hat{d}_0 && (27)\\
\hat{b}_1 &= \hat{c}_1 - \hat{c}_3\overline{x2} = && h_1 - h_3\hat{d}_1 && (28)\\
\hat{b}_2 &= \hat{c}_2 - \hat{c}_3\overline{x1} = && h_2 - h_3\hat{d}_2 && (29)
\end{aligned}$$

From the preceding, it should be clear enough that the three models are equivalent.

7 Conclusion

The rockchalk package is offered as a system of support for teachers and students of regression analysis. It should help with the preparation of plots, summary tables, and other diagnostics.

A number of functions are currently offered in this package that have not been emphasized in this writeup. I would draw the reader to the help pages for these functions

combineLevels a recoder for factor variables

mcDiagnose a one stop shop for multicollinearity diagnostic information

getDeltaRsquare calculate the change in the R^2 that results from the omission of each variable. This is the squared semi-partial correlation coefficient.

getPartialCor calculates the partial correlation matrix of the predictors in a regression model

lazyCor and lazyCov convenient ways to create correlation and covariance matrices that are needed in many simulation exercises

mvrnorm a slightly improved version of the MASS package's multivariate normal generator.

Development on the package will continue on the basis of classroom experience.

References

Aiken, L. S. and S. G. West (1991, January). *Multiple Regression: Testing and Interpreting Interactions*. Sage Publications, Inc.

Cohen, J., P. Cohen, S. G. West, and L. S. Aiken (2002, June). *Applied Multiple Regression/-Correlation Analysis for the Behavioral Sciences*. Routledge.

Johnson, P. N. J. (1936). Tests of certain linear hypotheses and their applications to some educational problems. *Statistical Research Memoirs 1*, 57–93.

King, G. (1986). How not to lie with statistics: Avoiding common mistakes in quantitative political science. *American Journal of Political Science 30*(3), 666–687.

Kromrey, J. D., . F.-J. (1998). Mean Centering in Moderated Multiple Regression: Much Ado about Nothing. *Educational and Psychological Measurement 58*(1), 42–67.

Little, T. D., J. A. Bovaird, and K. F. Widaman (2006). On the Merits of Orthogonalizing Powered and Product Terms: Implications for Modeling Interactions Among Latent Variables. *Structural Equation Modeling 13*(4), 497–519.

Preacher, K., P. Curran, and D. Bauer (2006). Computational Tools for Probing Interactions in Multiple Linear Regression, Multilevel Modeling, and Latent Curve Analysis. *Journal of Educational and Behavioral Statistics 31*(4), 437–448.

Quinn, G.P. and Keough, M. J. (2002). *Experimental Design and Data Analysis for Biologists.* Cambridge University Press.