Regression III

Linearity II

Dave Armstrong

Diagnosing Non-Linearity

Diagnosing non-linearity in relationships between continuous predictors is a bit more tricky.

We will use an analysis of the residuals to diagnose whether the relationship between $\mathbf{X}$ and y is well-characterized by a line.

We will also need to figure out a flexible way to model the dependencies between $\mathbf{X}$ and the residuals.

- To do this, we will need to learn something about non-parametric regression

Goals for Today

- Develop local polynomial regression as a tool for modeling bivariate relationships. + Use C+R Plots to diagnose unmodeled non-linearity.
- Discuss non-linear transformations for fixing simple, monotone non-linearity.
- Discuss polynomial regression for fixing non-linear relationships that are not simple and monotone.

Parametric vs. Non-parametric

Our goal is to trace the dependence of y on x. Specifically, we usually want to get something like:

$$y_i|x_i = f(x_i) + e_i$$

We usually define $f(\cdot)$ to be "smooth".

- The linear functional form - $f(x_i) = \alpha + \beta x_i$ - is the "smoothest" of smooth function.

The above model is parametric, because we are estimating *parameters* that describe relationship between y and x.

It is possible to characterize the relationship without estimating global parameters (i.e., parameters that apply to all of the observations equally) - what we call *non-parametric* models.

Notes

Type notes here...

Global vs. Local Parametric Models

All of the models we will talk about below are *locally* parametric.

- They fit a parametric model to a relatively small subset of the data.
- The sum total of these many local parametric fits is a non-parametric fit - one that does not impose the same functional form for all of the data.

Because these models remain locally parametric, we can usually use information from the many local models to derive standard errors for the fit. (More on this later)

Notes

Type notes here...

Local Polynomial Regression

To estimate the local polynomial regression between y and x, start with the smallest unique value of x, call it x_0, and you would estimate:

$$y_i w_i = \beta_0 + \beta_1 x_i w_i + \beta_2 x_i^2 w_i + \varepsilon_i w_i$$

for the $span \times 100\%$ of the observations closest to x_0. Let's say for the sake or argument that the $\text{span} = 0.5$.

- Find the 50% of the points closes to x_0 by calculating $d_i = |x_i - x_0|$ and then taking the 50% smallest values of d_i.
- For the observations in the subsample, calculate the scaled distance such that $\tilde{d}_i = \frac{d_i}{max(d_i)}$. This makes the largest distance in the subsample equal to 1.

Notes

Type notes here...

Local Polynomial Regression II

- Calculate the weights for the subset using the tricube weight function.

$$w_i = \left(1 - \tilde{d}^3\right)^3$$

w_i for observations outside the subset will be 0.

Notes

Type notes here...

Robustness Weighting in LPR

- Fit the local regressions using weights w_i
- Calculate the residuals $\hat{\varepsilon}_i = y_i - \hat{y}_i$
- Determine the median of the absolute values of the residuals $\hat{q}_{.5}$
- Find the robustness weights (with the Bisquare weight function):

$$r_i = B\left(\frac{\hat{\varepsilon}_i}{6\hat{q}_{.5}}\right)$$

where:

$$B(u) = \begin{cases} (1-u^2)^2, & \text{if } |u| < 1; \\ 0, & \text{otherwise.} \end{cases}$$

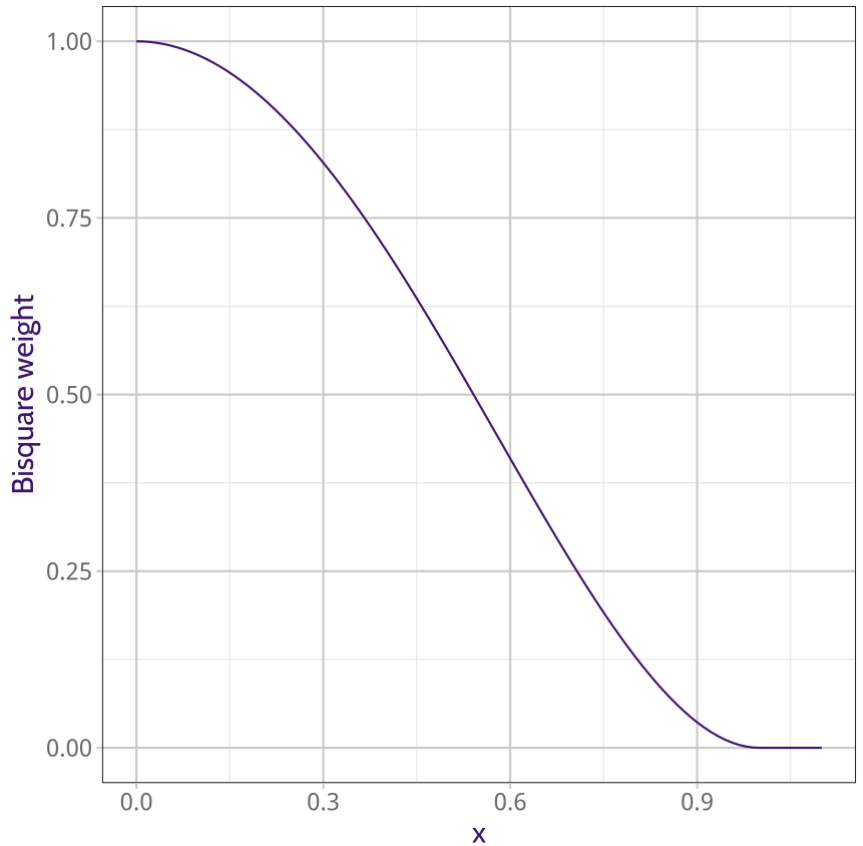
- Repeat the loess procedure using weights $r_i w_i$
- Repeat steps 2-5 until the loess model converges.

Notes

Type notes here...

Bisquare Weighting Function

What does the bisquare weight function look like?



Notes

Type notes here...

Choosing the Span

The choice of *span* (i.e., the number of points included in each local model) - this encapsulates the bias-variance tradeoff.

- A bigger span can induce bias which results in a non-parametric estimate that is not faithful to the local patterns in the data
- A smaller span can exhibit considerable variability while sticking very closely to the local pattern in the data. Overfitting is a potential problem here.

Overfitting is not necessarily a problem if we *only* care about the relationship in this sample. However, if we are (either explicitly or implicitly) trying to say something about a population with the sample, then overfitting can be a real problem.

Notes

Type notes here...

Choosing Polynomial Degree and Weight Function

Polynomial Degree:

- Higher degree polynomials are more likely to overfit the data.
- The most common advice is to set the polynomial degree to 2 and adjust the span to generate the required smoothness of fit.

Weight Function:

- The default in R is the *tricube* weight function.
- There is little reason to change this as it generally has a relatively small effect on the overall estimate.

Notes

Type notes here...

LPR in R

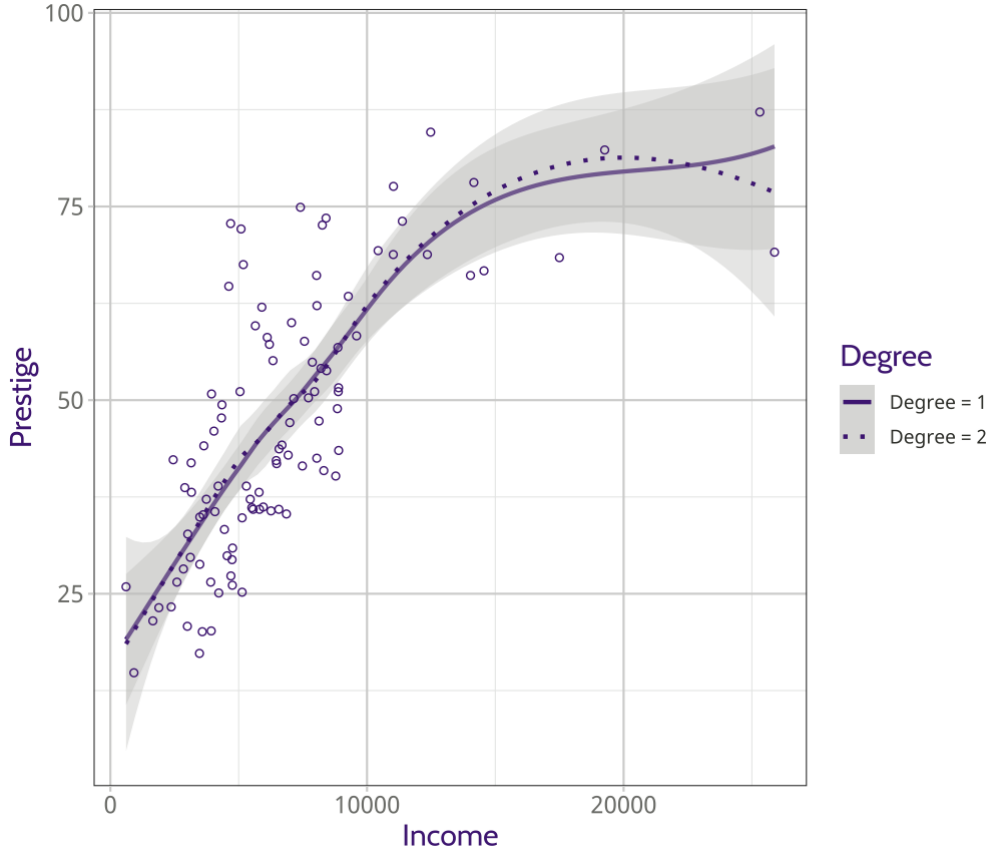
There are two different versions of this type of regression: Loess and Lowess.

- In R, The important difference between these two is that Loess can take multiple predictors (i.e., multiple nonparametric regression) whereas Lowess only takes 1. Further, the user has much more control over `loess` than `lowess`, so we spend time on the former.
  - Both `loess` and `lowess` are in the `stats` package that comes with every distribution of R.
  - The robustness weighting is done by specifying `family = symmetric` in the `loess` command. Otherwise, if `family = gaussian`, no robustness weighting (only distance weighting) will be done.

Notes

Type notes here...

Loess Graph


100
75
50
25
Prestige
0
10000
20000
Income
Degree
Degree = 1
Degree = 2

Notes

Type notes here...

Interpretation of Non-Parametric Fits

Often, we are tempted to impose some meaning on small bumps and dips in the local fit. As Keele (2007) suggests - "it is a temptation analysis should resist.''

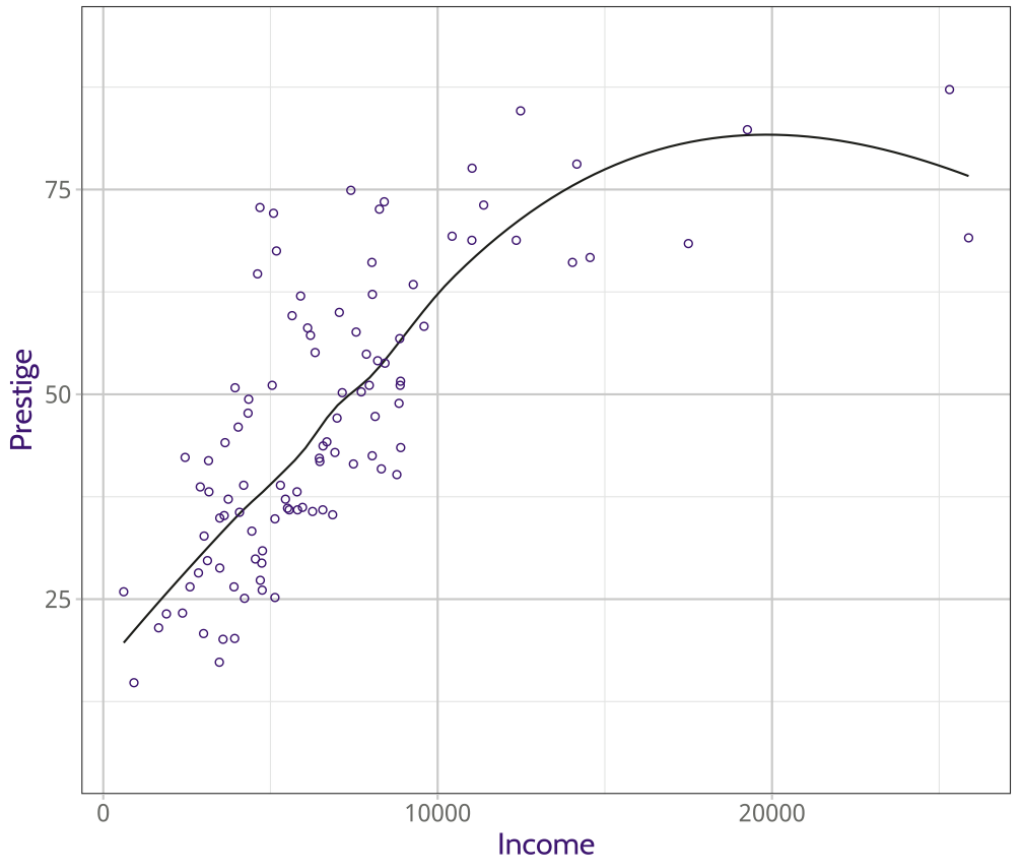
- It is often useful to consider the overall general pattern in the data and if there appears to be a pattern that can be modeled parametrically - impose that fit and assess the difference between the parametric and non-parametric models (more on this later).

Notes

Type notes here...

Plotting the LOESS curve

```
ggplot(Prestige, aes(x=income, y=prestige)) +
 stat_smooth(method="loess",
             span=.75,
             geom="line",
             method.args=list(
               family="symmetric")) +
  geom_point(pch=1) +
  theme_bw() +
  mytheme() +
  labs(x="Income",
       y="Prestige")
```



Notes

Type notes here...

Non-linearity

The assumption that the average error $E(\varepsilon)$ is everywhere zero implies that the regression surface accurately reflects the dependency of Y on the X's

- We can see this as linearity in the broad sense
- i.e., non-linearity refers to a partial relationship between two variables that is not summarized by a straight line, but it could also refer to situations when two variables specified to have additive effects actually interact.
  - Violating this assumption implies that the model fails to account for a systematic pattern between Y and the X's
  - Often models characterized by this violation will still provide a useful approximation of the pattern in the data, but they can also be misleading

It is impossible to directly view the regression surface when more than two predictors are specified, but we can employ *partial residual plots* to assess non-linearity.

Notes

Type notes here...

Partial-Residual Plots (C+R plots)

The partial residual for the j^{th} explanatory variable from a multiple regression is

$$E_i^{(j)} = E_i + B_j X_{ij}$$

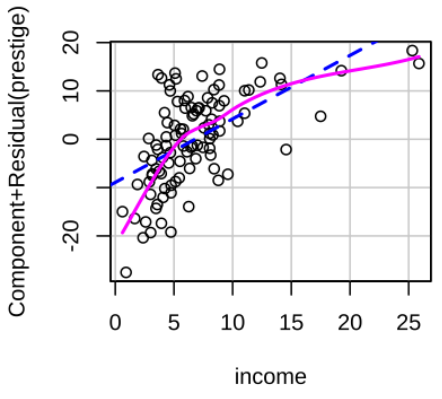
- This simply adds the linear component of the partial regression between Y and X_j (which may be characterized by a non-linear component) to the least squares residuals
- The "partial residuals" $E^{(j)}$ are plotted versus X_j, meaning that B_j is the slope of the multiple simple regression of $E^{(j)}$ on X_j
- A non-parametric smooth helps assess whether the linear trend adequately captures the partial relationship between Y and X.

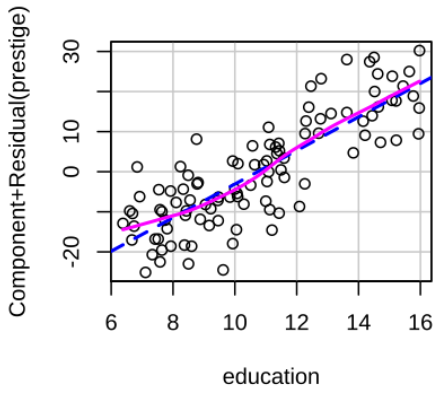
Notes

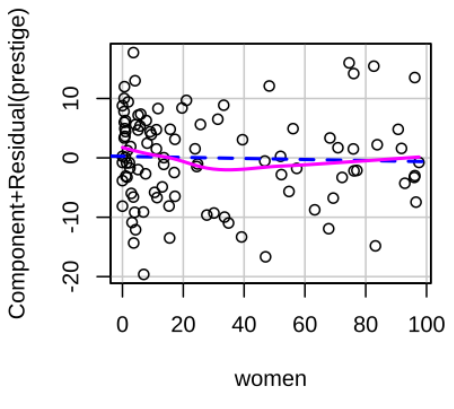
Type notes here...

Example: The Canadian Prestige Data

```
data(Prestige)
Prestige$income <- Prestige$income/1000
Prestige.model<-lm(prestige ~
  income + education +
    women, data=Prestige)
crPlot(Prestige.model, "income")
crPlot(Prestige.model, "education")
crPlot(Prestige.model, "women")
```







Notes

Type notes here...

Testing Non-linearity with CR Plots

While this is not a substitute for looking at the graphs, I have written a couple of functions that will allow you to use an F-test to evaluate significant departures from linearity.

```
library(DAMisc)
crTest(Prestige.model, adjust.method="holm")
```

```
##              RSSp   RSSnp DFnum DFdenom     F     p
## income    6033.57 5107.50 2.356  97.644 7.514 0.001
## education 6033.57 5740.21 1.235  98.765 4.088 0.075
## women     6033.57 5909.90 1.366  98.634 1.511 0.227
```

There is an option to use automatic span selection through wither AICc or GCV - both of which we'll talk about next lecture.

Notes

Type notes here...

Inference for Nonparametric Models

In the example above, we are testing the local polynomial regression against the straight line in the CR Plot. The main issue is figuring out the degrees of freedom for the LPR.

We know in OLS:

- $\hat{\mathbf{y}} = \mathbf{H}\mathbf{y}$ and $df_{\text{model}} = tr(\mathbf{H})$
- $\mathbf{H}$ is symmetric and idempotent so $tr(\mathbf{H}) = tr(\mathbf{H}\mathbf{H}')$
- Residual variance is $\frac{\mathbf{e}'\mathbf{e}}{tr[(\mathbf{I}-\mathbf{H})'(\mathbf{I}-\mathbf{H})]}$ where the denominator is the residual degrees of freedom.

Notes

Type notes here...

Degrees of Freedom II

In LPR, $\mathbf{y} = \mathbf{S}\mathbf{y}$, we have three different degrees of freedom estimates based on the OLS properties from above:

- $tr(\mathbf{S})$ (df model)
- $tr(\mathbf{S}\mathbf{S}')$ (df model)
- $tr[(\mathbf{I} - \mathbf{S})'(\mathbf{I} - \mathbf{S})] = n - tr(2\mathbf{S} + \mathbf{S}\mathbf{S}')$ (df residual), so $tr(2\mathbf{S} - \mathbf{S}\mathbf{S}')$ would be the model df.

Each provides a potentially different number with none being particularly preferred over the other.

Notes

Type notes here...

F-Tests and Nonparametric Models

We can perform an incremental F-tests for a local polynomial model versus a linear model with:

$$F = \frac{\frac{RSS_{linear} - RSS_{loess}}{tr(S) - 2}}{\frac{RSS_{loess}}{n - tr(S)}}$$

- This statistic follows and F distribution with $tr(S) - 2$ numerator and $n - tr(S)$ denominator degrees of freedom.

Notes

Type notes here...

Example

```
linmod <- lm(prestige ~ income, data=Prestige)
lomod <- loess(prestige ~ income, data=Prestige, span = .5)
testLoess(linmod, lomod)
```

```
## F = 3.6
## Pr( > F) = 0.002
## LOESS preferred to alternative
```

Notes

Type notes here...

Two Dimensions of Nonlinearity

Simple vs. Complex

- Simple means the curvature of the function relating x to y does not change direction (i.e., there is no inflection point).
- Complex means that there is an inflection point.

Monotone vs Non-monotone

- Monotone means that as x increases the function relating x to y never decreases or x increases the function relating x to y never increases, depending on the nature of the function.

Notes

Type notes here...

Handling Non-linearity: Common Strategies

Simple, monotone

- Transformations of Y and/or X

Complicated Non-linearity

- Polynomial Regression
- If pattern has too many turns, polynomials tend to oversmooth peaks
- Regression Splines
- More complicated non-parametric models.

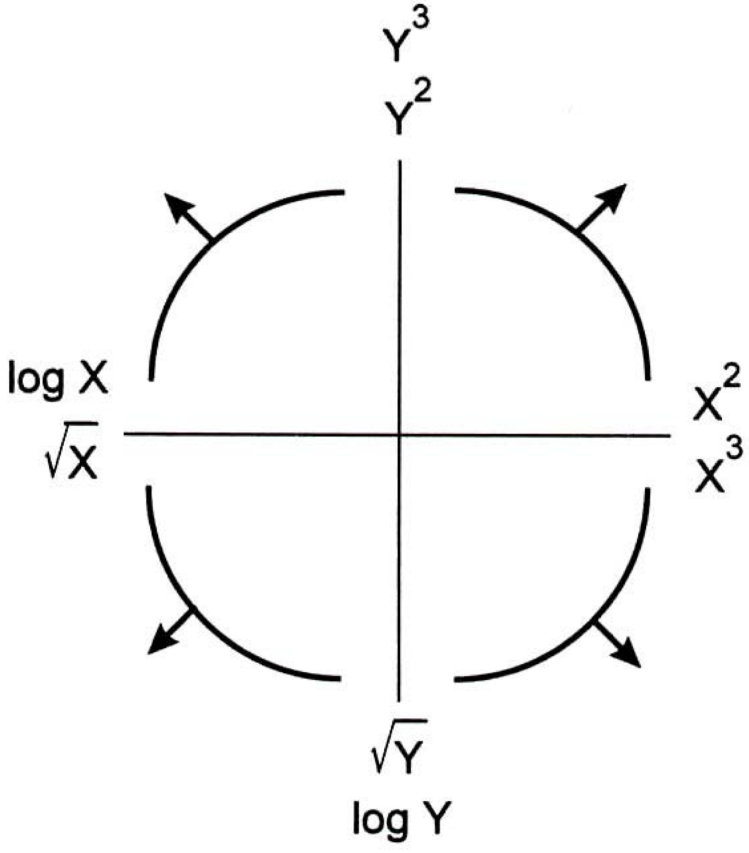
Notes

Type notes here...

Transformable Non-linearity: Bulging rule

- The direction of the bulge indicates the appropriate type of power transformation for Y and/or X
- A bulge to the top left of the scatterplot suggests transforming Y up the ladder of powers and/or X down the ladder of powers will straighten the relationship

Figure 4.6 from Fox (1997)



Notes

Type notes here...

Maximum Likelihood Transformation Methods

Although the *ad hoc* methods for assessing non-linearity are usually effective, there are more sophisticated techniques based on maximum likelihood estimation

- These techniques embed the usual multiple-regression model in a more general non-linear model that contains (a) parameter(s) for the transformation(s)
- The transformation parameter λ is estimated simultaneously with the usual regression parameters by maximizing the likelihood and this obtaining MLEs: $\mathcal{L}(\lambda, \alpha, \beta_1, \ldots, \beta_k, \sigma^2_\varepsilon)$
- If $\lambda = \lambda_0$ (i.e., there is no transformation), a likelihood ratio test, Wald test, or score test of $H_0 : \lambda = \lambda_0$ can assess whether the transformation is required
- If several variables need to be transformed, several such parameters need to be included

Notes

Type notes here...

Box-Tidwell Transformation of the X's (1)

- Maximum likelihood can also be used to find an appropriate linearizing transformation for the X variables
- The Box-Tidwell model is a non-linear model that estimates transformation parameters for the X's simultaneously with the regular parameters

$$Y_i = \alpha + \beta_1 X_{i1}^{\gamma_1} + \cdots + \beta_k X_{ik}^{\gamma_k} + \varepsilon_i$$

where the errors are iid: $\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2 \mathbf{I}_n)$ and the X_{ij} are positive

- Explicit in this model is a power transformation of each of the X's
- Of course, we would not want to transform dummy variables and the like, so we should not attempt to estimate transformation parameters for them

Notes

Type notes here...

Box-Tidwell Transformation of the X's (2)

The Box and Tidwell procedure yields a constructed variable diagnostic in the following way:

- Regress Y on the X's and obtain $A, B_1, \ldots, B_k$.
- Regress Y on the X's and the constructed variables $X_1 \log_e X_1, \ldots, X_k \log_e X_k$ to obtain $A', B'_1, \ldots, B'_k, D_1, \ldots, D_k$
- The constructed variables are used to assess the need for a transformation of X_j by testing the null hypothesis $H_0 : \delta_j = 0$ where $D_j = \hat{\delta}_j$
- A preliminary estimate of the transformation parameter γ_j is given by

$$\tilde{\gamma}_j = 1 + \frac{D_j}{B_j}$$

where B_j is the coefficient on X_j from the original equation in step 1

- Steps 1,2, and 4 are iterated until the transformation parameters converge

Notes

Type notes here...

Box-Tidwell transformation Example: Prestige Data

```
data(Prestige)
boxTidwell(prestige ~ income ,
           ~ education + women, data=Prestige)
```

```
##  MLE of lambda Score Statistic (z)  Pr(>|z|)
##       0.08073             -4.8338 1.339e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## iterations =  10
```

- The statistically significant score test indicates that a transformation is needed for income
- The MLE of Power suggests that income should be transformed by a power of -0.037 (suggesting the log might work well)

Notes

Type notes here...

Testing the Transformations

If you wanted to test whether the transformations were "close enough", you could just re-run the Box-Tidwell function on the new model with the transformed variables.

- If the transformations you provided (e.g., the log instead of -0.03) were good enough, then the transformation powers on the new data should be insignificant.

```
boxTidwell(prestige ~ log(income),
           ~ education + women, data=Prestige)
```

```
##  MLE of lambda Score Statistic (z) Pr(>|z|)
##           1.76              0.6638   0.5068
##
## iterations =  5
```

Notice that the p-value is > 0.05

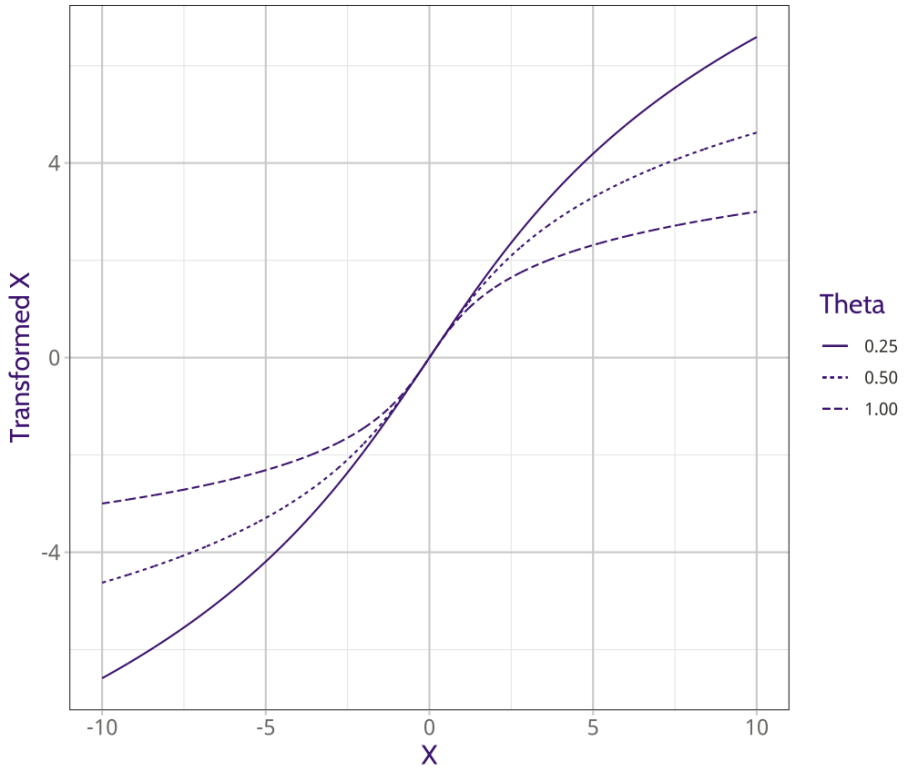
Notes

Type notes here...

Inverse Hyperbolic Sine Transformation

Sometimes, the log transform is not the most useful because a variable has lots of zeros and you don't want to add a constant to all counts. The IHS transformation is a good alternative.

$$\begin{aligned} \text{IHS}(x) &= \frac{\sinh^{-1}(\theta x)}{\theta} \\ &= \frac{log\left(\theta x + log(\theta x^2 + 1)^{\left(\frac{1}{2}\right)}\right)}{\theta} \end{aligned}$$



Notes

Type notes here...

Using the IHS Transform

```
IHS <- function(x,theta = 1){asinh(theta*x)/theta}
trans.mod2 <- lm(prestige ~ IHS(income) + education+
women, data=Prestige)
summary(trans.mod2)
```

```
##
## Call:
## lm(formula = prestige ~ IHS(income) + education + women, data = Prestige)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -17.364  -4.429  -0.101   4.316  19.179
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -120.2805    16.1459  -7.450 3.72e-11 ***
## IHS(income)   13.4382     1.9138   7.022 2.90e-10 ***
## education      3.7305     0.3544  10.527  < 2e-16 ***
## women          0.0469     0.0299   1.568     0.12
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.093 on 98 degrees of freedom
## Multiple R-squared:  0.8351,    Adjusted R-squared:   0.83
## F-statistic: 165.4 on 3 and 98 DF,  p-value: < 2.2e-16
```

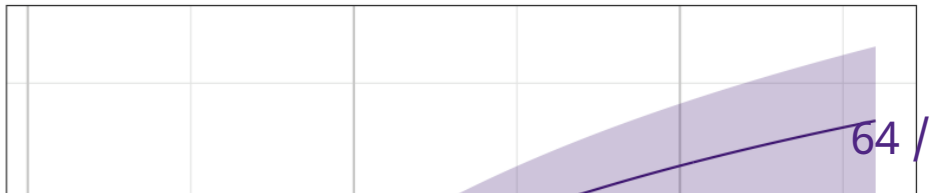
The IHS transform will also work with the `effects` and `ggeffects` packages.

Notes

Type notes here...

ggeffect plot

```
library(ggeffects)
ggpredict(trans.mod2, terms="income") %>%
  ggplot(aes(x=x, y=predicted)) +
  geom_ribbon(aes(ymin=conf.low,
                  ymax=conf.high),
              alpha=.25, fill="#4F2683") +
  geom_line(col="#4F2683") +
  theme_bw() +
  mytheme() +
  labs(x="Income", y="Predicted Prestige")
```



Notes

Type notes here...

Yeo-Johnson Transformation

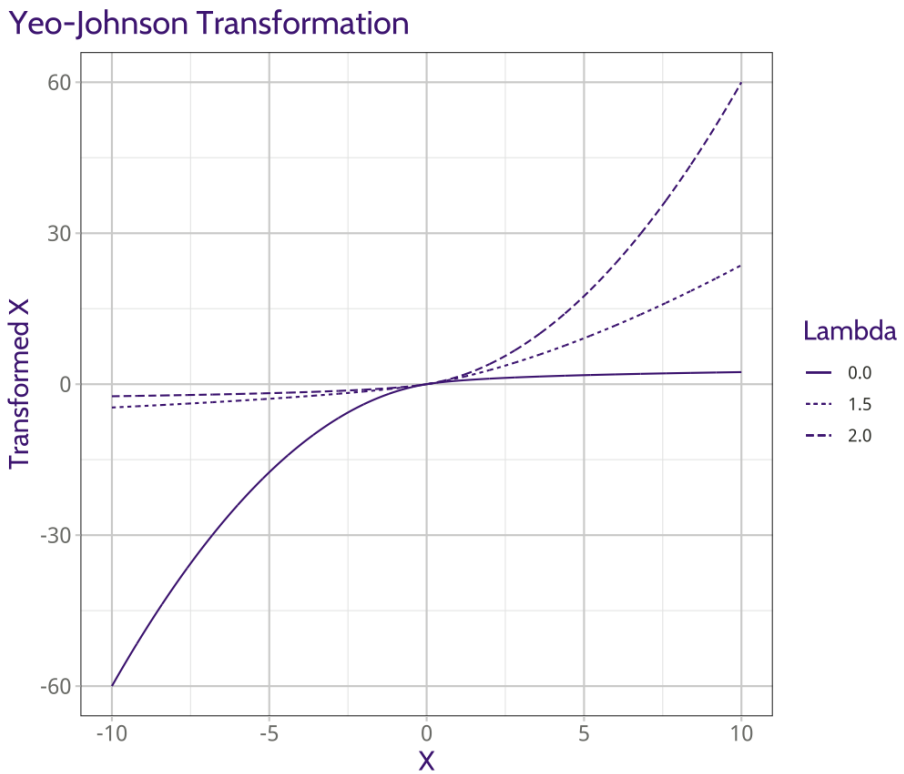
The Y-J transform is also an alternative when the variable of interest has negative or zero values. The transformation (with the parameter $\lambda = [0, 2]$) is as follows:

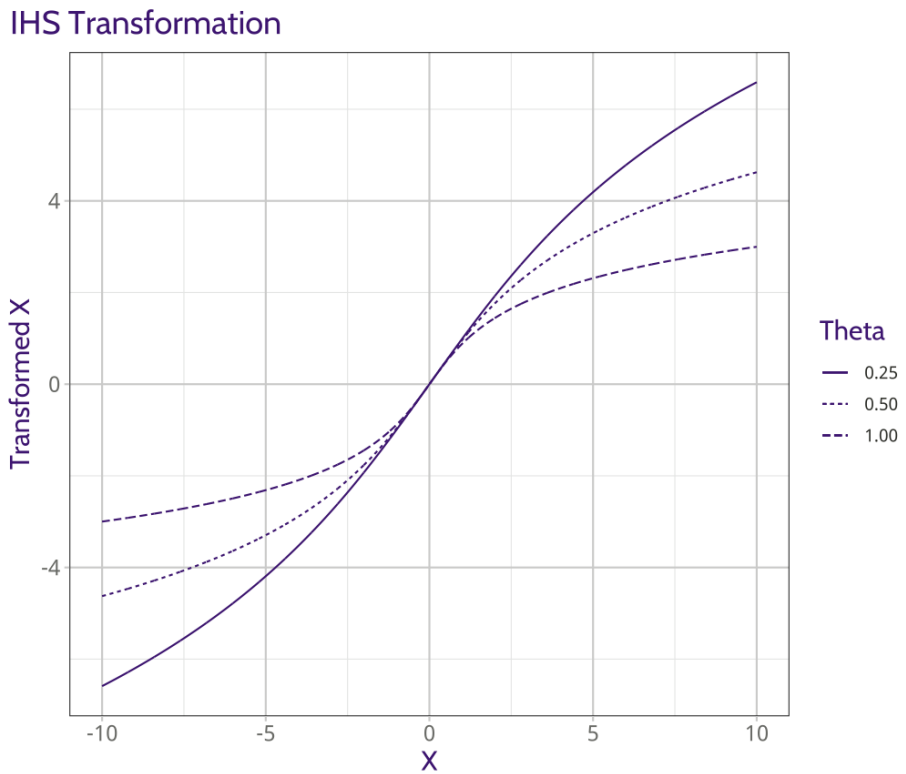
$$y_i^{(\lambda)} = \begin{cases} ((y_i + 1)^{\lambda} - 1)/\lambda & \text{if } \lambda \neq 0, y \geq 0 \\ \log(y_i + 1) & \text{if } \lambda = 0, y \geq 0 \\ -[(-y_i + 1)^{(2-\lambda)} - 1]/(2 - \lambda) & \text{if } \lambda \neq 2, y < 0 \\ -\log(-y_i + 1) & \text{if } \lambda = 2, y < 0 \end{cases}$$

Notes

Type notes here...

Y-J vs IHS


Yeo-Johnson Transformation
Transformed X
X
Lambda
0.0
1.5
2.0
60
30
0
-30
-60
-10
-5
0
5
10


IHS Transformation
Transformed X
X
Theta
0.25
0.50
1.00
4
0
-4
-10
-5
0
5
10

Notes

Type notes here...

Finding Optimal Values of λ

```
summary(yj_trans(prestige ~ income + education +
women + type, data=Prestige,
trans.vars=c("income"), round.digits=3))
```

```
##
## Call:
## lm(formula = form, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.8760  -4.0575   0.5504   4.2132  16.6404
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)            -115.69605   18.80605  -6.152 1.96e-08 ***
## yeo.johnson(income, 0)   14.65764    2.31198   6.340 8.43e-09 ***
## education                 2.97382    0.60206   4.939 3.49e-06 ***
## women                     0.08381    0.03223   2.601   0.0108 *
## typeprof                  5.29196    3.55588   1.488   0.1401
## typewc                   -3.21579    2.40657  -1.336   0.1848
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.44 on 92 degrees of freedom
##   (4 observations deleted due to missingness)
## Multiple R-squared:  0.8654,    Adjusted R-squared:  0.8581
## F-statistic: 118.3 on 5 and 92 DF,  p-value: < 2.2e-16
```

Notes

Type notes here...

Box-Cox Transformation of Y

- The Box-Cox transformation of Y functions to *normalize the error distribution, stabilize the error variance and straighten the relationship* of Y to the X's
- The general Box-Cox model is:

$$Y_i^{\lambda} = \alpha + \beta_1 X_{i1} + \cdots + \beta_k X_{ik} + \varepsilon_i$$

where $\varepsilon_i \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ and

$$Y_i^{\lambda} = \begin{cases} \frac{Y_i^{\lambda}-1}{\lambda}, & \text{for } \lambda \neq 0 \\ \log_e Y_i, & \text{for } \lambda = 0 \end{cases}$$

- If λ=1, no transformation is necessary
- Note that all of the Y_i *must* be positive

Notes

Type notes here...

Box-Cox Transformation Example: Ornstein Data (1)

```
library(car)
data(Ornstein)
optpwr <- powerTransform(I(interlocks + 1)  ~  log(assets) +
                           sector + nation, data=Ornstein)
summary(optpwr)
```

```
## bcPower Transformation to Normality
##    Est Power Rounded Pwr Wald Lwr Bnd Wald Upr Bnd
## Y1    0.2227        0.22        0.126       0.3193
##
## Likelihood ratio test that transformation parameter is equal to 0
##  (log transformation)
##                             LRT df       pval
## LR test, lambda = (0) 19.75696  1 8.7941e-06
##
## Likelihood ratio test that no transformation is needed
##                             LRT df       pval
## LR test, lambda = (1) 243.4049  1 < 2.22e-16
```

Model

You can then run the model with the optimized λ parameter from the previous slide:

```
mod <- lm(bcPower(I(interlocks + 1), optpwr$roundlam) ~
            log(assets) + sector + nation, data=Ornstein)
S(mod, brief=TRUE)
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.17047    0.58490  -3.711 0.000258 ***
## log(assets)  0.73699    0.08089   9.112  < 2e-16 ***
## sectorBNK   -0.26138    0.61547  -0.425 0.671461
## sectorCON   -0.59772    0.64389  -0.928 0.354213
## sectorFIN    0.05439    0.40505   0.134 0.893293
## sectorHLD   -0.45594    0.54763  -0.833 0.405935
## sectorMAN   -0.02196    0.28093  -0.078 0.937775
## sectorMER    0.12710    0.36024   0.353 0.724542
## sectorMIN    0.38246    0.29021   1.318 0.188843
## sectorTRN    0.43596    0.39201   1.112 0.267232
## sectorWOD    0.63923    0.36788   1.738 0.083600 .
## nationOTH   -0.31476    0.36648  -0.859 0.391292
## nationUK    -0.51269    0.36460  -1.406 0.161007
## nationUS    -1.17941    0.20387  -5.785 2.31e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard deviation: 1.337 on 234 degrees of freedom
## Multiple R-squared: 0.5051
## F-statistic: 18.37 on 13 and 234 DF,  p-value: < 2.2e-16
##    AIC    BIC
## 863.49 916.20
```

Notes

Type notes here...

Other Transformations

We talked about the IHS and Y-J transforms above, both of which permit negative values. Hawkins and Weisberg (2017) developed a new version of the Box-Cox transform that allows negative values. This distribution has two parameters λ and γ.

$$BCN(x_i) = \begin{cases} \log(.5(x_i + s_i)) & \text{if } \lambda \approx 0 \\ \frac{(0.5(x_i + s_i))^\lambda}{\lambda} & \text{if } |\lambda| > 0 \end{cases}$$

where $s_i = \sqrt{x_i^2 + \gamma^2}$

Notes

Type notes here...

Example of Other Power Transforms

```
p1 <- powerTransform(I(interlocks + 1) ~ log(assets) +
                       nation + sector, data=Ornstein)
p2 <- powerTransform(interlocks ~ log(assets) +
                       nation + sector, data=Ornstein, family="yj
p3 <- powerTransform(interlocks ~ log(assets) +
                       nation + sector, data=Ornstein, family="bc
m1 <- lm(bcPower(I(interlocks +1), p1$roundlam) ~
           log(assets) + nation + sector, data=Ornstein)
m2 <- update(m1, yjPower(interlocks, p2$roundlam) ~ .)
m3 <- update(m1, bcnPower(interlocks,
                          p3$roundlam, gamma=p3$gamma) ~ .)

cis <- bind_rows(
  as.data.frame(confint(m1)),
  as.data.frame(confint(m2)),
  as.data.frame(confint(m3)))
```

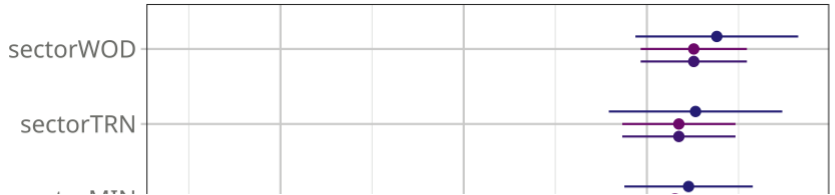
```
df <- tibble(
  b = c(coef(m1), coef(m2), coef(m3)),
  lower = cis[,1],
  upper = cis[,2],
  model = factor(rep(1:3, each=length(coef(m1))),
                 labels=c("BC", "YJ", "BCN")),
  var = factor(rep(1:length(coef(m1)), 3),
               labels=names(coef(m1)))
)
```

```
ggplot(df, aes(y=var, x=b, colour=model)) +
  geom_point(size=2, position = position_dodge(.5)) +
  geom_errorbarh(aes(xmin=lower, xmax=upper),
                 position = position_dodge(.5),
                 height=0) +
  scale_colour_manual(values = pal3) +
  theme_bw() +
  mytheme() +
  labs(x="Coefficient (95% CI)", y="")
```

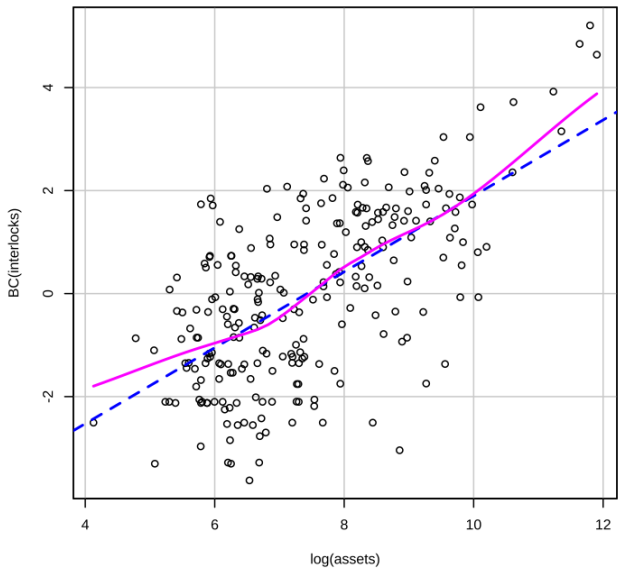
Notes

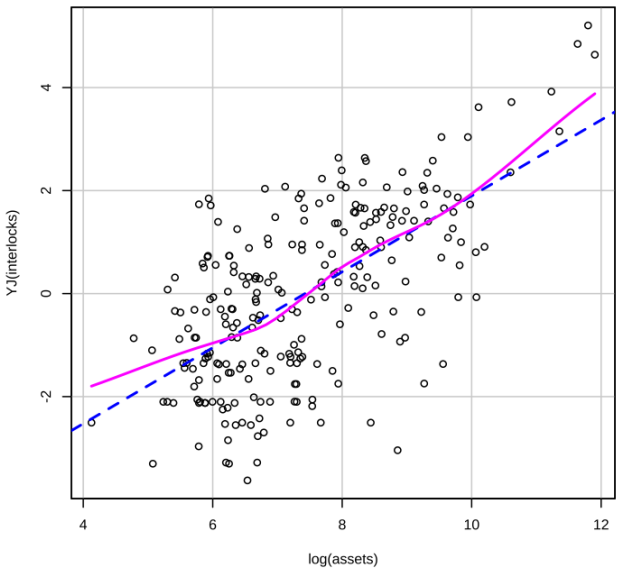
Type notes here...

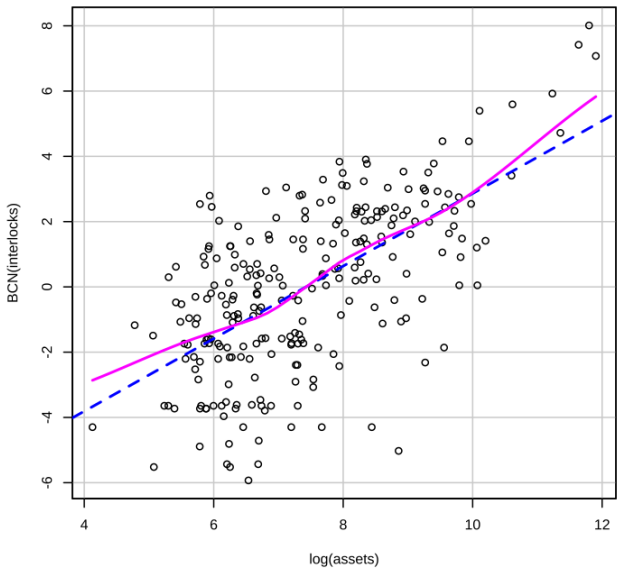
Figure



CR Plots


BC(interlocks)
log(assets)
4
6
8
10
12
-2
0
2
4


YJ(interlocks)
log(assets)
4
6
8
10
12
-2
0
2
4


BCN(interlocks)
log(assets)
4
6
8
10
12
-6
-4
-2
0
2
4
6
8

Notes

Type notes here...

Polynomial Regression

Two or more regressors of ascending power (i.e., linear, quadratic and cubic terms) are used to capture the effects of a single variable

- For every bend in the curve, we add another term to the model, going up in power each time
- Polynomial models are linear in the parameters even though they are non-linear in the variables

Order	Equation
First	$Y = \alpha + \beta_1 X$
Second	$Y = \alpha + \beta_1 X + \beta_2 X^2$
Third	$Y = \alpha + \beta_1 X + \beta_2 X^2 + \beta_3 X^3$

Notes

Type notes here...

Polynomial equations: How to choose the order

It is initially useful to look at the bends in a smooth of the scatterplot or partial residual plot

- If there is only one, a second order polynomial should be tried. For each extra bulge, we go up one in order
- A good strategy is to start with one more than you think the model needs and drop the term if it is not statistically significant
- Incremental F-tests can be used to help pick the "right" order to use in the equation
  - If the term is not statistically significant, it is usually advisable to delete the term from the model - we want as few order terms as possible
- For orthogonal polynomials, t-tests can be used

If the order is too high, however, the results will not be easy to interpret (higher than third order is rarely used)

Notes

Type notes here...

Orthogonal Polynomials: Prestige

One can fit a polynomial regression by calculating the regressors individually and adding them to the regression equation - i.e., calculate and add a quadratic term X^2 and a cubic term X^3 manually.

- *Orthogonal Polynomials* can be added in a much more simple - and better - way in R, however, by specifying a `poly` function of the variable.
- Non-orthogonal polynomials can be specified with the `raw=T` argument to `poly`.
- The order of the polynomial is specified after the variable name: eg `poly(income, 3)`

Note, in R, the `poly()` function doesn't allow missing values.

Notes

Type notes here...

Orthogonalizing Regressors

It is possible to orghotonalize the power regressors before fitting the model, below is an example for a 3^{rd} degree polynomial.

- Create $(p_1, p_2, p_3) = (X, X^2, X^3)$
- Use p_1 as the value for the first-degree term.
- Regress the p_2 and p_3 on p_1 and create residuals $e_2^{(1)}$ and $e_3^{(1)}$, respectively. Use $e_2^{(1)}$ as the value for the second-degree term
- Regress $e_3^{(1)}$ on p_1 and $e_2^{(1)}$ and use the residuals from that equation (call them $e_3^{(2)}$) as the third degree term.

This is not exactly what `poly` in R does, but the idea is similar. `poly()` also does some other normalization, so results using the above method, while equivalent in model fit terms will generate different coefficient estimates.

Notes

Type notes here...

Regression Output

```
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)            -62.910     15.916  -3.953 0.000149 ***
## log(income)             12.672      1.836   6.901 5.74e-10 ***
## poly(education, 3)1    106.494      9.284  11.471  < 2e-16 ***
## poly(education, 3)2     15.045      6.977   2.156 0.033577 *
## poly(education, 3)3    -13.348      6.984  -1.911 0.058972 .
## poly(women, 2)1         11.978      9.384   1.276 0.204893
## poly(women, 2)2         18.465      6.828   2.704 0.008110 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard deviation: 6.721 on 95 degrees of freedom
## Multiple R-squared: 0.8564
## F-statistic: 94.46 on 6 and 95 DF,  p-value: < 2.2e-16
##    AIC    BIC
## 686.89 707.89
```

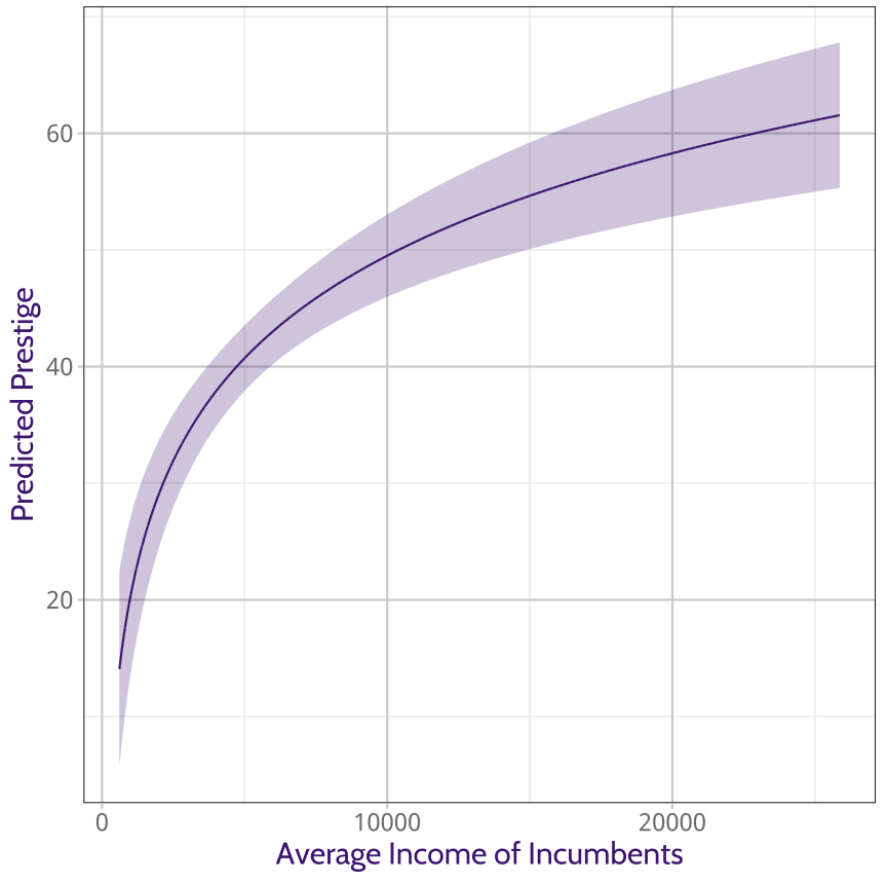
- After evaluating the 3^{rd} degree polynomial in education, it appears only two of those terms are needed. The third degree term is significant at the 0.1 level, so you might leave it in, but it would be a judgment call.
- Since the 2^{nd} degree term for women is significant we would have to leave in the first degree term as well. The inclusion of the first degree term allows the function to be non-monotonic.

Notes

Type notes here...

Effect Display for Income

```
ggpredict(mod, "income [n=25]") %>%
  ggplot(aes(x=x, y=predicted)) +
    geom_ribbon(aes(ymin=conf.low,
                    ymax=conf.high),
                alpha=.25, fill="#4F2683") +
    geom_line(col="#4F2683") +
    theme_bw() +
    mytheme() +
    labs(x="Average Income of Incumbents",
         y="Predicted Prestige")
```

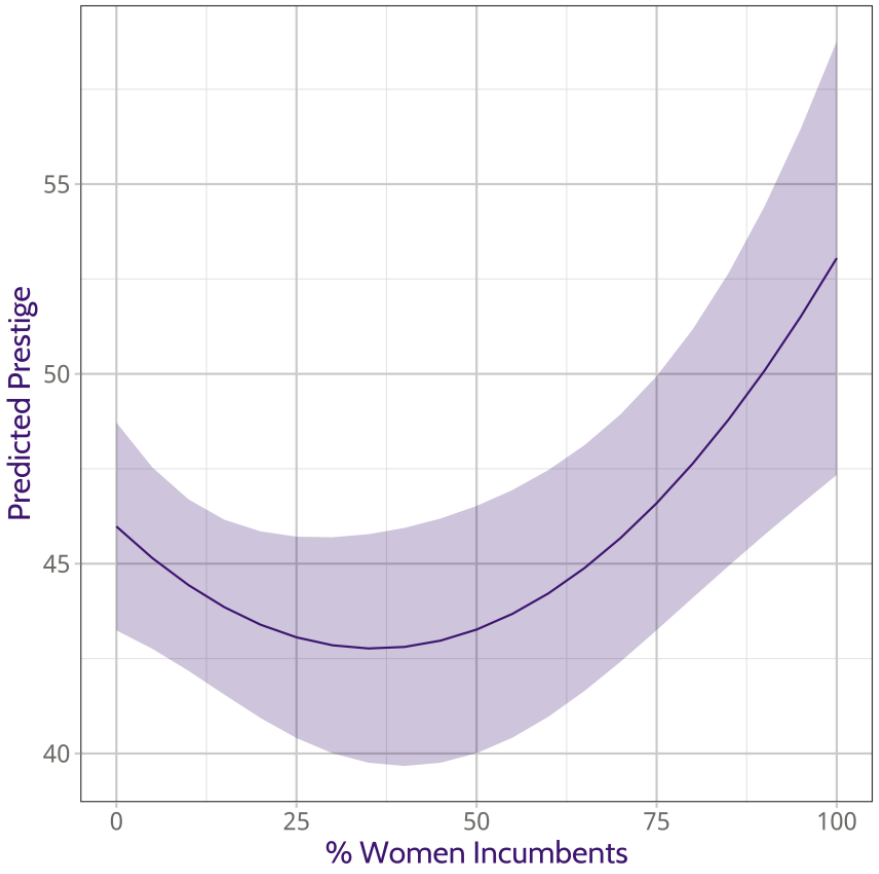


Notes

Type notes here...

Effect Display for Women

```
ggpredict(mod, "women [n=25]") %>%
  ggplot(aes(x=x, y=predicted)) +
    geom_ribbon(aes(ymin=conf.low,
                    ymax=conf.high),
                alpha=.25, fill="#4F2683") +
    geom_line(col="#4F2683") +
    theme_bw() +
    mytheme() +
    labs(x="% Women Incumbents",
         y="Predicted Prestige")
```

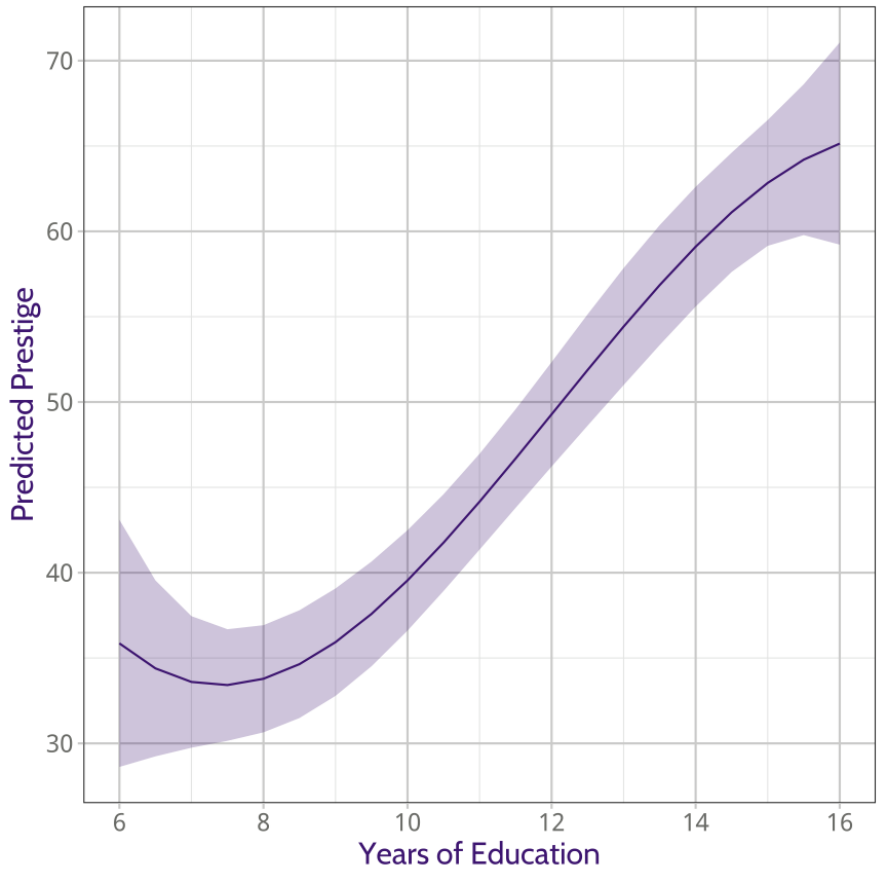


Notes

Type notes here...

Effect Display for Education

```
ggpredict(mod, "education [n=25]") %>%
  ggplot(aes(x=x, y=predicted)) +
    geom_ribbon(aes(ymin=conf.low,
                    ymax=conf.high),
                alpha=.25, fill="#4F2683") +
    geom_line(col="#4F2683") +
    theme_bw() +
    mytheme() +
    labs(x="Years of Education",
         y="Predicted Prestige")
```



Notes

Type notes here...