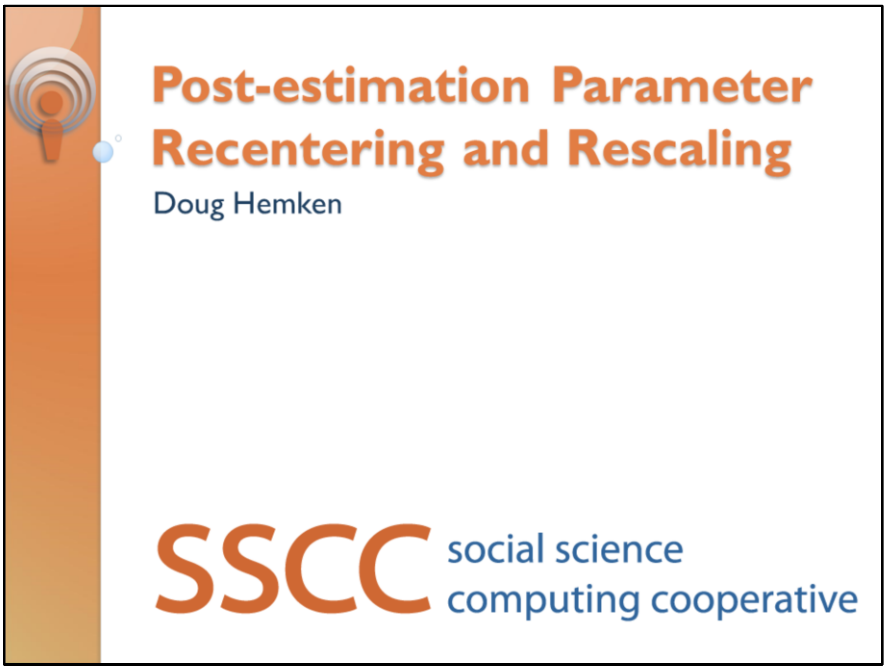
SSCC social science computing cooperative

Research Computing

University of Wisconsin – Madison



Hello.

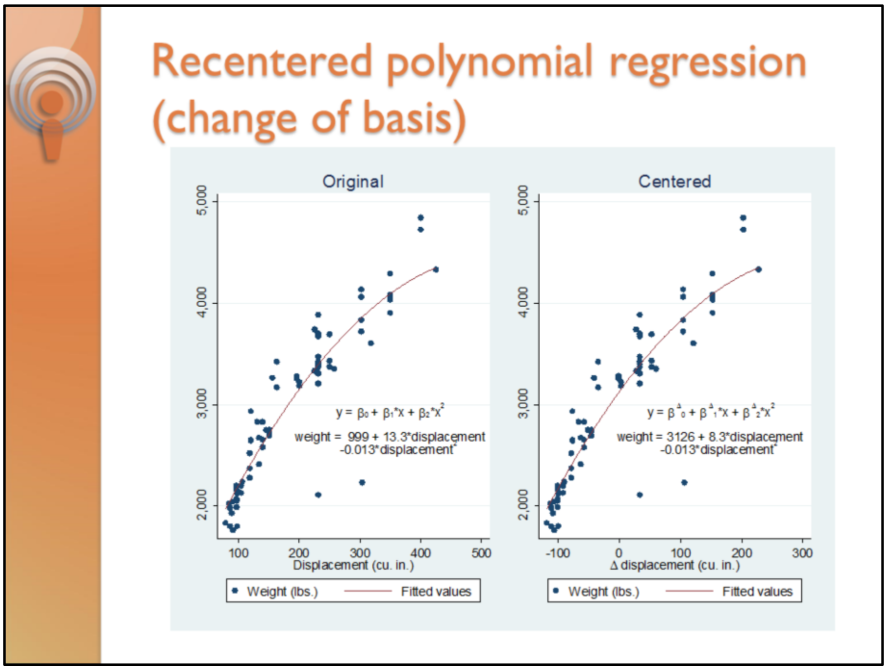
I was drawn into the problem of recentering and rescaling parameters a couple of years ago, when one of our post-docs came to me with a simulation she wanted help with.

She was writing for a Psychology journal that required all results to include standardized coefficients, so she needed to simulate both her original model and the standardized one.

She had her original model, and the descriptive statistics, but she had changed Universities and no longer had access to the data.

A main point of her model was it’s interaction term.

Her model was simple enough that it could be solved with high-school algebra, but it got me interested in the bigger question.

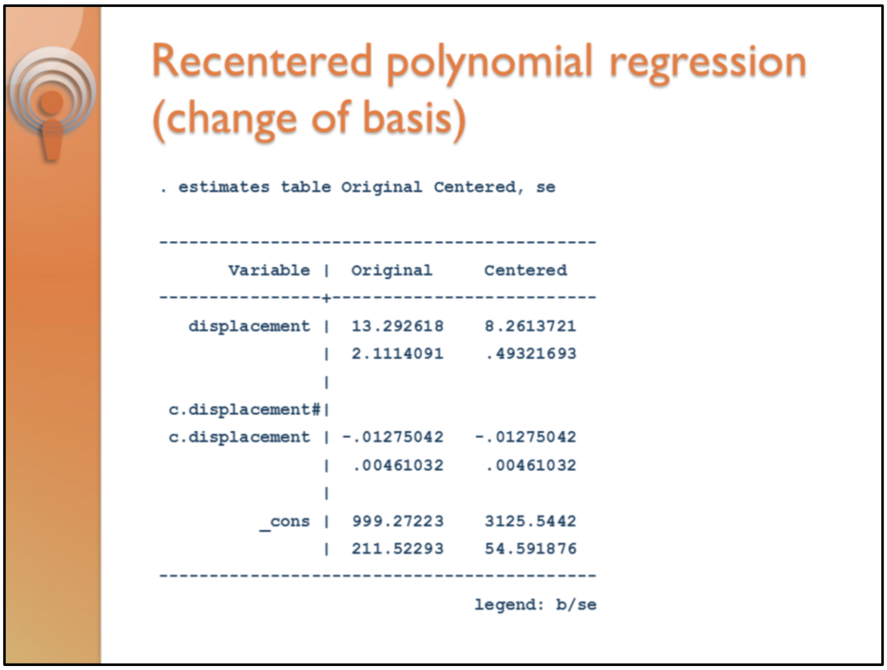


Recentering and rescaling data can be thought of as just a change of coordinates, or a change of basis.

Looking at a graph like this one, it is fairly intuitive that the relationships among the data points, and with the fitted line have not changed, but we have changed how the axis is labeled.

Here, the predicted values and the residuals are exactly the same.

A change of basis in the data induces a natural and intuitive change of basis for the parameters as well. So there is a linear transformation that takes our parameters, expressed in the original terms, and converts them to the new basis.



Recentered polynomial regression (change of basis)

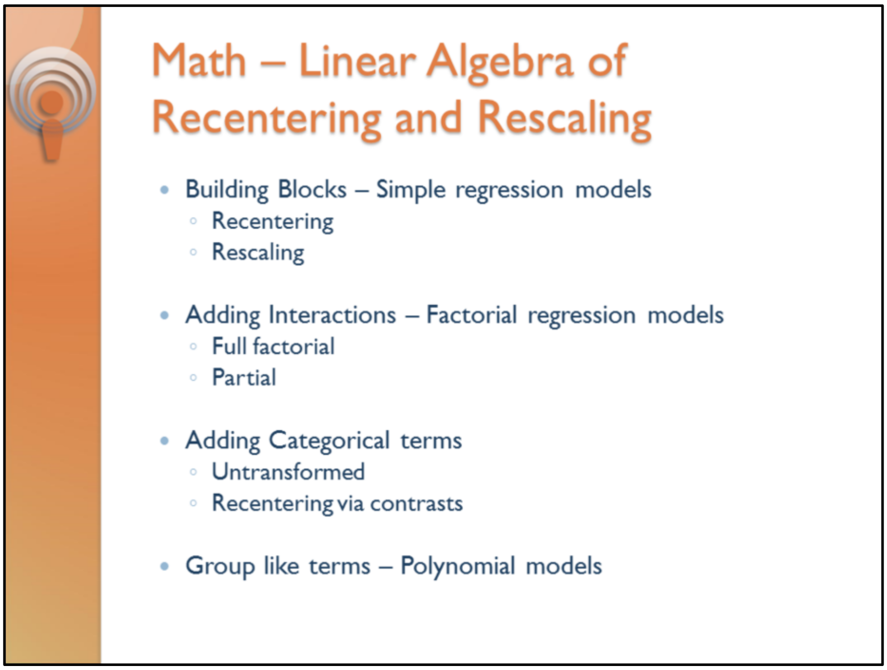
```
. estimates table Original Centered, se

--------------------------------------------
      Variable |  Original     Centered
---------------+----------------------------
  displacement |  13.292618    8.2613721
               |  2.1114091    .49321693
               |
c.displacement#|
c.displacement | -.01275042   -.01275042
               |  .00461032    .00461032
               |
         _cons |  999.27223    3125.5442
               |  211.52293    54.591876
--------------------------------------------
                                legend: b/se
```

Just looking at the numerical results, it is less intuitive that the Centered coefficients are a linear transformation of the Original ones.

But they are.

And we can write this linear transformation as a matrix.

I have written a little Stata code that generates this matrix, and uses it as a post-estimation command to center and standardize model coefficients.



I want to outline the math first, and then sketch the Stata programming.

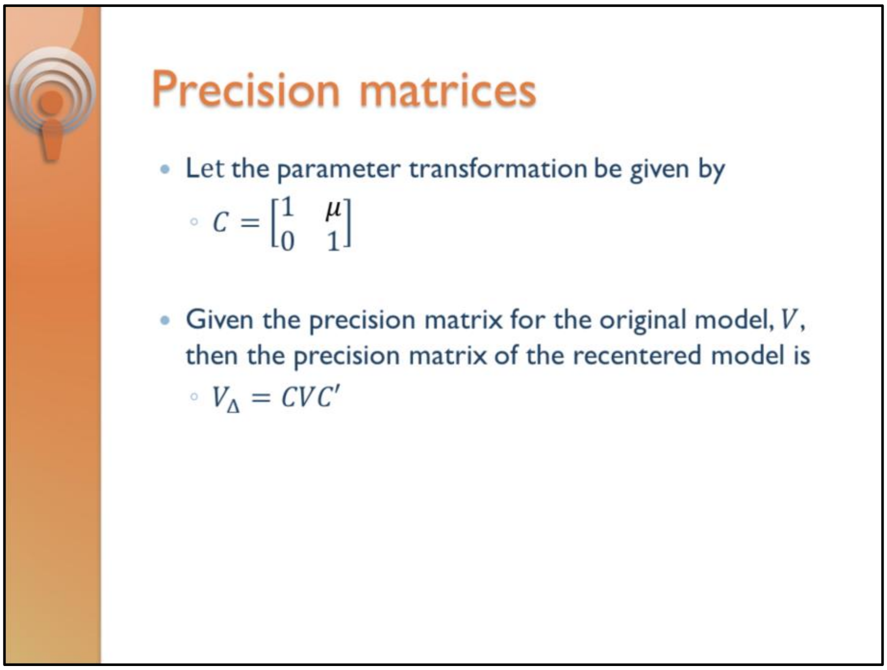
And this will be a hand-waving overview.

We'll start with ….

Simple regression recentering

- Given a model
  - $y = \beta_0 + \beta_1 x$
- And a recentering constant
  - $\Delta x = x - \boldsymbol{\mu}$
- Then the recentered model
  - $y = \beta_0^\Delta + \beta_1^\Delta \Delta x$
- Has parameters given by
  - $B^\Delta = \begin{bmatrix} 1 & \boldsymbol{\mu} \\ 0 & 1 \end{bmatrix} B$, or
  - $\begin{bmatrix} \beta_0^\Delta \\ \beta_1^\Delta \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{\mu} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$

Recentering parameters in the simplest regression model is very simple.



The change to the precision matrix is equally simple.

If we call the parameter transformation matrix C, we can use it to change the basis for the precision matrix this way.

In all the transformations that follow, this is how we deal with the precision matrix, so I won’t talk about it any more.

But this is kind of a useful matrix.

Recentering y

- Given
  - $y = \beta_0 + \beta_1 x$
  - $\Delta x = x - \mu_x$
  - $\Delta y = y - \boldsymbol{\mu_y}$
- Then
  - $\Delta y = \beta_0^{\Delta y} + \beta_1^{\Delta} \Delta x$
- Is
  - $B^{\Delta y} = \begin{bmatrix} 1 & \mu_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_0 + \boldsymbol{\mu_y} \\ \beta_1 \end{bmatrix}$

Recentering y only changes the intercept in the model, and we could write this a couple of different ways.

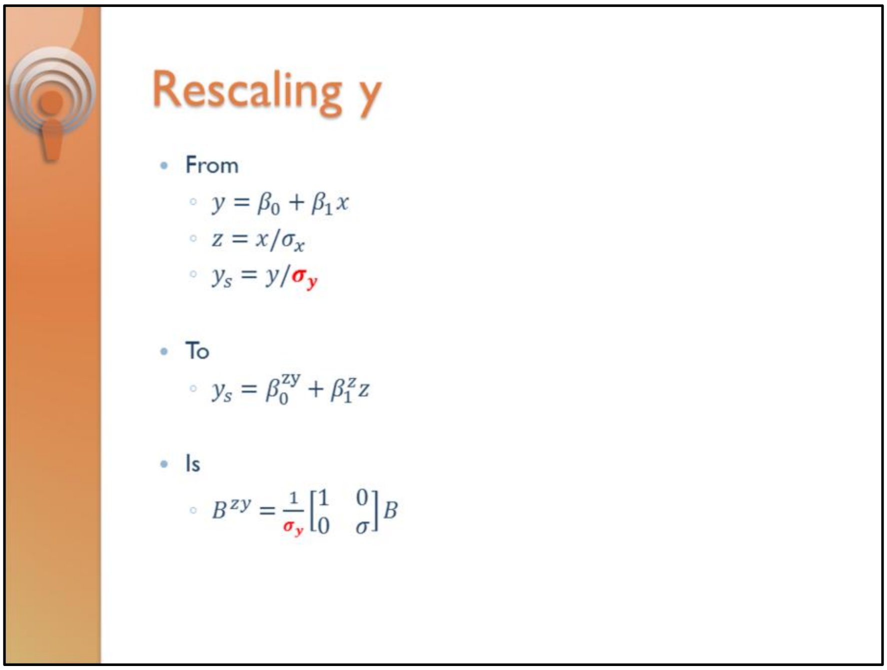
The real point is that it doesn’t change the parameter transformation matrix, and we won’t need to consider it much further.

Simple regression rescaling

- Given a model
  - $y = \beta_0 + \beta_1 x$
- And a rescaling constant
  - $z = x/\boldsymbol{\sigma}$
- Then the rescaled model
  - $y = \beta_0^z + \beta_1^z z$
- Has parameters given by
  - $B^z = \begin{bmatrix} 1 & 0 \\ 0 & \boldsymbol{\sigma} \end{bmatrix} B$

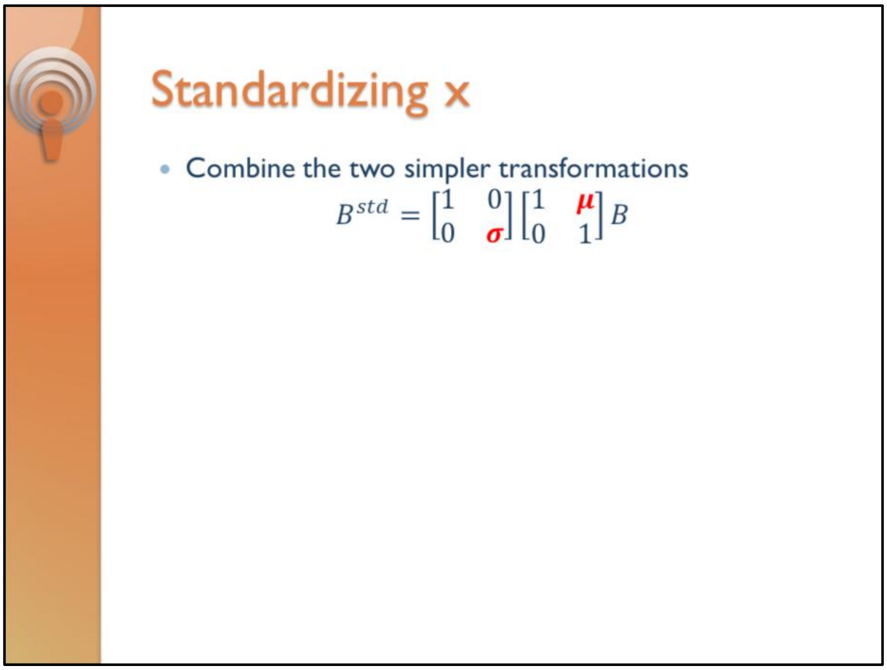
Rescaling a simple regression is as simple as recentering.

The matrix looks like this.

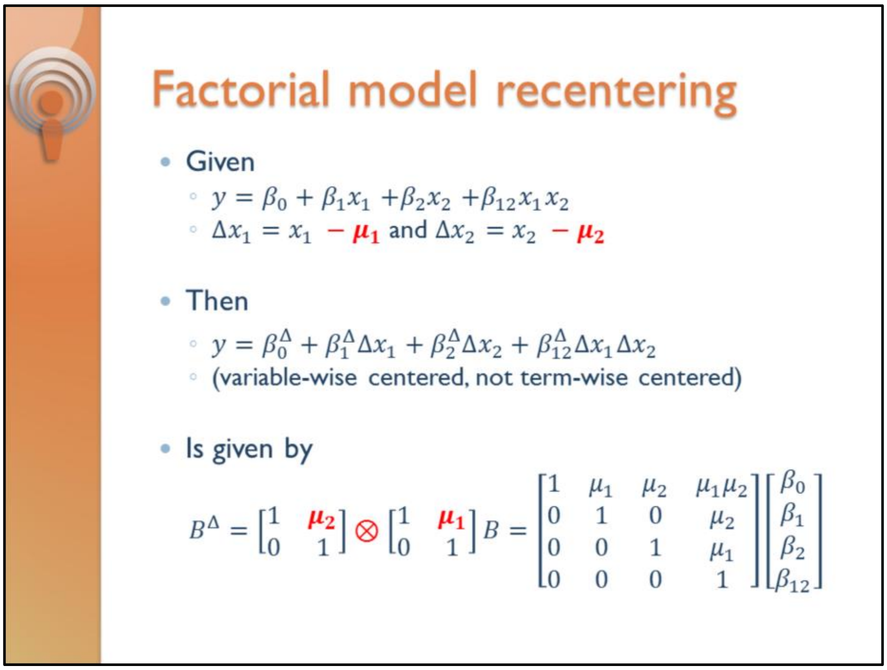


Rescaling y again has no effect within the transformation matrix, it is just a scalar transformation of the whole thing.

So we won't worry about y in what remains.



Standardizing is just centering and then rescaling.



Factorial models add interaction terms, and this is where we start to build.

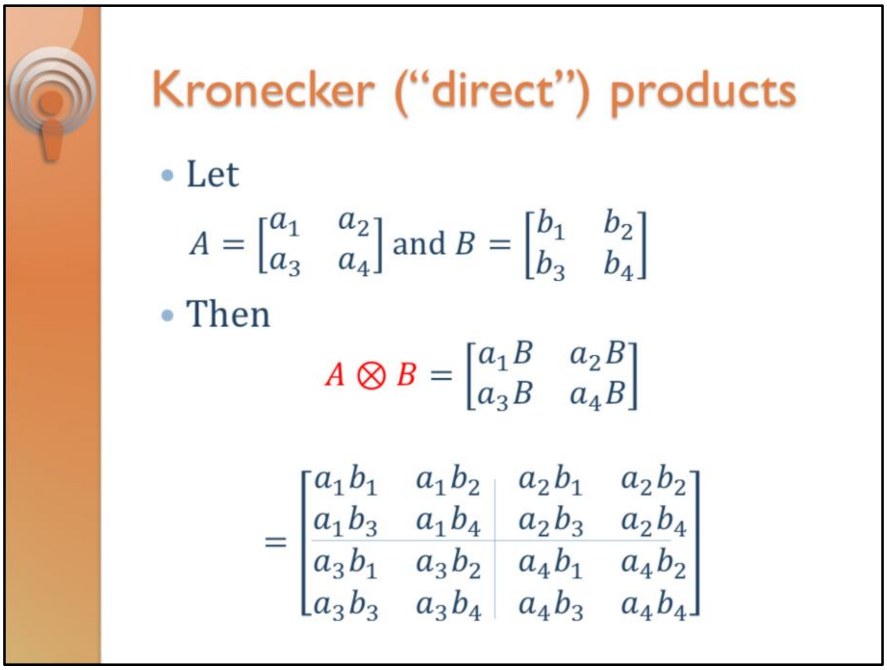
Variable-wise recentering.

We combine our building blocks with a “direct product” or “Kronekcer product”.

There is a theorem about change of basis in tensors that underlies this step.

The resulting linear transformation has a characteristic pattern.

(It also implies a particular order to our vector of parameters.)



If you don't work with Kronecker products a lot, let me remind you of how this operation works.

Each element of the first matrix is used as a scalar to multiply by the second matrix, and they are arranged as a partitioned matrix.

Factorial model rescaling

- Given
  - $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2$
  - $z_1 = x_1/\boldsymbol{\sigma_1}$ and $z_2 = x_2/\boldsymbol{\sigma_2}$

- Then
  - $y = \beta_0^z + \beta_1^z z_1 + \beta_2^z z_2 + \beta_{12}^z z_1 z_2$

- Is given by
  - $B^z = \begin{bmatrix} 1 & 0 \\ 0 & \boldsymbol{\sigma_2} \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & \boldsymbol{\sigma_1} \end{bmatrix} B$
  - $B^z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \boldsymbol{\sigma_1} & 0 & 0 \\ 0 & 0 & \boldsymbol{\sigma_2} & 0 \\ 0 & 0 & 0 & \boldsymbol{\sigma_1 \sigma_2} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_{12} \end{bmatrix}$

Rescaling transformations work the same way as recentering transformations, and we can again use both together to generate standardizing transformations.

Three-way recentering

- Given
  - $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \beta_3 x_3 + \beta_{13} x_1 x_3 + \beta_{23} x_2 x_3 + \beta_{123} x_1 x_2 x_3$
  - $\Delta x_1 = x_1 - \boldsymbol{\mu_1}, \Delta x_2 = x_2 - \boldsymbol{\mu_2}, \Delta x_3 = x_3 - \boldsymbol{\mu_3}$
- Then

$$B^{\Delta} = \begin{bmatrix} 1 & \mu_3 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & \mu_2 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & \mu_1 \\ 0 & 1 \end{bmatrix} B$$

$$= \begin{bmatrix} 1 & \mu_1 & \mu_2 & \mu_1\mu_2 & \mu_3 & \mu_1\mu_3 & \mu_2\mu_3 & \mu_1\mu_2\mu_3 \\ 0 & 1 & 0 & \mu_2 & 0 & \mu_3 & 0 & \mu_2\mu_3 \\ 0 & 0 & 1 & \mu_1 & 0 & 0 & \mu_3 & \mu_1\mu_3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \mu_3 \\ 0 & 0 & 0 & 0 & 1 & \mu_1 & \mu_2 & \mu_1\mu_2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & \mu_2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \mu_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_{12} \\ \beta_3 \\ \beta_{13} \\ \beta_{23} \\ \beta_{123} \end{bmatrix}$$

A bigger factorial design just extends the process.

(Point! 1 .. 2 ..3)

Partial Factorial

- Suppose a model has only 2nd order interaction terms
- This is $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \beta_3 x_3 + \beta_{13} x_1 x_3 + \beta_{23} x_2 x_3 + \beta_{123} x_1 x_2 x_3$ with $\boldsymbol{\beta_{123} = 0}$. In our centered model, likewise, we have $\boldsymbol{\beta_{123}^{\Delta} = 0}$
- Then we can simplify our notation:

$$\begin{bmatrix} 1 & \mu_1 & \mu_2 & \mu_1\mu_2 & \mu_3 & \mu_1\mu_3 & \mu_2\mu_3 & \mu_1\mu_2\mu_3 \\ 0 & 1 & 0 & \mu_2 & 0 & \mu_3 & 0 & \mu_2\mu_3 \\ 0 & 0 & 1 & \mu_1 & 0 & 0 & \mu_3 & \mu_1\mu_3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \mu_3 \\ 0 & 0 & 0 & 0 & 1 & \mu_1 & \mu_2 & \mu_1\mu_2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & \mu_2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \mu_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_{12} \\ \beta_3 \\ \beta_{13} \\ \beta_{23} \\ \boldsymbol{\beta_{123}} \end{bmatrix}$$

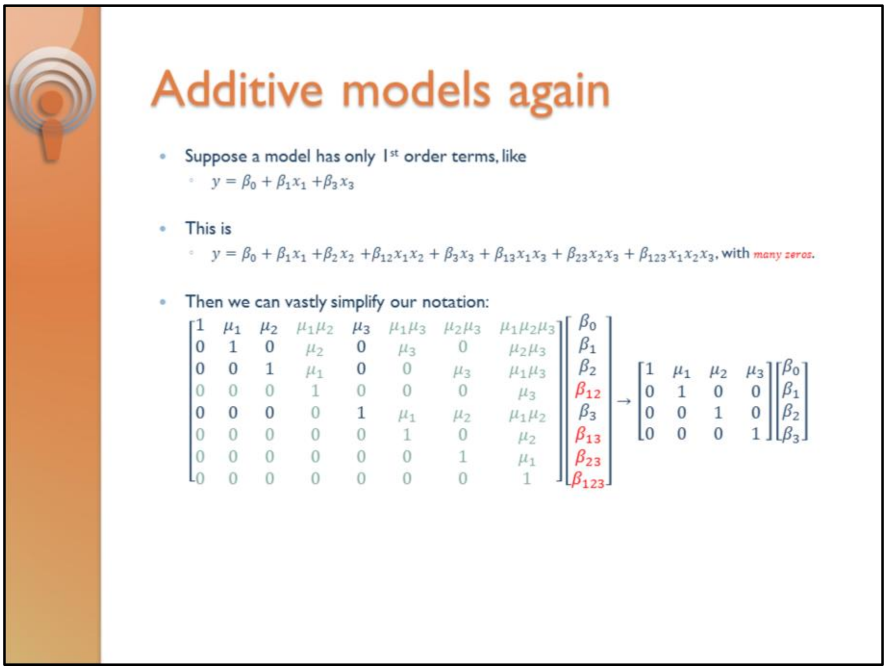
$$\rightarrow simplifies\ as \begin{bmatrix} 1 & \mu_1 & \mu_2 & \mu_1\mu_2 & \mu_3 & \mu_1\mu_3 & \mu_2\mu_3 \\ 0 & 1 & 0 & \mu_2 & 0 & \mu_3 & 0 \\ 0 & 0 & 1 & \mu_1 & 0 & 0 & \mu_3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \mu_1 & \mu_2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_{12} \\ \beta_3 \\ \beta_{13} \\ \beta_{23} \end{bmatrix}$$

Many models are less than full factorial: among other things we want to be able to consider models that are specified with an odd number of terms, and not just even numbers!

We get these transformations by whittling down the full factorial transformation.

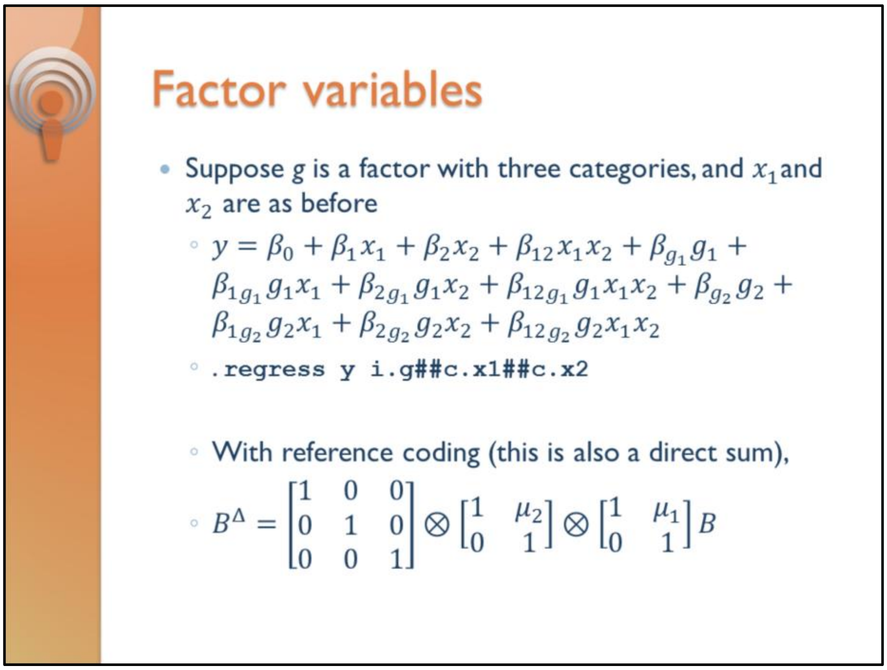
Here, if we set the three-way interaction to 0, we essentially zero out a column of our transformation matrix. We can simplify by removing both the column from the matrix and the parameter from the parameter vector.

If we do that, we are left with a row of nothing but zeros. So we can further simplify by dropping that row, and the parameter that is produces.



An additive model, then, comes out something like this.

Notice the characteristic pattern: none of the first order terms change, only the intercept, the zero-th order term, is changed.



Now let’s think about the intercept, or multiple intercepts.

If we start with reference coding, and use indicators for out categories, we may not want to recenter or rescale.

Then our transformation matrix is just expanded into block diagonal form, and our direct product is equivalent to a direct sum.

Block diagonal, or direct sum

$$\begin{bmatrix}
1 & \mu_1 & \mu_2 & \mu_1\mu_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & \mu_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & \mu_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & \mu_1 & \mu_2 & \mu_1\mu_2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & \mu_2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & \mu_1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \mu_1 & \mu_2 & \mu_1\mu_2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \mu_2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \mu_1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

So with three categories and two continuous variables, if I want to leave the model reference coded my transformation matrix looks like this.

One way to think of this is that I can create the transformation matrix for my covariates, and reuse it.

Factor Grand Mean Centering

- To transform from reference coding to grand mean centered coding, the transformation matrix depends on the number of categories:
- Two categories are centered by

$$\begin{bmatrix} 1 & 1/2 \\ 0 & -1/2 \end{bmatrix}$$

- Three categories

$$\begin{bmatrix} 1 & 1/3 & 1/3 \\ 0 & 2/3 & -1/3 \\ 0 & -1/3 & 2/3 \end{bmatrix}$$

- Four categories

$$\begin{bmatrix} 1 & 1/4 & 1/4 & 1/4 \\ 0 & 3/4 & -1/4 & -1/4 \\ 0 & -1/4 & 3/4 & -1/4 \\ 0 & -1/4 & -1/4 & 3/4 \end{bmatrix}$$

We could also consider other approaches to coding categorical variables.

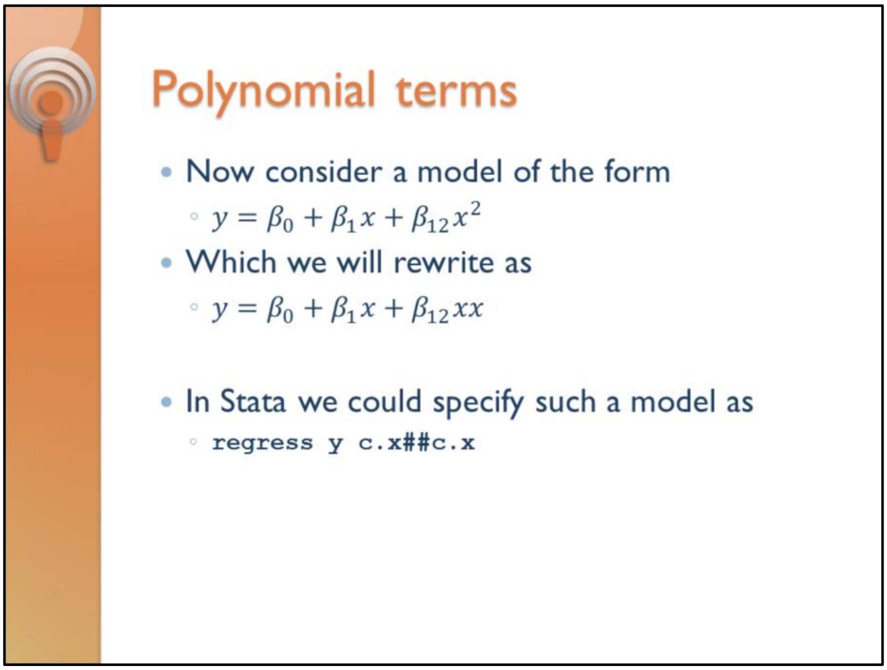
I haven't built this into my little package, so I'm just mentioning that there is nothing that requires us to stick with reference coding.

Grand Mean transformation

- For n categories:

$$\begin{bmatrix} 1 & 1/n & \cdots & 1/n \\ 0 & \frac{n-1}{n} & -1/n & -1/n \\ \vdots & -1/n & \ddots & \vdots \\ 0 & -1/n & \cdots & \frac{n-1}{n} \end{bmatrix}$$

I think it is useful to think about how this matrix relates to the “data centering” matrix you find in textbooks.



Finally we turn to polynomial models.

We can recast the polynomial terms as interactions.
(This is something you ***<u>cannot</u>*** do in R, by the way.)

Notice that we have only 1 first order term instead of the usual 2 terms that we see in other factorial models.

What we have done is collected our “x” terms.

Polynomial Terms

- Here we'll need to collect terms

  If $A = \begin{bmatrix} 1 & \mu_1 \\ 0 & 1 \end{bmatrix}$ then

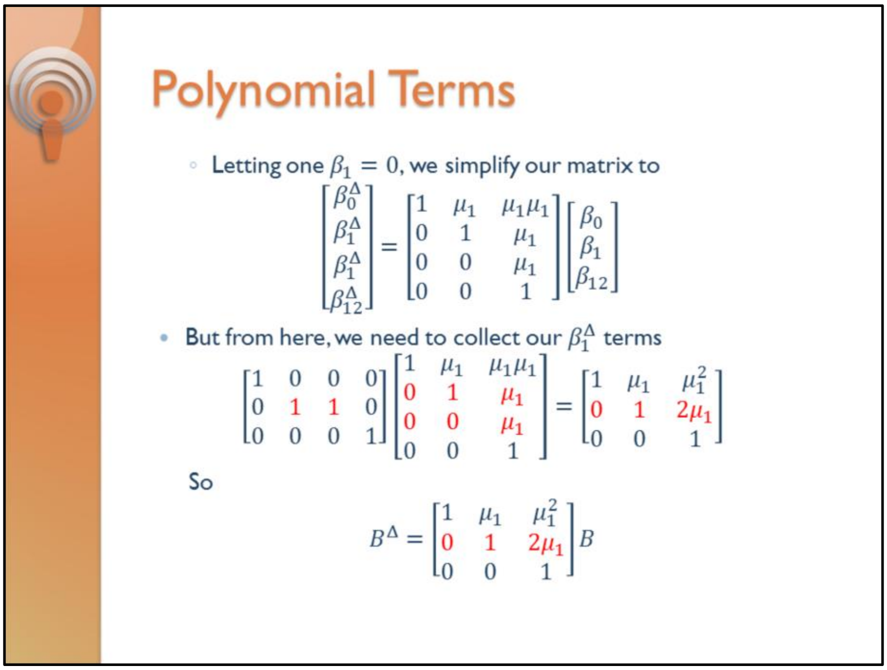
$$A \otimes A = \begin{bmatrix} 1 & \mu_1 & \mu_1 & \mu_1\mu_1 \\ 0 & 1 & 0 & \mu_1 \\ 0 & 0 & 1 & \mu_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- However, this is a matrix that starts with two β_1 and returns two β_1^Δ.

$$\begin{bmatrix} \beta_0^\Delta \\ \beta_1^\Delta \\ \beta_1^\Delta \\ \beta_{12}^\Delta \end{bmatrix} = \begin{bmatrix} 1 & \mu_1 & \mu_1 & \mu_1\mu_1 \\ 0 & 1 & 0 & \mu_1 \\ 0 & 0 & 1 & \mu_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_1 \\ \beta_{12} \end{bmatrix}$$

So we will need to collect terms in our transformation matrix.

Begin with the usual Kronecker operation for a factorial model.



We start to simplify like we did with the partial factorial models.

However, now zeroing out a column no longer leaves us with any rows of zeros.

This transformation still produces two β_1^Δ terms.

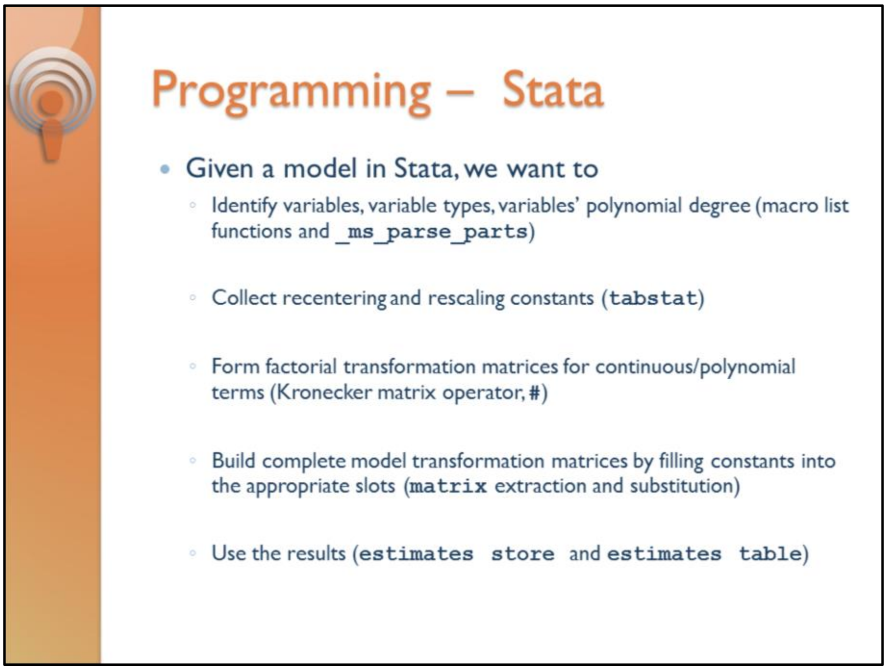
So we collect those terms: we add them together.

This is the result.

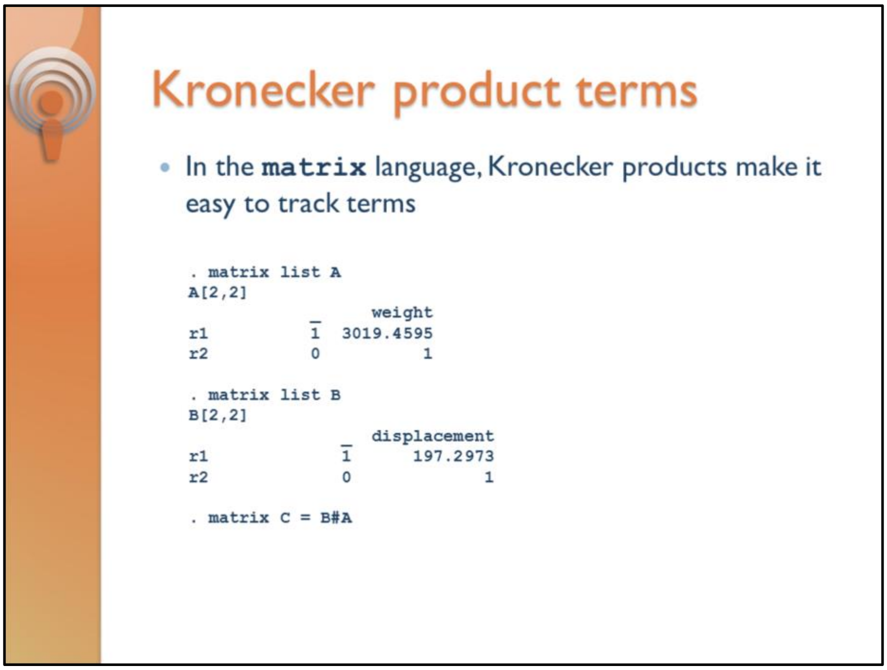
There are more terms to collect in higher order models, but this is the basic idea.

Math Summary

- We have building blocks for:
  - Continuous variables
  - Categorical variables
  - Polynomial terms
- We can combine them as:
  - Factorial models
  - Subsets of terms from factorial models
    - *(As long as no higher-order terms appear without their related lower-order terms)*



(I'm not going to go through these in order, but just highlight those parts I thought were obscure yet critically useful.)



One issue with these transformation matrices is keeping track of which rows and which columns relate to which parameters.

In the matrix language, Stata’s Kronecker operator makes it easy to keep track of your terms.

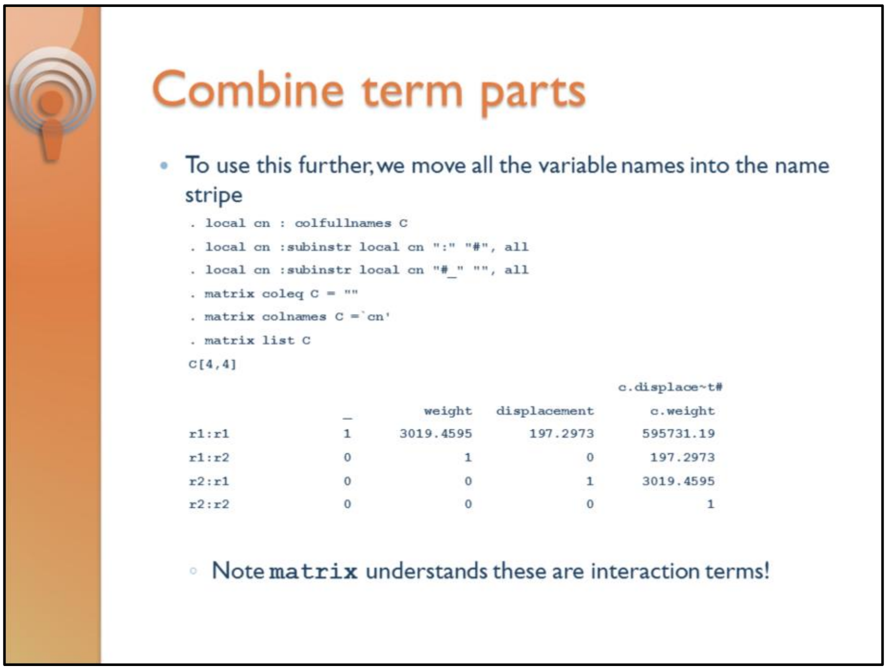
Kronecker product terms

- Column/row names are returned with the form
  equation(B):name(A)

```
. matrix list C

C[4,4]
                                 displacem~t:  displacem~t:
                   _      weight            _        weight
r1:r1              1   3019.4595     197.2973     595731.19
r1:r2              0           1            0      197.2973
r2:r1              0           0            1     3019.4595
r2:r2              0           0            0             1
```

- Note the **name** stripe is used, but the **equation** stripe is lost.

It returns two-part names, with the equation part from the first matrix, and the name part from the second matrix. The parts are separated by a colon.



In subsequent operations, the equation part is lost, so we need to move term names around if we want to keep them.

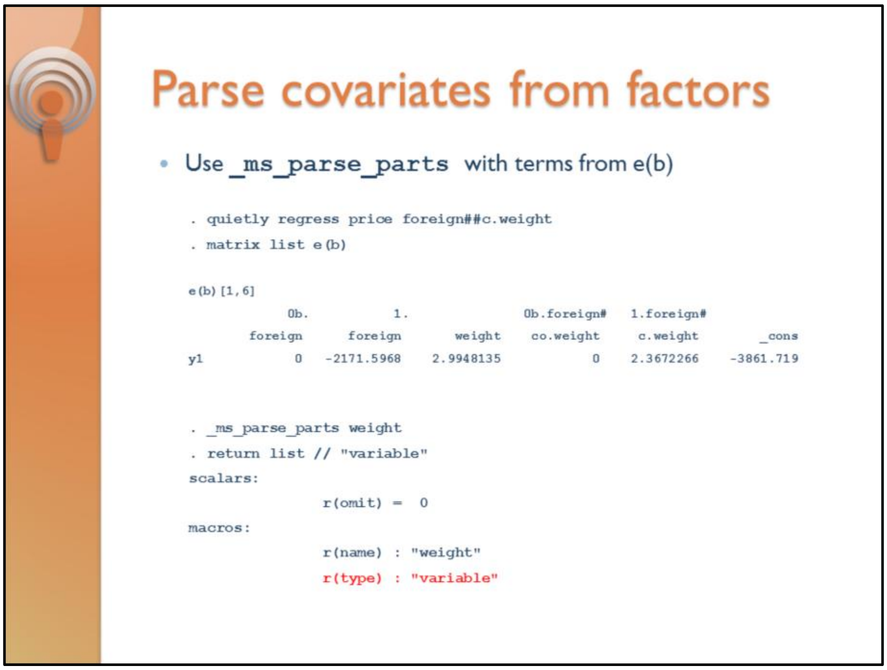
Kronecker product terms

- And we can keep building …

```
. matrix C = D#C

. matrix list C
```

C[8,8]

	_	weight	displacement	c.displace~t# c.weight	mpg	c.mpg# c.weight	c.mpg# c.displace~t	c.mpg# c.displace~t# c.weight
r1:r1	1	3019.4595	197.2973	595731.19	21.297297	64306.326	4201.8992	12687464
r1:r2	0	1	0	197.2973	0	21.297297	0	4201.8992
r1:r1	0	0	1	3019.4595	0	0	21.297297	64306.326
r1:r2	0	0	0	1	0	0	0	21.297297
r2:r1	0	0	0	0	1	3019.4595	197.2973	595731.19
r2:r2	0	0	0	0	0	1	0	197.2973
r2:r1	0	0	0	0	0	0	1	3019.4595
r2:r2	0	0	0	0	0	0	0	1



Another thing I want to do is separate out intercept terms from all the higher order terms.

The Stata command _ms_pars_parts is an amazingly useful tool!

You give it the name of a term, and it parses it into parts.

Parse factors from covariates

- Factors

```
. _ms_parse_parts 1.foreign
. return list // "factor"
scalars:
              r(base) =  0
             r(level) =  1
              r(omit) =  0
macros:
              r(name) : "foreign"
                r(op) : "1"
              r(type) : "factor"
```

Parse interactions

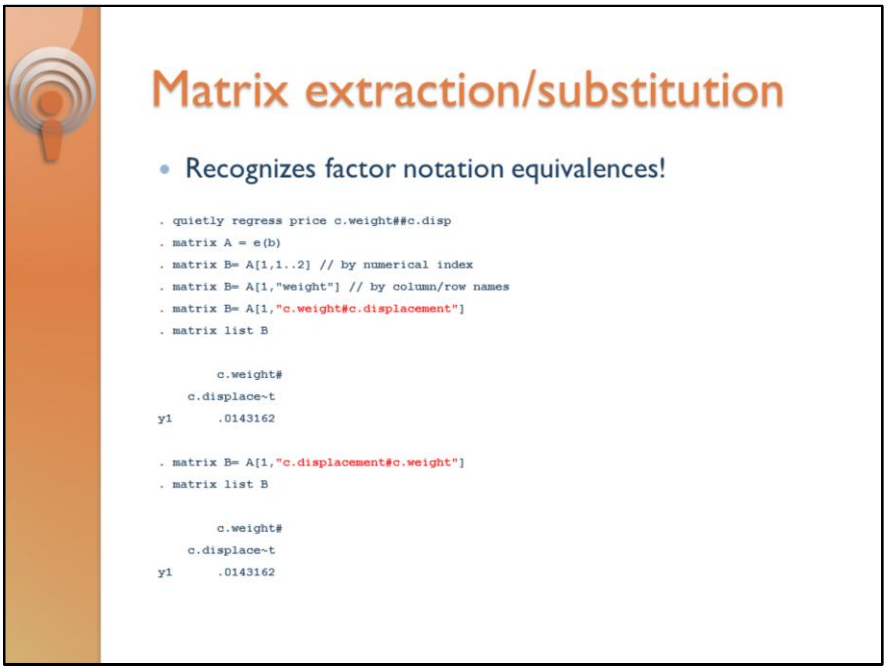
- Interactions

```
. _ms_parse_parts 1.foreign#c.weight
. return list // "interaction"

scalars:
              r(base1) =  0
             r(level1) =  1
            r(k_names) =  2
               r(omit) =  0
macros:
              r(name2) : "weight"
                r(op2) : "c"
              r(name1) : "foreign"
                r(op1) : "1"
               r(type) : "interaction"
```

Parse polynomials

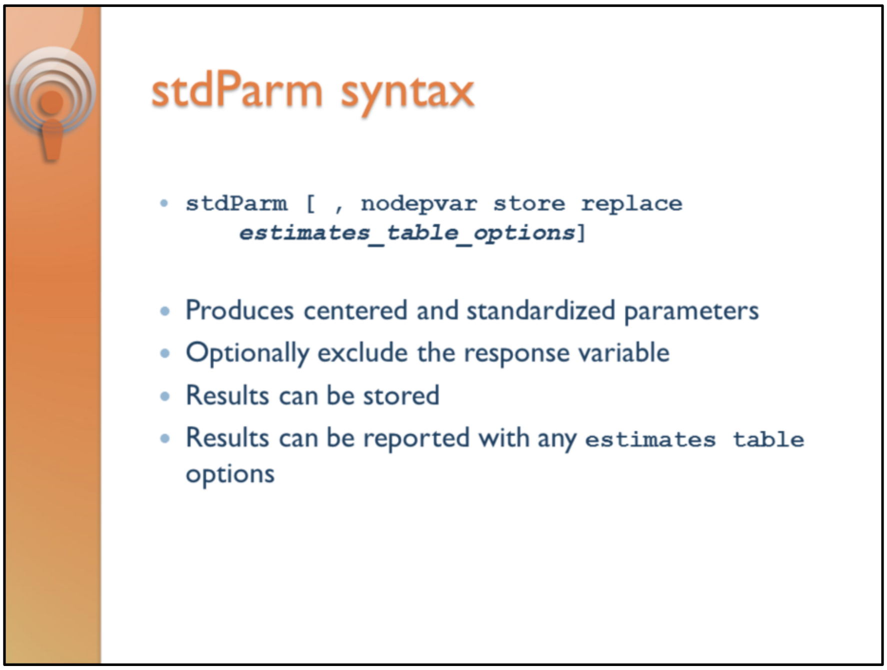
- Polynomial terms require some extra parsing

```
. _ms_parse_parts whatever#c.whatever

. return list // polynomial as interaction

scalars:
            r(k_names) =  2
               r(omit) =  0

macros:
              r(name2) : "whatever"
                r(op2) : "c"
              r(name1) : "whatever"
                r(op1) : "c"
               r(type) : "interaction"
```



Once we have sorted the covariates from the categorical variables, and formed our transformation matrix for the covariates, we can use matrix extraction and substitution to plug the components into a larger parameter transformation matrix that accommodates the categorical terms.

Here, it is useful to realize that factor variable notation is built into matrix extraction and substitution.



So I’ve put these pieces together into a little software routine, that works after regress and glm.

stdParm use

```
. quietly regress price c.weight##c.mpg

. stdParm

----------------------------------------------------------
    Variable |   Original       Centered     Standardized
-------------+--------------------------------------------
      weight |    5.0670077      .98475137      .25948245
         mpg |    396.78438     -181.98425     -.35696623
             |
   c.weight#|
       c.mpg |   -.19167955     -.19167955     -.29221218
             |
       _cons |   -5944.8806     -686.28559     -.23267895
----------------------------------------------------------
```

stdParm additional statistics

```
. stdParm, stats(N r2) star

--------------------------------------------------------------------
    Variable |   Original             Centered           Standardized
-------------+------------------------------------------------------
      weight |   5.0670077***         .98475137          .25948245
         mpg |   396.78438*          -181.98425         -.35696623
             |
   c.weight# |
       c.mpg |  -.19167955**         -.19167955**       -.29221218**
             |
       _cons |  -5944.8806           -686.28559         -.23267895
-------------+------------------------------------------------------
           N |          74                   74                 74
          r2 |   .35969597            .35969597          .35969597
--------------------------------------------------------------------
                            legend: * p<0.05; ** p<0.01; *** p<0.001
```

stdParm after logit

```
. quietly logit foreign c.price##c.weight

. stdParm

-----------------------------------------------------------
    Variable |   Original       Centered     Standardized
-------------+---------------------------------------------
       price |    .00331766      .00113549      3.3491337
      weight |   -.00141654     -.00587217     -4.5638148
             |
    c.price#|
    c.weight |   -7.227e-07     -7.227e-07     -1.6566669
             |
       _cons |   -4.5154515     -1.7920268     -1.7920268
-----------------------------------------------------------
```

stdParm, eform

```
. quietly logit foreign c.price##c.weight

. stdParm, eform

----------------------------------------------------------
    Variable |   Original        Centered     Standardized
-------------+--------------------------------------------
       price |    1.0033232       1.0011361      28.478052
      weight |    .99858446       .99414503      .01042222
             |
    c.price#|
    c.weight |    .99999928       .99999928      .19077378
             |
       _cons |    .01093867       .16662211      .16662211
----------------------------------------------------------
```

Download/ install

- `net from http://www.ssc.wisc.edu/~hemken/Stataworkshops`

- Tinker with the source code, suggest improvements:
  https://github.com/Hemken/stdParm