Brett Beutell
Econ 312 S
Parker

Final Project: Heteroskedasticity

Introduction

This project analyzes via Monte Carlo simulation the performance of three different methods for dealing with heteroskedasticity. First, weighted least squares are used based on known variances, then weighted least squares are used based on unknown variances. Finally, the two WLS methods are compared with the use of robust standard errors found during the ordinary least squares process. All relevant code and command prompts not explicitly listed can be found in the Appendices.

Data Generation Process

First, 5000 random "independent" observations were created from a uniform distribution in Stata **[Appendix A]**. These were the experiment's x-values. Subsequently, dependent-variable observations were created using varying degrees of heteroskedastic error functions. Error terms were randomly generated from a normal distribution a mean of zero (because heteroskedasticity does not bias our estimate of the parameter) and non-constant variance. The test equation was a single-variable, simple regression. The specifics of each heteroskedastic pattern are also listed below:

Assumed regression equation:

$$y_i = \beta_1 + \beta_2 * x_i + e_i$$

Patterns of heteroskedasticity:

$$1.\ var(e_i) = \sigma_i^2 = \sigma^2 * x_i$$

$$2.\ var(e_i) = \sigma_i^2 = \sigma^2 * x_i^2$$

$$3.\ var(e_i) = \sigma_i^2 = \sigma^2 * x_i^3$$

$$4.\ var(e_i) = \sigma_i^2 = \sigma^2 * \sqrt{x_i}$$

$$5.\ var(e_i) = \sigma_i^2 = \sigma^2 * \ln(x_i + 1)$$

$$6.\ var(e_i) = \sigma_i^2 = \sigma^2 * [\sin(x_i) + 1]$$

The true values of the parameters β_1 and β_2 were set equal to zero and one, respectively. For simplicity, the true value of σ^2 was also set equal to one.

Theory tells us that heteroskedasticity has two main implications. The first is that the least squares estimation procedure is still linear and unbiased, but it is no longer the best process of estimation. Second, the standard errors achieved through ordinary least squares estimation are wrong, meaning our t-tests and confidence intervals are not accurate.

Hence, the focus of this investigation will primarily be on the standard errors of the β_2 parameter, and whether or not they lead us to reject the null hypothesis that the β_2 parameter is equal to its true value (1). Because there are 5000 observations and the regression is estimating two parameters, all t-tests will have **4998 degrees of freedom**. For a two-tailed test at the 5% significance level, this renders a critical t-value of **+/- 1.96043870**.

[A] Weighted Least Squares Based on Known Variances (10,000 simulations)

```
. summ Kb2*

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
       Kb2_1 |     10000    .9999314    .0070772   .9715984   1.026385
       Kb2_2 |     10000    .9999381    .0142252   .9455313   1.065772
       Kb2_3 |     10000     1.00001    .0173253   .9308503   1.063522
       Kb2_4 |     10000    1.000105    .0060697    .978258   1.026211
       Kb2_5 |     10000    .9999285    .0043244   .9829566   1.017783
-------------+--------------------------------------------------------
       Kb2_6 |     10000     .999938    .0053343   .9804298   1.021367
```

Consistent with theory, the estimation of the β_2 parameter appears to be unbiased even in the presence of heteroskedasticity. All six simulations are very close to having a mean of 1, which is the true value of the parameter.

To test the efficacy of our standard error calculation, though, it is necessary to create a series of t-statistics from the data.

```
. summ Kse2*

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
      Kse2_1 |     10000    .0070608    .0000703   .0068015   .0073441
      Kse2_2 |     10000    .0141662    .0001417   .0136043   .0147508
      Kse2_3 |     10000    .0172716    .0001742   .0166915   .0179458
      Kse2_4 |     10000    .0060622    .0000615   .0058516   .0062786
      Kse2_5 |     10000    .0042936    .0000432   .0041274   .0044714
-------------+--------------------------------------------------------
      Kse2_6 |     10000    .0053367    .0000533   .0051198   .0055473


g Kb2_1t = (Kb2_1 - 1) / Kse2_1
g Kb2_2t = (Kb2_2 - 1) / Kse2_2
g Kb2_3t = (Kb2_3 - 1) / Kse2_3
g Kb2_4t = (Kb2_4 - 1) / Kse2_4
g Kb2_5t = (Kb2_5 - 1) / Kse2_5
g Kb2_6t = (Kb2_6 - 1) / Kse2_6
```

```
. count if (Kb2_1t >= 1.96043870) | (Kb2_1t <= -1.96043870)
  500

. count if (Kb2_2t >= 1.96043870) | (Kb2_2t <= -1.96043870)
  502

. count if (Kb2_3t >= 1.96043870) | (Kb2_3t <= -1.96043870)
  471

. count if (Kb2_4t >= 1.96043870) | (Kb2_4t <= -1.96043870)
  491

. count if (Kb2_5t >= 1.96043870) | (Kb2_5t <= -1.96043870)
  499

. count if (Kb2_6t >= 1.96043870) | (Kb2_6t <= -1.96043870)
  513
```

Under the heteroskedastic error assumption of $\mathbf{var(e_i) = \sigma^{2*} x_i}$, we achieved theoretically perfect results. Of our ten thousand simulations, exactly 5% would have led us to reject the null hypothesis that the true value of β_2 is 1. In fact, all of the simulations seem to have produced reasonable standard errors. The largest deviation from what we would expect was for $\mathbf{var(e_i) = \sigma^{2*} x_i^3}$, which rejected the (true) null hypothesis 4.71% of the time.

[B] Weighted Least Squares Based on Estimated Variances (10,000 simulations)

```
. summ Ub2*

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
       Ub2_1 |     10000    .9999353    .0088545   .9690486   1.040536
       Ub2_2 |     10000    .9998172    .0189966   .9301063   1.073115
       Ub2_3 |     10000    .9998518    .0394025   .8551449   1.143614
       Ub2_4 |     10000    .9998813    .0064553   .9717839   1.025519
       Ub2_5 |     10000    .9999789     .005465   .9796793    1.01945
-------------+--------------------------------------------------------
       Ub2_6 |     10000    .9999493    .0059335   .9772751   1.024626
       Ub2_7 |     10000    .9999837    .0345202   .8645389   1.124931
```

Again, estimates of the parameter of interest seem unbiased. They are very close to one.

```
. summ Use2*

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
      Use2_1 |     10000    .0100597    .0001073   .0096088    .010475
      Use2_2 |     10000    .0204653    .0002346   .0195657   .0212907
      Use2_3 |     10000    .0382149    .0009611     .03544    .042196
      Use2_4 |     10000     .006961    .0000719    .006704   .0072448
      Use2_5 |     10000     .006131    .0000644   .0058846    .006384
-------------+--------------------------------------------------------
      Use2_6 |     10000    .0053437    .0000619   .0051163   .0055948
```

On the whole, estimated standard errors from this simulation appear to be larger than those obtained by weighted least squares using known variances. The impact of this, though, can only be observed via t-tests:

```
. g Ub2_1t = (Ub2_1 - 1) / Use2_1

. g Ub2_2t = (Ub2_2 - 1) / Use2_2

. g Ub2_3t = (Ub2_3 - 1) / Use2_3

. g Ub2_4t = (Ub2_4 - 1) / Use2_4

. g Ub2_5t = (Ub2_5 - 1) / Use2_5

. g Ub2_6t = (Ub2_6 - 1) / Use2_6


. count if (Ub2_1t >= 1.96043870) | (Ub2_1t <= -1.96043870)
  256

. count if (Ub2_2t >= 1.96043870) | (Ub2_2t <= -1.96043870)
  345

. count if (Ub2_3t >= 1.96043870) | (Ub2_3t <= -1.96043870)
  550
```

```
. count if (Ub2_4t >= 1.96043870) | (Ub2_4t <= -1.96043870)
  338

. count if (Ub2_5t >= 1.96043870) | (Ub2_5t <= -1.96043870)
  266

. count if (Ub2_6t >= 1.96043870) | (Ub2_6t <= -1.96043870)
  768
```

At first glance, it seems that using estimated variances produces radically different results for our t-tests. Only in the case of $\mathbf{var(e_i) = \sigma^{2*} x_i^3}$ did the count of significant t-statistics near the number we would expect; it rejected 5.5% of results.

Otherwise, the simulations suggest that the method of weighted least squares using estimated variances is too conservative in its rejection of the (true) null hypothesis. For $\mathbf{var(e_i) = \sigma^{2*} x_i}$ and $\mathbf{var(e_i) = \sigma^{2*} ln(x_i + 1)}$, the null hypothesis was rejected only half as many times as we would expect from theory. For $\mathbf{var(e_i) = \sigma^{2*} x_i^2}$ and $\mathbf{var(e_i) = \sigma^{2*} \sqrt{x_i}}$, slightly fewer than 3.5% of the results were rejected.

However, the glaring exception to the conclusion that weighted least squares based on known variances is too conservative lies in the simulation with a sinuous error variance term ($\mathbf{var(e_i) = \sigma^{2*} [sin(x_i) + 1]}$). In this case, it doesn't seem that the method of variance estimation was able to properly capture the (cyclical) nature of the error terms' variances. As a consequence, substantially more results than we would expect from the simulation were rejected as having true values different from one. (Noteworthy is that this simulation was comparatively too liberal in its rejection of the null hypothesis for weighted least squares with known variances as well.)

[C] Ordinary Least Squares with Robust Standard Errors (10,000 simulations)

```
. summ RoB*

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        RoB1 |     10000    .9999235    .0077572    .973525   1.032184
        RoB2 |     10000    .9996687    .0221183    .914623   1.088525
        RoB3 |     10000    .9998195    .0647119   .7575594   1.259192
        RoB4 |     10000    1.000005    .0048954   .9817757   1.019442
        RoB5 |     10000    1.000052    .0042597   .9849213   1.018259
-------------+--------------------------------------------------------
        RoB6 |     10000    .9999832    .0041302   .9841889   1.014413
        RoB7 |     10000    .9998409    .0244042   .9095108   1.097674


. summ Rose*

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
       Rose1 |     10000    .0077483     .000097   .0074013   .0081201
       Rose2 |     10000    .0219194    .0003146   .0207285   .0229809
       Rose3 |     10000    .0648196     .001016   .0606939   .0694729
       Rose4 |     10000    .0048866    .0000527   .0046728   .0050755
       Rose5 |     10000    .0042404    .0000464   .0040574   .0043922
-------------+--------------------------------------------------------
       Rose6 |     10000    .0041722    .0000393    .004003   .0043218
       Rose7 |     10000    .0245702    .0002706    .023654   .0256004


. g RoB1_t = (RoB1 - 1)/Rose1

. g RoB2_t = (RoB2 - 1)/Rose2

. g RoB3_t = (RoB3 - 1)/Rose3

. g RoB4_t = (RoB4 - 1)/Rose4

. g RoB5_t = (RoB5 - 1)/Rose5

. g RoB6_t = (RoB6 - 1)/Rose6

. g RoB7_t = (RoB7 - 1)/Rose7

. count if (RoB1_t >= 1.96043870) | (RoB1_t <= -1.96043870)
  515

. count if (RoB2_t >= 1.96043870) | (RoB2_t <= -1.96043870)
  487

. count if (RoB3_t >= 1.96043870) | (RoB3_t <= -1.96043870)
  461

. count if (RoB4_t >= 1.96043870) | (RoB4_t <= -1.96043870)
  490
```

```
. count if (RoB5_t >= 1.96043870) | (RoB5_t <= -1.96043870)
  496

. count if (RoB6_t >= 1.96043870) | (RoB6_t <= -1.96043870)
  492

. count if (RoB7_t >= 1.96043870) | (RoB7_t <= -1.96043870)
  482
```

Robust standard errors seem to be much more conservative than their counterparts from the simulation using weighted least squares with known variances. All except $\mathbf{var(e_i) = \sigma^{2*} x_i}$ were close to theoretically perfect, the largest deviation from 5% rejection being $\mathbf{var(e_i) = \sigma^{2*} x_i^3}$, just as it was in [A]. This suggests that exponential heteroskedastic error functions are dealt with more conservatively through all three methods.

Robust standard errors also seem to be the most capable of dealing with sinuous heteroskedasticity. In this regard, they may be at least as accurate as weighted least squares with known variances, but they are certainly more accurate than weighted least squares with unknown variances. As noted above, this may have to do with the inability of variance function estimation to account for an error term with cyclical variance.

Conclusion

Weighted least squares with known variances appears to give the most accurate standard errors in the face of heteroskedasticity. However, it is not realistic to assume that we know the nature of a given error term's variance function. Therefore, in practice, we are more likely to find ourselves using weighted least squares with estimated variances. The procedure by which the error term's variance function is estimated seems to produce more conservative (larger) standard errors, as shown by the unexpectedly small number of estimates that were rejected at the 5% significance level. Also, this procedure did not seem capable of dealing with a sinuous error variance function; it produce too small of a standard error, leading to almost 8% of the simulation's results being rejected in a t-test at the 5% significance level.

Robust standard errors were less conservative than those obtained from weighted least squares with unknown variances, but they seemed to be more conservative than weighted least squares with known variances. (Although there isn't substantial evidence to support this claim, it appeared as though having an error variance function with a larger exponent would produce more conservative estimates of a parameter's standard error.) Finally, robust standard errors were able to deal with the aforementioned sinuous error variance function more effectively than weighted least squares with estimated variances.

Appendix A: DGP Code and Stata Commands with Comments

```
* Create 5000 randomly generated observations
       set obs 5000
       set seed 1000
       * Makes simulation replicable
       g x = 10*runiform()

       g vare1 = x
       g vare2 = x^2
       g vare3 = x^3
       g vare4 = sqrt(x)
       g vare5 = log(x+1)
       * "x+1" ensures nonnegative output
       g vare6 = sin(x) + 1
       * "+ 1" ensures nonnegative output
       g vare7 = 100*runiform()
       * a control for the "unknown" simulation, not used in report

*load the do-file for known variances

simulate  Kb1_1=r(Kb1_1) Kse1_1=r(Kse1_1) Kb2_1=r(Kb2_1) Kse2_1=r(Kse2_1)
Kb1_2=r(Kb1_2) Kse1_2=r(Kse1_2) Kb2_2=r(Kb2_2) Kse2_2=r(Kse2_2) Kb1_3=r(Kb1_3)
Kse1_3=r(Kse1_3) Kb2_3=r(Kb2_3) Kse2_3=r(Kse2_3) Kb1_4=r(Kb1_4) Kse1_4=r(Kse1_4)
Kb2_4=r(Kb2_4) Kse2_4=r(Kse2_4) Kb1_5=r(Kb1_5) Kse1_5=r(Kse1_5) Kb2_5=r(Kb2_5)
Kse2_5=r(Kse2_5) Kb1_6=r(Kb1_6) Kse1_6=r(Kse1_6) Kb2_6=r(Kb2_6) Kse2_6=r(Kse2_6),
reps(10000):knownVarFin

*load the do-file for unknown variances

simulate Ub1_1 =r(Ub1_1) Use1_1 =r(Use1_1) Ub2_1 =r(Ub2_1) Use2_1 =r(Use2_1) Ub1_2
=r(Ub1_2) Use1_2 =r(Use1_2) Ub2_2 =r(Ub2_2) Use2_2 =r(Use2_2) Ub1_3=r(Ub1_3) Use1_3
=r(Use1_3) Ub2_3=r(Ub2_3) Use2_3=r(Use2_3) Ub1_4=r(Ub1_4) Use1_4 =r(Use1_4) Ub2_4
=r(Ub2_4) Use2_4=r(Use2_4) Ub1_5=r(Ub1_5) Use1_5=r(Use1_5) Ub2_5=r(Ub2_5)
Use2_5=r(Use2_5) Ub1_6=r(Ub1_6) Use1_6=r(Use1_6) Ub2_6=r(Ub2_6) Use2_6=r(Use2_6)
Ub1_7=r(Ub1_7) Use1_7=r(Use1_7) Ub2_7=r(Ub2_7) Use2_7=r(Use2_7),
reps(10000):unknownVar

       * All of the following is for use in robust standard error calculation:
       sum(x)
       g    xdev2    = (x - r(mean))^2
       * Squared deviations from the mean

       egen denom    = total(xdev2)
       replace denom = denom^2
       * Denominator of robust variance estimation

       g    dof       = 5000/(5000-2)
       * Degrees of freedom adjustment

*load the do-file for robust

simulate RoB1=r(RoB1) Rose1=r(Rose1) RoB2=r(RoB2) Rose2=r(Rose2) RoB3=r(RoB3)
Rose3=r(Rose3) RoB4=r(RoB4) Rose4=r(Rose4) RoB5=r(RoB5) Rose5=r(Rose5) RoB6=r(RoB6)
Rose6=r(Rose6) RoB7=r(RoB7) Rose7=r(Rose7), reps(10000):robust2
```

Appendix B: Do-file for Weighted Least Squares with Known Variances

```
program knownVarFin, rclass

/*
Brett Beutell
Econ 312, Spring '12

Monte Carlo simulation of weighted least squares
with known variances
*/

//Generate errors

	g e1 = rnormal(0,sqrt(vare1))
	g e2 = rnormal(0,sqrt(vare2))
	g e3 = rnormal(0,sqrt(vare3))
	g e4 = rnormal(0,sqrt(vare4))
	g e5 = rnormal(0,sqrt(vare5))
	g e6 = rnormal(0,sqrt(vare6))

//Generate y variables based off of random errors
	g yk1 = 0 + 1*x + e1
	g yk2 = 0 + 1*x + e2
	g yk3 = 0 + 1*x + e3
	g yk4 = 0 + 1*x + e4
	g yk5 = 0 + 1*x + e5
	g yk6 = 0 + 1*x + e6

//Generate transformed y-variable
	g ys1 = yk1/sqrt(vare1)
	g ys2 = yk2/sqrt(vare2)
	g ys3 = yk3/sqrt(vare3)
	g ys4 = yk4/sqrt(vare4)
	g ys5 = yk5/sqrt(vare5)
	g ys6 = yk6/sqrt(vare6)

//Generate transformed x1 variable (result of dividing the constant)
	g x1s1 = 1/sqrt(vare1)
	g x1s2 = 1/sqrt(vare2)
	g x1s3 = 1/sqrt(vare3)
	g x1s4 = 1/sqrt(vare4)
	g x1s5 = 1/sqrt(vare5)
	g x1s6 = 1/sqrt(vare6)

//Generate transformed x2 variable
	g x2s1 = x/sqrt(vare1)
	g x2s2 = x/sqrt(vare2)
	g x2s3 = x/sqrt(vare3)
	g x2s4 = x/sqrt(vare4)
	g x2s5 = x/sqrt(vare5)
	g x2s6 = x/sqrt(vare6)


//Regress transformed model with suppressed constant
// Return scalars
	reg ys1 x1s1 x2s1, noc
		return scalar Kb1_1=_b[x1s1]
```

```
            return scalar Kse1_1=_se[x1s1]
            return scalar Kb2_1=_b[x2s1]
            return scalar Kse2_1=_se[x2s1]

      reg ys2 x1s2 x2s2, noc
            return scalar Kb1_2=_b[x1s2]
            return scalar Kse1_2=_se[x1s2]
            return scalar Kb2_2=_b[x2s2]
            return scalar Kse2_2=_se[x2s2]

      reg ys3 x1s3 x2s3, noc
            return scalar Kb1_3=_b[x1s3]
            return scalar Kse1_3=_se[x1s3]
            return scalar Kb2_3=_b[x2s3]
            return scalar Kse2_3=_se[x2s3]

      reg ys4 x1s4 x2s4, noc
            return scalar Kb1_4=_b[x1s4]
            return scalar Kse1_4=_se[x1s4]
            return scalar Kb2_4=_b[x2s4]
            return scalar Kse2_4=_se[x2s4]

      reg ys5 x1s5 x2s5, noc
            return scalar Kb1_5=_b[x1s5]
            return scalar Kse1_5=_se[x1s5]
            return scalar Kb2_5=_b[x2s5]
            return scalar Kse2_5=_se[x2s5]

      reg ys6 x1s6 x2s6, noc
            return scalar Kb1_6=_b[x1s6]
            return scalar Kse1_6=_se[x1s6]
            return scalar Kb2_6=_b[x2s6]
            return scalar Kse2_6=_se[x2s6]

// Drop variables to allow for repetition
      drop e*
      drop y*
      drop x1*
      drop x2*


end
```

Appendix C: Do-file for Weighted Least Squares with Estimated Variances

```
program unknownVar, rclass

/*
Brett Beutell
Econ 312, Spring '12

Monte Carlo simulation of weighted least squares
with unknown, estimated variances
*/

	g e  = rnormal(0,1)
	g e1 = rnormal(0,sqrt(vare1))
	g e2 = rnormal(0,sqrt(vare2))
	g e3 = rnormal(0,sqrt(vare3))
	g e4 = rnormal(0,sqrt(vare4))
	g e5 = rnormal(0,sqrt(vare5))
	g e6 = rnormal(0,sqrt(vare6))
	g e7 = rnormal(0,sqrt(vare7))


	g yuk1 = 0 + 1*x + e1
	g yuk2 = 0 + 1*x + e2
	g yuk3 = 0 + 1*x + e3
	g yuk4 = 0 + 1*x + e4
	g yuk5 = 0 + 1*x + e5
	g yuk6 = 0 + 1*x + e6
	g yuk7 = 0 + 1*x + e7


/*
The following process will be repeated seven times for each degree of
heteroskedasticity.
Only the first regression equation will have comments, which should illuminate what is
happening for each equation.
*/

	// Ordinary Least Squares Regression of y on x
	reg yuk1 x

	// Find the error terms from said regression
	predict ehat1, resid

	// Transform the error terms
	g ln_ehat1Sq = log(ehat1^2)

	// Regress the transformed error terms on x
	reg ln_ehat1Sq x

	// Get the fitted values from said regression
	predict ln_eVarHat1


	reg yuk2 x
	predict ehat2, resid
	g ln_ehat2Sq = log(ehat2^2)
	reg ln_ehat2Sq x
```

```
        predict ln_eVarHat2


        reg yuk3 x
        predict ehat3, resid
        g ln_ehat3Sq = log(ehat3^2)
        reg ln_ehat3Sq x
        predict ln_eVarHat3


        reg yuk4 x
        predict ehat4, resid
        g ln_ehat4Sq = log(ehat4^2)
        reg ln_ehat4Sq x
        predict ln_eVarHat4


        reg yuk5 x
        predict ehat5, resid
        g ln_ehat5Sq = log(ehat5^2)
        reg ln_ehat5Sq x
        predict ln_eVarHat5

        reg yuk6 x
        predict ehat6, resid
        g ln_ehat6Sq = log(ehat6^2)
        reg ln_ehat6Sq x
        predict ln_eVarHat6

        reg yuk7 x
        predict ehat7, resid
        g ln_ehat7Sq = log(ehat7^2)
        reg ln_ehat7Sq x
        predict ln_eVarHat7

// Exponentiate the fitted values from above to obtain variance estimates

        g eVarHat1 = exp(ln_eVarHat1)
        g eVarHat2 = exp(ln_eVarHat2)
        g eVarHat3 = exp(ln_eVarHat3)
        g eVarHat4 = exp(ln_eVarHat4)
        g eVarHat5 = exp(ln_eVarHat5)
        g eVarHat6 = exp(ln_eVarHat6)
        g eVarHat7 = exp(ln_eVarHat7)


// Create transformed y-variable for weighted least squares regression

        g ys1 = yuk1/sqrt(eVarHat1)
        g ys2 = yuk2/sqrt(eVarHat2)
        g ys3 = yuk3/sqrt(eVarHat3)
        g ys4 = yuk4/sqrt(eVarHat4)
        g ys5 = yuk5/sqrt(eVarHat5)
        g ys6 = yuk6/sqrt(eVarHat6)
        g ys7 = yuk7/sqrt(eVarHat7)

// Create transformed x1-variable for weighted least squares regression
```

```
        g x1s1 = 1/sqrt(eVarHat1)
        g x1s2 = 1/sqrt(eVarHat2)
        g x1s3 = 1/sqrt(eVarHat3)
        g x1s4 = 1/sqrt(eVarHat4)
        g x1s5 = 1/sqrt(eVarHat5)
        g x1s6 = 1/sqrt(eVarHat6)
        g x1s7 = 1/sqrt(eVarHat7)



// Create transformed x2-variable for weighted least squares regression

        g x2s1 = x/sqrt(eVarHat1)
        g x2s2 = x/sqrt(eVarHat2)
        g x2s3 = x/sqrt(eVarHat3)
        g x2s4 = x/sqrt(eVarHat4)
        g x2s5 = x/sqrt(eVarHat5)
        g x2s6 = x/sqrt(eVarHat6)
        g x2s7 = x/sqrt(eVarHat7)


// Regress transformed y-var on the transformed x-vars
// Suppress the constant because it is estimated by x1s variables


        reg ys1 x1s1 x2s1, noc
                return scalar Ub1_1=_b[x1s1]
                return scalar Use1_1=_se[x1s1]
                return scalar Ub2_1=_b[x2s1]
                return scalar Use2_1=_se[x2s1]

        reg ys2 x1s2 x2s2, noc
                return scalar Ub1_2=_b[x1s2]
                return scalar Use1_2=_se[x1s2]
                return scalar Ub2_2=_b[x2s2]
                return scalar Use2_2=_se[x2s2]

        reg ys3 x1s3 x2s3, noc
                return scalar Ub1_3=_b[x1s3]
                return scalar Use1_3=_se[x1s3]
                return scalar Ub2_3=_b[x2s3]
                return scalar Use2_3=_se[x2s3]

        reg ys4 x1s4 x2s4, noc
                return scalar Ub1_4=_b[x1s4]
                return scalar Use1_4=_se[x1s4]
                return scalar Ub2_4=_b[x2s4]
                return scalar Use2_4=_se[x2s4]

        reg ys5 x1s5 x2s5, noc
                return scalar Ub1_5=_b[x1s5]
                return scalar Use1_5=_se[x1s5]
                return scalar Ub2_5=_b[x2s5]
                return scalar Use2_5=_se[x2s5]

        reg ys6 x1s6 x2s6, noc
                return scalar Ub1_6=_b[x1s6]
```

```
            return scalar Use1_6=_se[x1s6]
            return scalar Ub2_6=_b[x2s6]
            return scalar Use2_6=_se[x2s6]

      reg ys7 x1s7 x2s7, noc
            return scalar Ub1_7=_b[x1s7]
            return scalar Use1_7=_se[x1s7]
            return scalar Ub2_7=_b[x2s7]
            return scalar Use2_7=_se[x2s7]


// Lastly, just have to drop a lot of variables
      drop e*
      drop y*
      drop ln*
      drop x1*
      drop x2*
end
```

Appendix D: Do-file for Robust Standard Errors via Ordinary Least Squares

```
program robust2, rclass

/*
Brett Beutell
Econ 312, Spring '12

Monte Carlo simulation of ordinary least squares
with robust standard errors
*/

// Data Generation
       g e1 = rnormal(0,sqrt(vare1))
       g e2 = rnormal(0,sqrt(vare2))
       g e3 = rnormal(0,sqrt(vare3))
       g e4 = rnormal(0,sqrt(vare4))
       g e5 = rnormal(0,sqrt(vare5))
       g e6 = rnormal(0,sqrt(vare6))
       g e7 = rnormal(0,sqrt(vare7))


       g yr1 = 0 + 1*x + e1
       g yr2 = 0 + 1*x + e2
       g yr3 = 0 + 1*x + e3
       g yr4 = 0 + 1*x + e4
       g yr5 = 0 + 1*x + e5
       g yr6 = 0 + 1*x + e6
       g yr7 = 0 + 1*x + e7


// OLS regression with robust standard errors, record results
// I trust Stata's robust standard error calculation much more than my own


       reg yr1 x, vce(r)
              return scalar RoB1 = _b[x]
              return scalar Rose1 = _se[x]

       reg yr2 x, vce(r)
              return scalar RoB2 = _b[x]
              return scalar Rose2 = _se[x]

       reg yr3 x, vce(r)
              return scalar RoB3 = _b[x]
              return scalar Rose3 = _se[x]

       reg yr4 x, vce(r)
              return scalar RoB4 = _b[x]
              return scalar Rose4 = _se[x]

       reg yr5 x, vce(r)
              return scalar RoB5 = _b[x]
              return scalar Rose5 = _se[x]

       reg yr6 x, vce(r)
              return scalar RoB6 = _b[x]
              return scalar Rose6 = _se[x]
```

```
       reg yr7 x, vce(r)
              return scalar RoB7 = _b[x]
              return scalar Rose7 = _se[x]


// Drop variables
       drop e*
       drop y*

end
```