Supplementary/Online Appendix for:

Don't Jettison the General Error Correction Model Just Yet: A Practical Guide to Avoiding Spurious Regression with the GECM

Peter K. Enns
peterenns@cornell.edu
Cornell University

Nathan J. Kelly
nathan.j.kelly@gmail.com
University of Tennessee

Takaaki Masaki
tmasaki@wm.edu
College of William & Mary

Patrick C. Wohlfarth
patrickw@umd.edu
University of Maryland, College Park

Contents

Appendix 1 The bias in α_1 decreases with T

The following Stata replication code reproduces the results in Footnote 10 which show that the bias in α_1 decreases as the sample size increases.

```
#d ;
set more off ;
clear ;
set seed 456789 ;
cap mkdir "Change Directory Here" ;
global nocoint "Change Directory Here" ;
cap mkdir "$nocoint\results" ;

/*bivariate*/
#d ;
cap program drop bivariate ;
program define bivariate, rclass ;
syntax [, obs(integer 200)] ;
#d ;
drop _all ;
set obs ‘obs’ ;
gen t = _n ;
tsset t ;
gen e_1 = invnorm(uniform()) ;
gen x_1=e_1 if t==1 ;
gen u=invnorm(uniform()) ;
gen y=u if t==1 ;
replace x_1= l.x_1+e_1 if t>1 ;
replace y= l.y + u if t>1 ;
/*drop the first 100 observations*/
drop if t<=100 ;
/*Error Correction Model (ECM)*/
reg d.y l.y d.x_1 l.x_1 ;
return scalar beta_ly = _b[l.y] ; /*Save these coefficients to check bias.*/
return scalar obs=‘obs’ ;
end ;

#d ;
postfile bivariate
obs beta_mean_ly
using "$nocoint\results\nocoint_1", replace ;
foreach obs of numlist 200 300 { ;
simul obs=r(obs) beta_ly=r(beta_ly),
reps(10000):
bivariate, obs(‘obs’) ;
/*Save coefficients*/
local obs_n=‘obs’-100 ;
sum beta_ly ;
local beta_mean_ly = round(r(mean), .001) ;
post bivariate (‘obs_n’) (‘beta_mean_ly’) ;
} ;
postclose bivariate ;
clear ;

/*five IVs*/
#d ;
cap program drop fiveivs ;
program define fiveivs, rclass ;
syntax [, obs(integer 200)] ;
#d ;
drop _all ;
set obs ‘obs’ ;
gen t = _n ;
tsset t ;
gen e_1 = invnorm(uniform()) ;
gen x_1=e_1 if t==1 ;
gen e_2 = invnorm(uniform()) ;
gen x_2=e_2 if t==1 ;
gen e_3 = invnorm(uniform()) ;
gen x_3=e_3 if t==1 ;
gen e_4 = invnorm(uniform()) ;
gen x_4=e_4 if t==1 ;
gen e_5 = invnorm(uniform()) ;
gen x_5=e_5 if t==1 ;
gen u=invnorm(uniform()) ;
gen y=u if t==1 ;
replace x_1= l.x_1+e_1 if t>1 ;
replace x_2= l.x_2+e_2 if t>1 ;
replace x_3= l.x_3+e_3 if t>1 ;
replace x_4= l.x_4+e_4 if t>1 ;
replace x_5= l.x_5+e_5 if t>1 ;
replace y= l.y + u if t>1 ;
drop if t<=100 ;
/*Error Correction Model (ECM)*/
reg d.y l.y d.x_1 l.x_1 d.x_2 l.x_2 d.x_3 l.x_3 d.x_4 l.x_4 d.x_5 l.x_5 ;
return scalar beta_ly = _b[l.y] ; /*Save these coefficients to check bias.*/
return scalar obs=‘obs’ ;
end ;

#d ;
```

```
postfile fiveivs
obs beta_mean_ly
using "$nocoint\results\nocoint_5", replace ;
foreach obs of numlist 200 300 { ;
simul obs=r(obs) beta_ly=r(beta_ly),
reps(10000):
fiveivs, obs(‘obs’) ;
/*Save coefficients*/
local obs_n=‘obs’-100 ;
sum beta_ly ;
local beta_mean_ly = round(r(mean), .001) ;
post fiveivs (‘obs_n’) (‘beta_mean_ly’) ;
} ;
postclose fiveivs ;
clear ;
```

Appendix 2 Bounded Unit Root Simulations

The Stata simulation code comes from “I.1.5 Multivariate (2IVs) Bounded Data - Stata” in Grant and Lebo. We then use Grant and Lebo’s Table I.1: Parameter Values for Generating Bounded Series to identify the specific parameters. Below, we reproduce their Table I.1 and the exact code we used.

Table A-1: Reproduction of Grant and Lebo, Table I.1: Parameter Values for Generating Bounded Series

	Parameters		
σ & Bounds	$\alpha_1 = \alpha_2$	κ	τ
$\sigma = 1$			
(0,100)	1.5	73.5	49
(49,71)	1.5	16.5	59
$\sigma = 2$			
(0,100)	0.5	24.5	49
(49,71)	0.5	5.49998	59
$\sigma = 3$			
(0,100)	0.3	14.7	49
(49,71)	0.3	3.29864	59

Note: MacKinnon Values with 2 IVs: T=60:-3.565

The following Stata do file replicates the results we report in Table 1.

```
#d ;
global sim "Change Directory Here" ;
cap mkdir "$sim\ECM\" ;
cap mkdir "$sim\ECM\BI\" ;
cap mkdir "$sim\ECM\MV\" ;

/*bivariate*/
#d ;
```

```
cd "$sim\ECM\BI\" ;
clear all ;
foreach b of numlist 1 2 { ;
foreach t of numlist 60 100 150 { ;
foreach v of numlist 1 2 3 { ;
clear ;
global nobs = ‘t’ ;
global nmc = 10,000 ;
set seed 5000 ;
set obs ‘t’ ;
gen id = _n ;
tsset id ;
/* Set the values of the parameters*/
local obs_df =‘t’-4 ;
scalar ecmsig = -1*invttail(‘obs_df’,0.05) ;
scalar ecmMCVm1 = -3.2145-3.21/‘obs_df’-2/(‘obs_df’^2) +17/(‘obs_df’^3) ;
scalar up = invttail(‘obs_df’,0.025) ;
scalar lp = -1*invttail(‘obs_df’,0.025)  ;

/* Set values for bounds (Table H.1 in Lebo and Grant)*/
if (‘v’==1 & ‘b’==1) { ;
scalar a = 1.5 ;
scalar b = 1.5 ;
scalar tau = 49 ;
scalar k = 73.5 ;
gen dv1 = (99-1)*runiform()+1 ;
} ;

if (‘v’==2 & ‘b’==1) { ;
scalar a = 0.5 ;
scalar b = 0.5 ;
scalar tau = 49 ;
scalar k = 24.5 ;
gen dv1 = (99-1)*runiform()+1 ;
} ;

if (‘v’==3 & ‘b’==1) { ;
scalar a = 0.3 ;
scalar b = 0.3 ;
scalar tau = 49 ;
scalar k = 14.7 ;
gen dv1 = (99-1)*runiform()+1 ;
} ;

if (‘v’==1 & ‘b’==2) { ;
scalar a = 1.5 ;
scalar b = 1.5 ;
scalar tau = 59 ;
scalar k = 16.5 ;
gen dv1 = (59-50)*runiform()+50 ;
} ;

if (‘v’==2 & ‘b’==2) { ;
scalar a = 0.5 ;
scalar b = 0.5 ;
scalar tau = 59 ;
scalar k = 5.49998 ;
gen dv1 = (59-50)*runiform()+50 ;
} ;

if (‘v’==3 & ‘b’==2) { ;
scalar a = 0.3 ;
scalar b = 0.3 ;
scalar tau = 59 ;
scalar k = 3.29864 ;
gen dv1 = (59-50)*runiform()+50 ;
} ;
/* Generating starting values for DV*/
/*gen dv1 = (59-50)*runiform()+50*/
gen iv1 = 0 ;
/* generating errors*/
gen e = . ;
/*gen e2 = . ;
gen e3 = . ;*/
gen u = . ;
gen v = . ;
tempname sim ;
postfile ‘sim’ m1asigMCV m1asigMCV_df using MCMC_bounded_BI_v‘v’_b‘b’_t‘t’, replace ;
quietly { ;
forvalues i = 1/$nmc { ;
replace e = rnormal(0,‘v’) ;
replace u = rnormal() ;
replace v = rnormal() ;
replace dv1 = l.dv1 + exp(-k)*(exp((-a)*(l.dv1 - tau)) - (exp((b)*(l.dv1 - tau)))) + e in 2/$nobs ;
replace iv1 = l.iv1 + u in 2/$nobs ;

/*dicky-fuller test*/
/*Identify appropriate lags for the Dicky-Fuller test (y). */
reg d.dv1 l.dv1 ;
predict resid_dv1, r ;
wntestq resid_dv1 ;
local i=r(p) ;
drop resid_dv1 ;
local j = 0 ;
while ‘i’<0.05 { ; /*The "while" command allows us to keep running the set of stata commands
                    enclosed in the braces until the p-value from the White noise Q test becomes
```

```
                   greater than 0.05. */
local j ='j'+ 1 ; /*If the p-value is lower than 0.05, which is a sign of the presence of serial correlation,
'j' increases by one unit such that one more lag of ld.y is added in the equation below.*/
reg d.dv1 l.dv1 l(1/'j')d.dv1 ;
predict resid_dv1, r ;
wntestq resid_dv1 ;
local i=r(p) ;
drop resid_dv1 ;
} ;

/*Augmented Dicky-Fuller test */
dfuller dv1, lags('j') ; /*dfuller: dfuller=Dickey-Fuller (DF) Test
H0: I(1) or y is unit root.
H1: (p-1)<0 or y is stationary.
The appropriate lags 'j' is imputed so as to address
the issues of serial correlation.
If p-value is greater than 0.05, it indicates that y
has a unit root.
Note that rejecting the null hypothesis does not mean
we can accept H0 (Woodridge 2009, 633)!
dfuller by default regresses d.y on l.y.
How many lags to be included often depends on
the frequency of the data.*/

scalar p_value_df_dv1=r(p) ; /*Save p-value to see if the
estimated coefficient on l.y is
statistically significant at the .05 level
given the Dicket-Fuller Distribution.*/

reg d.dv1 l.dv1 d.iv1 l.iv1 ;
scalar m1a1 = _b[l.dv1] ;
scalar m1sea1 = _se[l.dv1] ;
scalar m1ta1 = m1a1/m1sea1 ;
scalar m1asig = m1ta1<ecmsig ;
scalar m1asigMCV = m1ta1<=ecmMCVm1 ;
scalar m1asigMCV_df = (m1ta1<=ecmMCVm1 & p_value_df_dv1>0.05) ;
post ‘sim’ (m1asigMCV) (m1asigMCV_df) ;
} ;
} ;
postclose ‘sim’ ;
} ;
} ;
} ;

/*multivariate*/
#d ;
cd "$sim\ECM\MV\" ;
foreach b of numlist 1 2 { ;
foreach t of numlist 60 100 150 { ;
foreach v of numlist 1 2 3 { ;
#d ;
clear ;
global nobs = ‘t’ ;
global nmc = 10,000 ;
set seed 5000 ;
set obs ‘t’ ;
gen id = _n ;
tsset id ;
/* Set the values of the parameters*/
local obs_df =‘t’-6 ;
scalar ecmsig = -1*invttail(‘obs_df’,0.05) ;
scalar ecmMCVm1 = -3.5057-3.27/‘obs_df’+1.1/(‘obs_df’^2)-34/(‘obs_df’^3) ;
scalar up = invttail(‘obs_df’,0.025) ;
scalar lp = -1*invttail(‘obs_df’,0.025)  ;
```

```
/* Set values for bounds (Table H.1 in Lebo and Grant)*/
#d ;
if (‘v’==1 & ‘b’==1) { ;
#d ;
scalar a = 1.5 ;
scalar b = 1.5 ;
scalar tau = 49 ;
scalar k = 73.5 ;
gen dv1 = (99-1)*runiform()+1 ;
} ;

#d ;
if (‘v’==2 & ‘b’==1) { ;
#d ;
scalar a = 0.5 ;
scalar b = 0.5 ;
scalar tau = 49 ;
scalar k = 24.5 ;
gen dv1 = (99-1)*runiform()+1 ;
} ;

#d ;
if (‘v’==3 & ‘b’==1) { ;
#d ;
scalar a = 0.3 ;
scalar b = 0.3 ;
scalar tau = 49 ;
scalar k = 14.7 ;
gen dv1 = (99-1)*runiform()+1 ;
} ;
```

```
#d ;
if (‘v’==1 & ‘b’==2) { ;
#d ;
scalar a = 1.5 ;
scalar b = 1.5 ;
scalar tau = 59 ;
scalar k = 16.5 ;
gen dv1 = (59-50)*runiform()+50 ;
} ;

#d ;
if (‘v’==2 & ‘b’==2) { ;
#d ;
scalar a = 0.5 ;
scalar b = 0.5 ;
scalar tau = 59 ;
scalar k = 5.49998 ;
gen dv1 = (59-50)*runiform()+50 ;
} ;

#d ;
if (‘v’==3 & ‘b’==2) { ;
#d ;
scalar a = 0.3 ;
scalar b = 0.3 ;
scalar tau = 59 ;
scalar k = 3.29864 ;
gen dv1 = (59-50)*runiform()+50 ;
} ;

/* Generating starting values for DV*/
/*gen dv1 = (59-50)*runiform()+50*/
```

```
#d ;
gen iv1 = 0 ;
gen iv2 = 0 ;
/* generating errors*/
#d ;
gen e = . ;
/*gen e2 = . ;
gen e3 = . ;*/
#d ;
gen u = . ;
gen v = . ;
tempname sim ;
postfile `sim' m1asigMCV m1asigMCV_df using MCMC_bounded_MV_v`v'_b`b'_t`t', replace ;
quietly { ;
forvalues i = 1/$nmc { ;
#d ;
replace e = rnormal(0,`v') ;
replace u = rnormal() ;
replace v = rnormal() ;
replace dv1 = l.dv1 + exp(-k)*(exp((-a)*(l.dv1 - tau)) - (exp((b)*(l.dv1 - tau)))) + e in 2/$nobs ;
replace iv1 = l.iv1 + u in 2/$nobs ;
replace iv2 = l.iv2 + v in 2/$nobs ;

/*dicky-fuller test*/
/*Identify appropriate lags for the Dicky-Fuller test (y). */
reg d.dv1 l.dv1 ;
predict resid_dv1, r ;
wntestq resid_dv1 ;
local i=r(p) ;
drop resid_dv1 ;
local j = 0 ;
while `i'<0.05 { ; /*The "while" command allows us to keep running the set of stata commands
```

```
                enclosed in the braces until the p-value from the White noise Q test becomes
                greater than 0.05. */
local j =‘j’+ 1 ; /*If the p-value is lower than 0.05, which is a sign of the presence of serial correlation,
‘j’ increases by one unit such that one more lag of ld.y is added in the equation below.*/
reg d.dv1 l.dv1 l(1/‘j’)d.dv1 ;
predict resid_dv1, r ;
wntestq resid_dv1 ;
local i=r(p) ;
drop resid_dv1 ;
} ;

/*Augmented Dicky-Fuller test */
dfuller dv1, lags(‘j’) ; /*dfuller: dfuller=Dickey-Fuller (DF) Test
H0: I(1) or y is unit root.
H1: (p-1)<0 or y is stationary.
The appropriate lags ‘j’ is imputed so as to address
the issues of serial correlation.
If p-value is greater than 0.05, it indicates that y
has a unit root.
Note that rejecting the null hypothesis does not mean
we can accept H0 (Woodridge 2009, 633)!
dfuller by default regresses d.y on l.y.
How many lags to be included often depends on
the frequency of the data.*/

scalar p_value_df_dv1=r(p) ; /*Save p-value to see if the
estimated coefficient on l.y is
statistically significant at the .05 level
given the Dicket-Fuller Distribution.*/

reg d.dv1 l.dv1 d.iv1 l.iv1 d.iv2 l.iv2 ;
scalar m1a1 = _b[l.dv1] ;
scalar m1sea1 = _se[l.dv1] ;
scalar m1ta1 = m1a1/m1sea1 ;
scalar m1asigMCV = m1ta1<=ecmMCVm1 ;
scalar m1asigMCV_df = (m1ta1<=ecmMCVm1 & p_value_df_dv1>0.05) ;
post ‘sim’ (m1asigMCV) (m1asigMCV_df) ;
} ;
} ;
postclose ‘sim’ ;
} ;
} ;
} ;
```

Appendix 3 Concerns with Grant and Lebo's Simulations of Kelly and Enns (2010)

Footnote 18 acknowledged that Grant and Lebo also conducted simulations to evaluate Kelly and Enns' data, but that serious concerns exist with these simulations casting further doubt on Grant and Lebo's criticisms of Kelly and Enns (2010). We details these concerns below.

When Grant and Lebo did conduct simulations to evaluate Kelly and Enns' data, they combined simulated series with actual data (as opposed to data simulated to mimic the actual data). This decision is problematic because when they retain the actual dependent variable in their simulations, even though they simulate 10,000 different datasets, the dependent variable is a constant (see Grant and Lebo's Tables E.11, E.12, and E.13). If one (or more) series do not vary across simulations, valid inferences cannot be made. Instead of mixing simulated data with a particular variable that is constant across all simulations, the standard approach is to simulate all variables. When we generated our own data, simulating series to mimic the data in Kelly and Enns, different conclusions emerged.

Specifically, we generated 10,000 datasets designed to mimic the time series properties Grant and Lebo attribute to Kelly and Enns' data. Our focus is Kelly and Enns' most parsimonious model that examines the relationship between inequality, policy liberalism (i.e., an annual measure of liberal/conservativeness of policy outputs), and policy mood. The dependent variable is bounded between 49 and 71 and the variance is set to σ=2 to match the range and variance of mood (although the potential range of mood is clearly much greater than 49 and 71). Grant and Lebo estimate that inequality is fractionally integrated (d=0.83) and that policy liberalism is an explosive process (d=1.44). As we saw above, estimates of d can be highly sensitive to the specific choices made. However, to test how these processes might influence Kelly and Enns' results, we generated our independent variables to follow these time series processes. All variables were generated to be independent of each other. For each of the 10,000 simulated data sets we estimated a GECM (See the next subsection for the R code). If we found evidence of a unit root in the outcome variable, we then evaluated whether the GECM showed evidence of cointegration (with the correct MacKinnon critical value). We find incorrect evidence of cointegration in just 2.8 percent of simulations. Because we would not interpret the results of the GECM unless we found evidence of a unit root in the outcome variable and cointegration with the predictors, setting aside the other results (as Grant and Lebo recommend) would ensure an appropriate Type I error rate. These results not only support Kelly and Enns' use of the GECM to analyze policy mood but they reinforce a growing body of literature that finds that the mass public has not increased support for redistribution as inequality has risen (e.g., Ashok, Kuziemko and Washington 2015).

Other concerns also exist with Grant and Lebo's simulations. For example, it appears that they consistently did *not* test for cointegration prior to reporting the rate of spurious relationships between predictors and the dependent variable. Consider Column 1 of their Table E.11, which shows that the GECM incorrectly found evidence of cointegration (with the correct MacKinnon critical values) in 6.9 percent of simulations. Yet, Grant and Lebo report that the relationship between X_{t-1} and Y was significant in 21 percent of simulations. This result can only emerge if Grant and Lebo ignore their own advice to "set aside the estimates" when there is no cointegration. Another concern emerges in Grant and Lebo's Table E.12 where they are interested in the performance of the Prais-Winsten estimator, which they report in Column 1. It appears they relied on the Prais-Winsten estimator for all simulations in this column, which yields an uninformative result. What researchers need to know is how well (or poorly) does the Prais-Winsten estimator perform when there is evidence of serial correlation (i.e., when an applied researcher might consider using this estimator).

In sum, Grant and Lebo's decision to keep the dependent variable constant across all simulated data sets, their failure to follow their own advice for the GECM when reporting the results of their simulations, and their decision to evaluate the Prais-Winsten estimator without testing whether the Prais-Winsten estimator was appropriate raises serious concerns about the inferences they draw from their simulations.

Appendix 3.1 R Code to Replicate the previous result (2.8 %)

The R code below generates 10,000 datasets where the dependent variable follows a bounded unit root process that mimics policy mood and the independent variables mimic policy liberalism and income inequality (the two independent variables in Kelly and Enns (2010)). The code then tests the proportion of simulations where we incorrectly find evidence of cointegration and of a relationship between X_{t-1} and Y.

```
library(data.table)
library(DataCombine)
library(foreign)
library(fracdiff)
library(parallel)
library(portes)
library(urca)
library(dplyr)
library(tseries)
# remove (almost) everything in the working environment, such as local macro variables.
remove(list = ls())

set.seed(987654321)

# define a that estimates our models and saves results. We call it "model". It’s a function of the variables, n.
model <- function(n){ #

n <- 54
t <- seq(1:n) #generate a sequence of {0,1,2,...,n}

# Set values for bounds
#See Grant and Lebo Table I.1 and Grant and Lebo replication code
#bounds of mood, 49 to 71
a <- 0.5
b <- 0.5
tau <- 59
k <- 5.49998

e<-rnorm(n,0,2) #generate a random variable e which has a normal distribution with mean=0; variance=2

# Generating starting values for DV, bound b/t 1 and 100
dv1 <- c(1:n,NA) #generate a vector of 1 by n (dv)
```

```
dv1[1] <- runif(1, min=50, max=59)
    for(i in 2:n){ #for values 2 through n of dv1, replace with bounded unit root
   dv1[i] <- dv1[i-1] + exp(-k)*(exp((-a)*(dv1[i-1] - tau)) - (exp((b)*(dv1[i-1] - tau)))) + e[i]
    }

#Grant and Lebo estimate d=1.44 for policy liberalism and d=.83 for inequality

    #generate fractionally integrated series
    #because fracdiff.sim only generated -.5<d<.5, generate (d-1) and then add 1
    d <- fracdiff.sim(n, d = .44) #
    dpl <- as.ts(d$series) #save fractionally integrated series y_frac as a time series
    xpl <- c(1:n,NA) #generate a vector of 1 by n with all missing values (NA)
    xpl[1]<-dpl[1] #replace the first element of xpl with the first element of d44
    for(i in 2:n){ #for values 2 through n of xpl, replace with lagged value + FI series to produce d=1.44
   xpl[i] <- xpl[i - 1] + dpl[i]
    }

    d <- fracdiff.sim(n, d = -.17) #
    din <- as.ts(d$series) #save fractionally integrated series y_frac as a time series
    xin <- c(1:n,NA) #generate a vector of 1 by n (y_d62) with all missing values (NA)
    xin[1]<-din[1] #replace the first element of xin with the first element of din
    for(i in 2:n){ #for values 2 through n of xpl, replace with lagged value + FI series to produce d=.83
   xin[i] <- xin[i - 1] + din[i]
    }

  dv1 <- dv1[1:n]
  xpl <- xpl[1:n]
  xin <- xin[1:n]
    data <- data.frame(cbind(t,dv1,xpl,xin)) #combine all time series that we have generated so far.


    write.dta(data, file = "data.dta") #export data in the DTA format
```

```
data$lxpl <- lag(data$xpl) #generate lagged value
data$lxin <- lag(data$xin) #generate lagged value
data$ldv1 <- lag(data$dv1) #generate lagged value

data$d_xpl <- data$xpl - data$lxpl #generate first-differenced
data$d_xin <- data$xin - data$lxin #generate first-differenced
data$d_dv1 <- data$dv1 - data$ldv1 #generate first-differenced

#test for a unit root in dv1
mlag <- as.integer(12*(n/100)^(1/4)) #Use Schwert’s (1987) criterion for the max n of lag length
adf_test <- ur.df(data$dv1, type="drift", lags=mlag, selectlag="AIC")
adf_dv_s <- summary(adf_test)
df_dv_tstat <- adf_dv_s@teststat[1,1]
df_dv_cv <- adf_dv_s@cval[1,2]

#estimate the GECM
ecm <- lm(d_dv1~ldv1 + d_xin + lxin + d_xpl + lxpl, data=data)
beta_ldv1<- summary(ecm)$coefficients[2,1] #save coefficient for ldv1
beta_d_xin<- summary(ecm)$coefficients[3,1] #save coefficient for d_xin
beta_lxin<- summary(ecm)$coefficients[4,1]
beta_d_xpl<- summary(ecm)$coefficients[5,1]
beta_lxpl<- summary(ecm)$coefficients[6,1]

se_ldv1<- summary(ecm)$coefficients[2,2] #save std. error for the coefficient
se_d_xin<- summary(ecm)$coefficients[3,2] #save std. error for the coefficient
se_lxin<- summary(ecm)$coefficients[4,2]
se_d_xpl<- summary(ecm)$coefficients[5,2]
se_lxpl<- summary(ecm)$coefficients[6,2]
tstat_ldv1<-beta_ldv1/se_ldv1 #save t statistic of the coefficient
tstat_d_xin<-beta_d_xin/se_d_xin
tstat_lxin<-beta_lxin/se_lxin
```

```
    tstat_d_xpl<-beta_d_xpl/se_d_xpl
    tstat_lxpl<-beta_lxpl/se_lxpl


#save all the results and parameters in results file.
    results<-cbind(n,tstat_ldv1,tstat_lxin,tstat_lxpl,df_dv_tstat,df_dv_cv)
    return(results)
}

#Run simulations
n_list<-c(54) #define the set of values n takes (n=number of T for each series).
nobs <- length(n_list) #save the length of n_list, which is 1 as we only have one element in n_list.
numparams<- 7 #set the number of columns in the output to which results are saved for each iteration
reps <- 1000 #set the number of iteration
d <- data.frame(matrix(, nrow = nobs*reps, ncol = numparams)) #d is the matrix that we save our results to.
i <- 0 #define i so that we can use it to save results to d[i,] for each iteration
output = NULL
for(n in n_list) {
            for(i3 in 1:reps) {
                i <- i + 1
                d[i,] <- cbind(i3,model(n)) #for each iteration i3, we save the results computed from model(n).
            }
        }
colnames(d) = c("iterations","n","tstat_ldv1","tstat_lxin","tstat_lxpl","df_dv_stat","df_dv_cv")
write.dta(d, file = "KellyEnns_sim_outputs.dta")
attach(d)
#Cointegration w/ EC test if evidence that DV contains a unit root
100*sum(tstat_ldv1 < -3.570 & df_dv_stat<df_dv_cv)/reps
#spurious regression on "inequality" variable
100*sum((tstat_lxin < -1.96 & tstat_ldv1 < -3.570) | (tstat_lxin > 1.96 & tstat_ldv1 < -3.570))/reps
```

Appendix 4   Replication Code for Table 3

The data necessary to replicate the following analyses comes from Casillas, Enns and Wohlfarth (2011) and can be found at `https://dataverse.harvard.edu/dataset.xhtml?persistentId=hdl:1902.1/14568`. The data file is `po_sc.dta`.

The following `Stata` code replicates the results reported in Table 3 and compares the model fit of Grant and Lebo's ARFIMA model and the ARIMA models we report.

```
arfima d.all_rev
estimates stats
predict filtered, r
corrgram filtered
drop filtered

arima all_rev, arima(2,1,0)
estimates stats
predict filtered, r
corrgram filtered
drop filtered

arfima d.nosal_rev
estimates stats
predict filtered, r
corrgram filtered
drop filtered

arima nosal_rev, arima(0,1,1)
estimates stats
predict filtered, r
corrgram filtered
drop filtered
```

Appendix 5   Semi–Parametric Estimates of d

In the text, we indicated that despite critiquing Keele, Linn, and Webb for using Stata's Exact Maximum Likelihood (EML) to estimate the fractional integration parameter d, Grant and Lebo report results from this same estimator. In Table 2, we present the estimates for d based on three semi–parametric estimators, the Geweke/Porter–Hudak log periodogram estimator (Geweke and Porter-Hudak 1983), the Phillips modified log periodogram estimator (Phillips 1999*a*, Phillips 1999*b*), and the Robinson log periodogram estimator (Robinson 1995).[1] As Box-Steffensmeier, Freeman, Hitt and Pevehouse (2014, 178)

[1]All estimates were performed in Stata using the default settings (Baum and Wiggins 2000). Table 2 in the Appendix reports the same estimates with standard errors. The average estimated standard error is 0.27, with a range from 0.13 to 0.45. These values further highlight the uncertainty around estimates of d.

note, "semi-parametric methods…are popular because they do not require researchers to make strong assumptions about the short–term components [i.e., the autoregressive and moving average components] of the model." Because the estimates are sensitive to the number of ordinates specified, for the Geweke/Porter–Hudak and Phillips estimators, we report commonly employed powers that range from 0.50 to 0.75 (see Column 1) (Baum and Wiggins 2000).[2] Because Robinson's estimator can include powers that include a sizeable fraction of the sample size, we extend the power to 0.95 (Baum and Wiggins 2000). We follow Grant and Lebo and first difference our series prior to estimating d and then add 1 to the estimate.

Table 2: Semi–Parametric Estimates of d

	All Reversals						Nonsalient Reversals					
	Geweke/ Porter-Hudak		Phillips Modified		Robinson		Geweke/ Porter-Hudak		Phillips Modified		Robinson	
Power	d	(s.e.)	d	(s.e.)	d	(s.e.)	d	(s.e.)	d	(s.e.)	d	(s.e.)
0.50	1.41	(0.19)	1.08	(0.33)	1.05	(0.33)	1.33	(0.24)	1.49	(0.23)	1.00	(0.34)
0.55	1.00	(0.28)	0.80	(0.29)	1.21	(0.30)	0.99	(0.28)	1.07	(0.30)	1.42	(0.44)
0.60	1.21	(0.30)	1.01	(0.31)	1.21	(0.30)	1.42	(0.45)	1.36	(0.36)	1.42	(0.44)
0.65	1.02	(0.26)	0.87	(0.26)	1.02	(0.25)	1.14	(0.39)	1.11	(0.32)	1.13	(0.38)
0.70	0.88	(0.26)	0.75	(0.24)	0.85	(0.22)	0.91	(0.32)	0.91	(0.27)	0.90	(0.27)
0.75	0.73	(0.24)	0.62	(0.23)	0.73	(0.22)	0.76	(0.28)	0.76	(0.25)	0.75	(0.26)
0.80					0.74	(0.19)					0.74	(0.20)
0.85					0.76	(0.16)					0.73	(0.17)
0.90					0.68	(0.14)					0.70	(0.14)
0.95					0.82	(0.13)					0.84	(0.15)

Note: Estimates of d based on three log periodogram estimators. Standard errors (s.e.) in parentheses. Stata was used for all estimates.

In Table 2, across all estimators, for many powers we see evidence that $d = 1$ or close to 1. This result is consistent with the ARIMA models reported in the text, which suggested that the series contain a unit root. Nevertheless, it is also clear that the estimates of fractional integration are highly sensitive to the estimator used and to how the models are specified. This is an important consideration that Grant and Lebo did not discuss. When testing series for fractional integration, researchers must remember to report the choices they make and whether the results are sensitive to these decisions.

The results reported above in Table 2 were conducted in Stata. To replicate these results the `gphudak`, `modlpr`, and `roblpr` packages, which can be downloaded from `http://fmwww.bc.edu/RePEc/bocode/g/`, `http://fmwww.bc.edu/RePEc/bocode/m`, and `http://fmwww.bc.edu/RePEc/bocode/r`. The Stata search feature can also be used to download these

[2]Consistent with Baum and Wiggins (2000), Grant (2015, 13) writes, "when estimating the long memory of a series with no a priori information, it is best to begin with a conservative bandwidth, $m = T^{0.5}$ or $T^{0.65}$."

files, but they must be downloaded separately (not as a package) to work correctly. After downloading these packages, the following replication code can be used.

```
gphudak d.all_rev, powers(0.5 0.55:0.75)
modlpr d.all_rev, powers(0.5 0.55:0.75)
roblpr d.all_rev, powers(0.5 0.55:.95)

gphudak d.nosal_rev, powers(0.5 0.55:0.75)
modlpr d.nosal_rev, powers(0.5 0.55:0.75)
roblpr d.nosal_rev, powers(0.5 0.55:.95)
```

References

Ashok, Vivekinan, Ilyana Kuziemko and Ebony Washington. 2015. "Support for Redistribution in an Age of Rising Inequality: New stylized facts and some tentative explanations." *Brookings Papers on Economic Activity* .

Baum, Christopher F. and Vince Wiggins. 2000. "Tests for Long Memory in a Time Series." *Stata Technical Bulletin* 57(Sep.):39–44.

Box-Steffensmeier, Janet M., John R. Freeman, Matthew P. Hitt and Jon C.W. Pevehouse. 2014. *Time Series Analysis for the Social Sciences*. New York: Cambridge University Press.

Casillas, Christopher J., Peter K. Enns and Patrick C. Wohlfarth. 2011. "How Public Opinion Constrains the U.S. Supreme Court." *American Journal of Political Science* 55(1):74–88.

Geweke, John and Susan Porter-Hudak. 1983. "The Estimation and Application of Long Memory Time Series." *Journal of Time Series Analysis* 4(4):221–238.

Grant, Taylor. 2015. "Fractional Integration in Short Samples: Parametric Versus Semiparametric Methods." Unpublished Manuscript.

Kelly, Nathan J. and Peter K. Enns. 2010. "Inequality and the Dynamics of Public Opinion: The Self-Reinforcing Link Between Economic Inequality and Mass Preferences." *American Journal of Political Science* 54(4):855–870.

Phillips, Peter C.B. 1999*a*. "Discrete Fourier Transforms of Fractional Processes." *Unpublished Working Paper No. 1243, Cowles Foundation for Research in Economics, Yale University* 1243(`http://cowles.econ.yale.edu/P/cd/d12a/d1243.pdf`).

Phillips, Peter C.B. 1999*b*. "Unit Root Log Periodogram Regression." *Unpublished Working Paper No. 1243, Cowles Foundation for Research in Economics, Yale University* 1244(`http://cowles.econ.yale.edu/P/cd/d12a/d1244.pdf`).

Robinson, P.M. 1995. "Gaussian Semiparametric Estimation of Long Range Dependence." *Annals of Statistics* 23(5):1630–1661.