EC771: Econometrics, Spring 2010

Greene, Econometric Analysis (6th ed, 2008)

Chapter 17: Simulation Based Estimation and Inference

We often want to evaluate the properties of estimators, or compare a proposed estimator to another, in a context where analytical derivation of those properties is not feasible. In that case, econometricians resort to **Monte Carlo studies**: simulation methods making use of (pseudo-)random draws from an error distribution and multiple replications over a set of known parameters. This methodology is particularly relevant in situations where the only analytical findings involve asymptotic, large–sample results. Applied researchers need to understand how a particular estimation strategy will perform in small samples: for instance, when working with macro data on the national aggregates, we have no more than 150–200 quarterly observations available for many series. Where only annual data are available, the problem becomes even more striking. In that case, we require an understanding of the performance of estimation techniques, test statistics, etc. in a very small sample. Monte Carlo studies, although they do not generalize to cases beyond those performed in the experiment, may be useful in these situations. They also are useful in modelling quantities for which no analytical results have yet been derived: for instance, the critical values for many unit-root test statistics have been derived by simulation experiments, in the absence of closed-form expressions for the sampling distributions of the statistics.

Most econometric software provide some facilities for Monte Carlo experiments. Although one can write the code to generate an experiment in any programming language, it is most useful to do so in a context where one may readily save the results of each replication for further analysis. The quality of the pseudo-random number generators available is also an important concern. Recent studies published in the *Journal of Applied Econometrics* have compared many software packages' performance on a standard set of benchmarks for randomness. Although most packages meet these criteria, all but the most recent versions of GAUSS fail miserably—casting considerable doubt on those many published studies making use of GAUSS software. State-of-the-art pseudo-random number generators do exist, and you should use a package that implements them. You will also want a package with a full set of statistical functions, permitting random draws to be readily made from a specified distribution-not merely normal or t, but from a number of additional distributions, depending upon the experiment.

Stata version 11 provides a useful environment for Monte Carlo simulations. Setting up a simulation requires that you write a Stata program: not merely a “do-file” containing a set of Stata commands, but a sequence of commands beginning with the `program define` statement. This program sets up the simulation experiment and specifies what is to be done in one replication; you then invoke it with the `simulate` prefix to execute a specified number of replications.

These notes on Monte Carlo siimulation are adapted from Cameron and Trivedi, *Microeconomics Using Stata*, 2009.

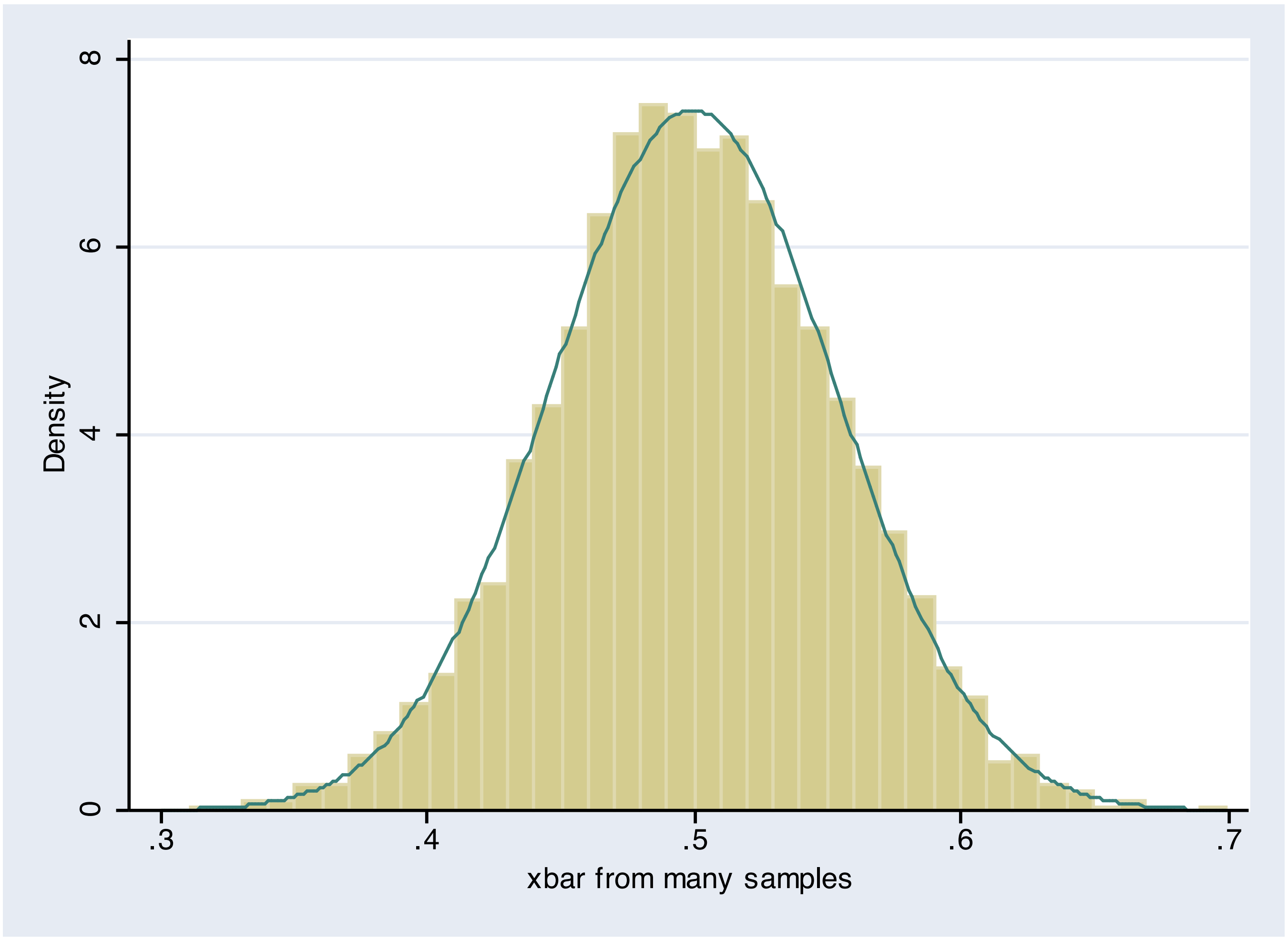
We first consider a very simple program in which we demonstrate the central limit their result that sample mean is approximately normally distributed as $N \rightarrow \infty$. We consider a random variable that has the uniform distribution and a sample size of 30. The `simulate` command runs a specified command a number of times, where the command will often be a user-written program. A number of expressions are returned by the command, and saved in a new Stata dataset by `simulate`. As an example:

```
// Program to draw sample of size 30 from uniform
// and return sample mean
program onesample, rclass
    version 11
    drop _all
    quietly set obs 30
    generate x = runiform()
    summarize x
    return scalar meanforonesample = r(mean)
end
```

We can run the program with `simulate`, returning the one result as `xbar`:

```
.
. * Program to draw 1 sample of size 30 from uniform ///
>   and return sample mean
. program onesample, rclass
  1.     version 11
  2.     drop _all
  3.     quietly set obs 30
  4.     generate x = runiform()
  5.     summarize x
  6.     return scalar meanforonesample = r(mean)
  7. end
.
. * Run program onesample once as a check
. set seed 10101
. onesample
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
           x |        30    .5459987    .2803788   .0524637   .9983786
. return list
scalars:
   r(meanforonesample) =  .5459987225631873
.
. * Run program onesample 10,000 times to get 10,000 sample means
. simulate xbar = r(meanforonesample), seed(10101) ///
> reps(10000) nodots: onesample
      command:  onesample
         xbar:  r(meanforonesample)
.
. * Summarize the 10,000 sample means and draw histogram
. summarize xbar
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        xbar |     10000    .4995835    .0533809   .3008736   .6990562
. histogram xbar, normal xtitle("xbar from many samples")
(bin=40, start=.30087364, width=.00995456)
.
. quietly graph export mus04fig1clt2.pdf, replace
.
```

The `set seed` ensures that the same sequence of pseudo-random numbers will be used every time the simulation is run. This is useful when debugging the program to ensure that it is coded properly. The results of `simulate` can then be graphed:



As a more interesting example of Monte Carlo simulation, let us consider simulation methods to investigate the finite-sample properties of the OLS estimator with random regressors and skewed errors. If errors are $i.i.d.$, skewness will have no effect on the large-sample properties of the OLS estimator. But with skewed errors, we will need a larger sample size for the asymptotic distribution to approximate the finite-sample distribution of the OLS estimator than when errors are normal.

We consider the DGP

$$y = \beta_1 + \beta_2 x + u,\ u \sim \chi^2(1) - 1,\ x \sim \chi^2(1)$$

where $\beta_1 = 1$, $\beta_2 = 2$, $N = 150$. The error is independent of x, ensuring consistency of OLS, with a mean of zero, variance of 2, skewness of $\sqrt{8}$ and kurtosis of 15, compared to the Normal error, with a skewness of 0 and kurtosis of 3.

For each simulation, we obtain parameter estimates, standard errors, t-values for the test that $\beta_2 = 2$ and the outcome of a two-tailed test of that hypothesis at the 0.05 level.

We store the sample size and the number of simulations in global macros, as we often may want to change them.

Our simulation program becomes

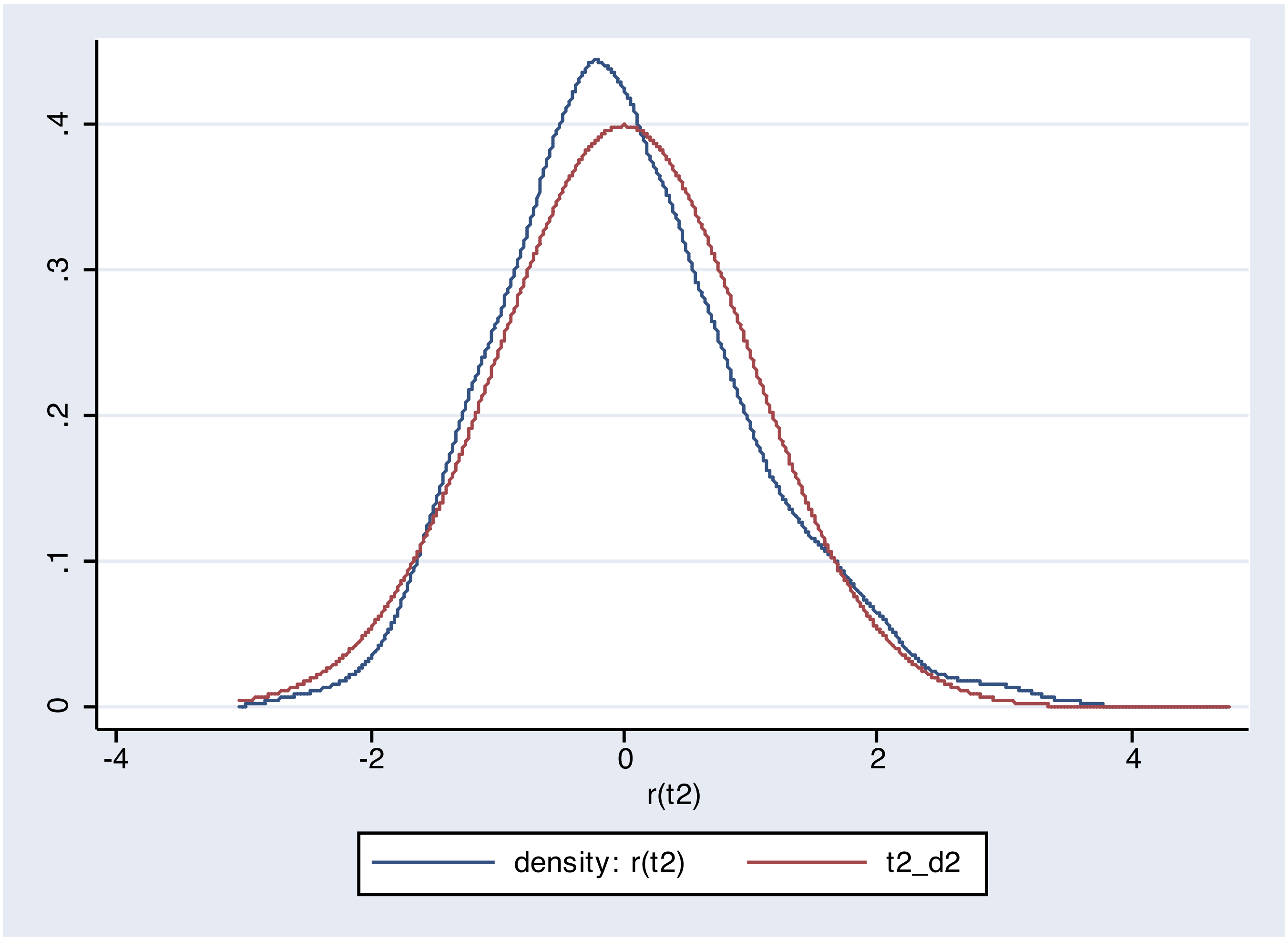
```
* Program for finite-sample properties of OLS
program chi2data, rclass
    version 11
    drop _all
    set obs $numobs
    generate double x = rchi2(1)
// demeaned chi^2 error
    generate y = 1 + 2*x + rchi2(1)-1
    regress y x
    return scalar b2 =_b[x]
    return scalar se2 = _se[x]
    return scalar t2 = (_b[x]-2)/_se[x]
    return scalar r2 = abs(return(t2))> ///
                       invttail($numobs-2,.025)
    return scalar p2 = 2*ttail($numobs-2, ///
                       abs(return(t2)))
end
```

We can now run this program with `simulate`, producing a dataset containing the five scalars listed above for each simulation:

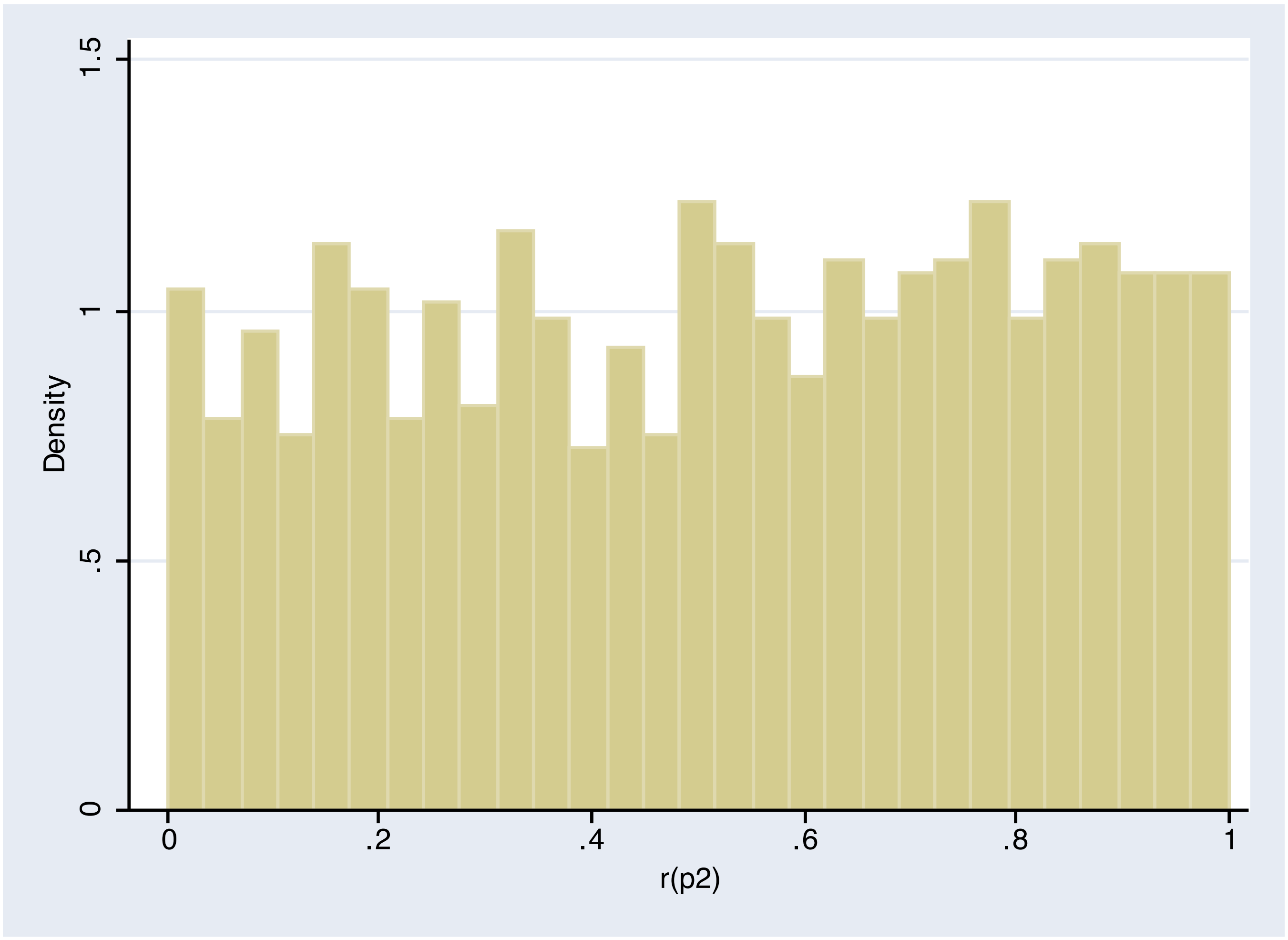
```
.
. * Define global macros for sample size and number of simulations
. global numobs 150              // sample size N
. global numsims "1000"          // number of simulations
.
. * Program for finite-sample properties of OLS
. program chi2data, rclass
  1.     version 11
  2.     drop _all
  3.     set obs $numobs
  4.     generate double x = rchi2(1)
  5.     generate y = 1 + 2*x + rchi2(1)-1     // demeaned chi^2 error
  6.     regress y x
  7.     return scalar b2 =_b[x]
  8.     return scalar se2 = _se[x]
  9.     return scalar t2 = (_b[x]-2)/_se[x]
 10.     return scalar r2 = abs(return(t2))>invttail($numobs-2,.025)
 11.     return scalar p2 = 2*ttail($numobs-2,abs(return(t2)))
 12. end
.
. set seed 10101
. * Simulation for finite-sample properties of OLS
. simulate b2f=r(b2) se2f=r(se2) t2f=r(t2) ///
> reject2f=r(r2) p2f=r(p2), reps($numsims) ///
> saving(chi2datares, replace) nolegend nodots: chi2data
.
. summarize b2f se2f reject2f
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         b2f |      1000    2.000506      .08427   1.719513    2.40565
        se2f |      1000    .0839776    .0172588   .0415919    .145264
    reject2f |      1000        .046    .2095899          0          1
. * Summarize results
. mean b2f se2f reject2f
Mean estimation                     Number of obs    =    1000
--------------------------------------------------------------
             |       Mean   Std. Err.     [95% Conf. Interval]
-------------+------------------------------------------------
         b2f |   2.000506   .0026649      1.995277    2.005735
        se2f |   .0839776   .0005458      .0829066    .0850486
    reject2f |       .046   .0066278       .032994     .059006
--------------------------------------------------------------
.
. // histogram t2f
. histogram p2f
(bin=29, start=.0000108, width=.0344747)
```

```
. graph export mus04p2test.pdf, replace
(file /Users/baum/Documents/Courses 2009-2010/EC771 S2010/mus04p2test.pdf
> en in PDF format)
.
. * t-statistic distribution
. kdensity t2f,  n(1000) gen(t2_x t2_d) nograph
. generate double t2_d2 = tden(148, t2_x)
. graph twoway (line t2_d t2_x) (line t2_d2 t2_x)
.
. graph export mus04ttest.pdf, replace
(file /Users/baum/Documents/Courses 2009-2010/EC771 S2010/mus04ttest.pdf w
> n in PDF format)
```

We can conclude that our estimator of β_2 is unbiased, as the quite narrow 95% confidence interval from 1000 simulations contains the true value of 2.0. We can consider how closely the distribution of t-statistics from the program approximate the asymptotic distribution of a t_{148}:



To evaluate the *size* of the test (the probability of rejecting a true null hypothesis), we can examine the rejection rate, `r2` above. The estimated rejection rate from 1000 simulations is 0.046, with a 95% confidence interval of (0.033, 0.059): wide, but containing 0.05. With 10,000 replications, the estimated rejection rate is 0.049 with a confidence interval of (0.044, 0.052). The last item computed is the p-value of the test. If the t-distribution is the correct distribution, then $p2$ should be uniformly distributed on (0,1).



We can also evaluate the *power* of the test: its ability to reject a false null hypothesis. We estimate the rejection rate for the test against a false null hypothesis. The larger the difference between the tested value and the true value, the greater the power and the rejection rate. This modified version of the `chi2data` program estimates the power of a test against the false null hypothesis $\beta_2 = 2.1$.

```
. * Program for finite-sample properties of OLS: power
. program chi2datab, rclass
  1.      version 11
  2.      drop _all
  3.      set obs $numobs
  4.      generate double x = rchi2(1)
  5.      generate y = 1 + 2*x + rchi2(1)-1     // demeaned chi^2 error
  6.      regress y x
  7.      return scalar b2  =_b[x]
  8.      return scalar se2 =_se[x]
  9.      test x=2.1
 10.      return scalar r2 = (r(p)<.05)
 11. end
.
. * Power simulation for finite-sample properties of OLS
. simulate b2f=r(b2) se2f=r(se2) reject2f=r(r2),  ///
> reps($numsims) saving(chi2databres, replace) ///
> nolegend nodots: chi2datab
.
. mean b2f se2f reject2f
Mean estimation                     Number of obs    =    1000
--------------------------------------------------------------
             |       Mean   Std. Err.     [95% Conf. Interval]
-------------+------------------------------------------------
         b2f |   2.001852   .0026976      1.996559    2.007146
        se2f |   .0836442   .0005591      .0825471    .0847414
    reject2f |       .241   .0135315      .2144465    .2675535
--------------------------------------------------------------
.
```

In this case, the power is not high, with a mean of 0.241. Using a larger sample size or increasing the distance between the true and false values would increase the power of the test (see `771mcsim2.html`).

Simulating a spurious regression

We can demonstrate Granger's concept of a *spurious regression* with a simulation. We create two independent random walks, regress one on the other, and record the coefficient, standard error, t-ratio and its tail probability in the returns from the program:

```
* spurious regression: independent random walks
prog irwd, rclass
version 11
drop _all
set obs $N
local drift 2
g double x = 0 in 1
g double y = 0 in 1
replace x = x[_n - 1] + $trcoef * `drift´ ///
                   + rnormal() in 2/l
```

```
replace y = y[_n - 1] + $trcoef * `drift´ ///
                   + rnormal() in 2/l
reg y x
return scalar b = _b[x]
return scalar se = _se[x]
return scalar t = _b[x]/_se[x]
return scalar r2 = ///
abs(return(t)) > invttail($N - 2, 0.025)
end
```

We use a global, **`trcoef`**, to allow the program to be used for both pure random walks and random walks with drift.

```
. capt prog drop _all
. * spurious regression: independent random walks
. prog irwd, rclass
  1.   version 11
  2.   drop _all
  3.   set obs $N
  4.   local drift 2
  5.   g double x = 0 in 1
  6.   g double y = 0 in 1
  7.   replace x = x[_n - 1] + $trcoef * `drift´ + rnormal() ///
>   in 2/l
  8.   replace y = y[_n - 1] + $trcoef * `drift´ + rnormal() ///
>   in 2/l
  9.   reg y x
 10.   return scalar b = _b[x]
 11.   return scalar se = _se[x]
 12.   return scalar t = _b[x]/_se[x]
 13.   return scalar r2 = abs(return(t)) > invttail($N - 2, 0.025)
 14. end
.
. global N 100
. global nsim 10000
. set seed 1010101
.
. // consider IRWs with no drift
. global trcoef 0
. simulate birwd=r(b) sirwd=r(se) tirwd=r(t) rejirwd=r(r2), ///
>  reps($nsim) nodots saving(irw0, replace): irwd
      command:  irwd
        birwd:  r(b)
        sirwd:  r(se)
        tirwd:  r(t)
      rejirwd:  r(r2)
. su
    Variable |       Obs        Mean    Std. Dev.       Min        Ma
-------------+----------------------------------------------------------
       birwd |     10000    .0003267    .6336891  -3.682241   3.30113
       sirwd |     10000     .100645    .0649186   .0116683   .689878
       tirwd |     10000   -.0225758    7.390102  -34.39042    35.837
     rejirwd |     10000       .7572    .4287966          0
. l in 1/20

     +---------------------------------------------+
     |    birwd      sirwd        tirwd   rejirwd |
     |---------------------------------------------|
  1. |  .4797036   .0366694    13.08186         1 |
  2. | -.3215804    .126415   -2.543845          1 |
  3. |  .5518465   .0509739    10.82607          1 |
  4. | -.9704604   .0774199   -12.53502          1 |
  5. |  .9274789   .0991612    9.353246          1 |
     |-------------------------------------------|
  6. | -.7858061   .0432042   -18.18819          1 |
  7. | -.5938631   .0538909   -11.01972          1 |
  8. |  .6821204   .1127544    6.049611          1 |
  9. |  .5334677   .0528825     10.0878          1 |
 10. |  .0958101   .0618102    1.550069          0 |
     |-------------------------------------------|
 11. | -.3524039   .0844574   -4.172564          1 |
 12. |   .251032   .1512837    1.659346          0 |
 13. |  .8574678   .1752042    4.894105          1 |
 14. | -.3218689   .0791635   -4.065877          1 |
 15. |  .2293266   .0713637     3.21349          1 |
     |-------------------------------------------|
 16. | -.3609286   .1122861   -3.214365          1 |
 17. | -.5108765   .1038393   -4.919877          1 |
 18. | -.0265709   .0448767   -.5920879          0 |
 19. | -1.321453   .0955171   -13.83473          1 |
 20. | -1.061208   .0308577   -34.39042          1 |
     +-------------------------------------------+

.
. // consider IRWs with drift
. global trcoef 1
. simulate birwd=r(b) sirwd=r(se) tirwd=r(t) rejirwd=r(r2), ///
>  reps($nsim) nodots saving(irw1, replace): irwd
      command:  irwd
        birwd:  r(b)
        sirwd:  r(se)
        tirwd:  r(t)
      rejirwd:  r(r2)
. su
    Variable |       Obs        Mean    Std. Dev.       Min        Ma
-------------+------------------------------------------------------------
       birwd |     10000    1.001838    .0777357   .7386733   1.31321
       sirwd |     10000    .0061999    .0019246   .0020809    .01884
       tirwd |     10000    175.4512    49.80395   60.38373   473.091
     rejirwd |     10000           1           0          1
. l in 1/20
     +-----------------------------------------+
     |    birwd      sirwd      tirwd   rejirwd |
     |-----------------------------------------|
  1. | 1.002217   .0063258   158.4322         1 |
  2. | 1.103606   .0050269   219.5395         1 |
  3. | 1.083013   .0069692    155.399         1 |
  4. | .9581092   .0051847   184.7967         1 |
  5. | 1.085191   .0053837   201.5711         1 |
     |-----------------------------------------|
  6. | .9154854   .0059151   154.7701         1 |
  7. | 1.065713    .008413   126.6743         1 |
  8. | 1.042774     .00848   122.9693         1 |
  9. | 1.104847   .0062487   176.8136         1 |
 10. | .9713724   .0048474   200.3911         1 |
     |-----------------------------------------|
 11. | .8808553   .0062393   141.1783         1 |
 12. |  1.08984   .0076755   141.9889         1 |
 13. | 1.003118   .0056677   176.9884         1 |
 14. | .9593877    .003884   247.0105         1 |
 15. | .9122529   .0037094   245.9296         1 |
     |-----------------------------------------|
 16. | .9130253   .0113659   80.33036         1 |
 17. | .9648722   .0040714    236.988         1 |
 18. | .9089801   .0085264   106.6083         1 |
 19. | 1.079193   .0054492    198.045         1 |
 20. | .9045779   .0067149   134.7114         1 |
     +-----------------------------------------+

.
```

For pure random walks with $N = 100$, the true null hypothesis that $\partial y/\partial x = 0$ is rejected in over 75% of 10,000 simulations. For random walks with drift, the null is rejected in every simulation.

Bootstrapping

A closely related topic to Monte Carlo simulation is that of the technique of **bootstrapping,** developed by Efron (1979). A key difference: whereas Monte Carlo simulation is designed to utilize purely random draws from a specified distribution (which with sufficient sample size will follow that theoretical distribution) bootstrapping is used to obtain a description of the sampling properties of empirical estimators, using the empirical distribution of sample data. If we derive an estimate θ_{obs}

from a sample $X = (x_1, x_2, ..., x_N)$, we can derive a bootstrap estimate of its precision by generating a sequence of bootstrap estimators $\left(\widehat{\theta}_1, \widehat{\theta}_2, ..., \widehat{\theta}_B\right)$, with each estimator generated from an $m-$observation sample from X, *with replacement*. The size of the bootstrap sample m may be larger, smaller or equal to N. The estimated asymptotic variance of θ may then be computed from this sequence of bootstrap estimates and the original estimator, θ_{obs} (for observed):

$$Est.Asy.Var[\theta] = B^{-1} \sum_{b=1}^{B} \left[\widehat{\theta}_b - \theta_{obs}\right] \left[\widehat{\theta}_b - \theta_{obs}\right]'$$

where the formula has been written to allow $\widehat{\theta}$ to be a vector of estimated parameters. The square roots of this variance-covariance matrix are known as the *bootstrap standard errors* of $\widehat{\theta}$. They will often prove useful when doubt exists regarding the appropriateness of the conventional estimates of the precision matrix, as well as in cases where no analytical expression for that matrix is available, e.g., in the context of a highly nonlinear estimator for which the numerical Hessian may not be computed.

After bootstrapping, we have the mean of the estimated statistic—e.g., $\widehat{\theta}$, which may be compared with the point estimate of the statistic computed from the original sample, θ_{obs} (for observed). The difference $(\widehat{\theta} - \theta_{obs})$ is an estimate of the bias of the statistic; in the presence of a biased point estimate, this bias may be nontrivial. However we cannot use that difference to construct an unbiased estimate, since the bootstrap estimate contains an indeterminate amount of random error.

Why do we bootstrap quantities for which asymptotic measures of precision exist? All measures of precision come from the statistic's sampling distribution, which is in turn determined by the distribution of the population and the formula used to estimate the statistic from a sample of size N. In some cases, analytical estimates of the sampling distribution are difficult or infeasible to compute, such as those relating to the means from non-normal populations. Bootstrapping estimates of precision rely on the notion that the observed distribution in the sample is a good approximation to the population distribution.

The `bootstrap` command specifies a single estimation command, the results to be retained from that command, and the number of bootstrap samples (B) to be drawn. You may optionally specify the size of the bootstrap samples (m); if you do not, it defaults to `_N` (the currently defined sample size). This is very useful, since it makes estimating bootstrap standard errors no more difficult than performing the estimation itself. If you are trying to construct a bootstrap distribution for a set of statistics which are forthcoming from a single Stata command, this may be done without further programming.

The `bootstrap` command is not limited to generating bootstrap estimates from a single Stata command. To compute a bootstrap distribution for more complicated quantities, you must write a Stata program (just as with the `simulate` command) that specifies the estimation to be performed in the bootstrap sample. The confidence interval is based on the assumption of approximate normality of the sampling (and bootstrap) distribution, and will be reasonable if that assumption is so. One may then execute `bootstrap`, specifying the name of your program, and the number of bootstrap samples to be drawn. For instance, if we wanted to generate a bootstrap estimate of the ratio of two means, we could not do so with a single Stata command. We could do so by writing a program that returned that ratio:

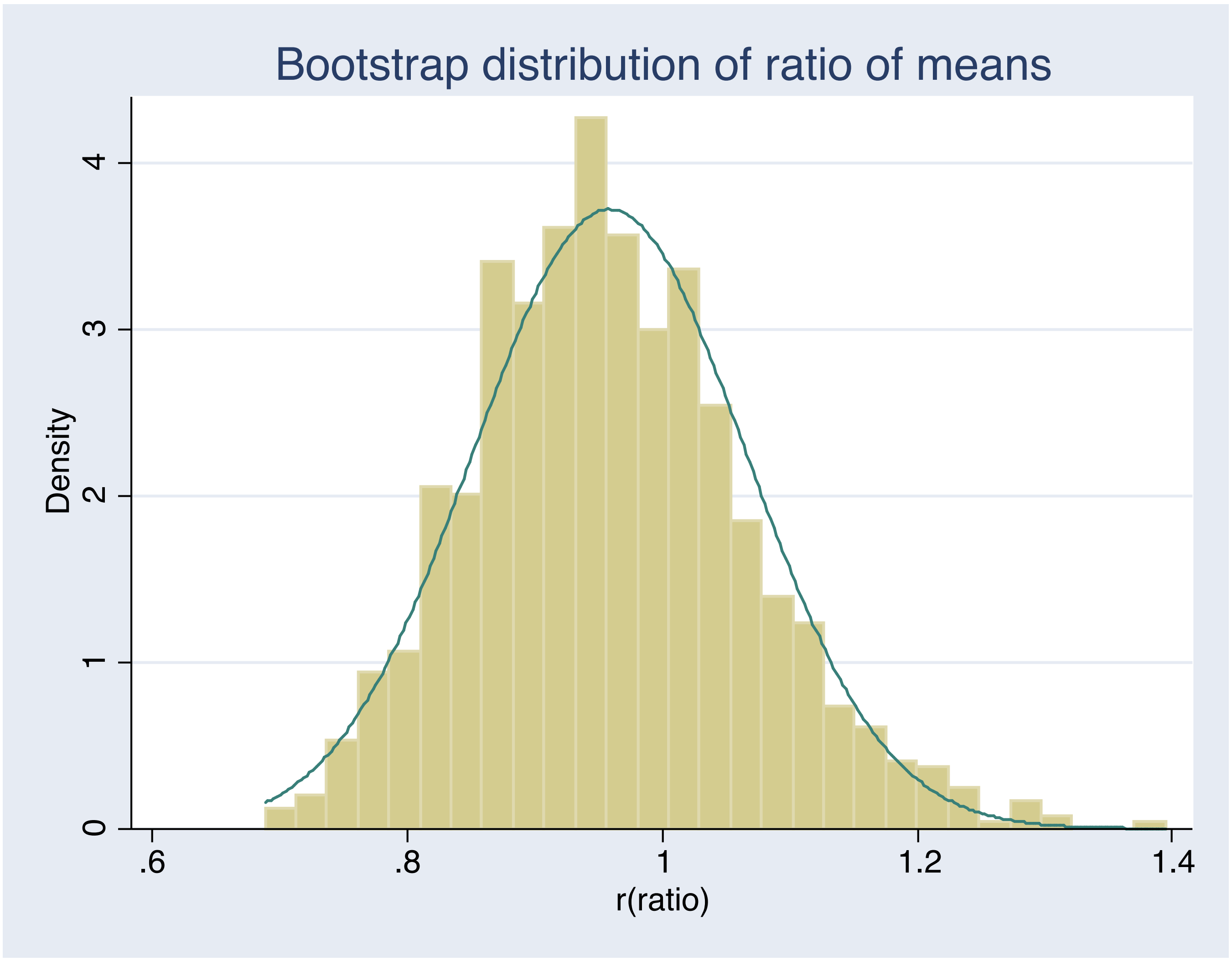
```
capture program drop muratio
program define muratio, rclass
           version 11
           syntax varlist(min=2 max=2)
           tempname ymu
           summarize `1´, meanonly
           scalar `ymu´ = r(mean)
           summarize `2´, meanonly
           return scalar ratio = `ymu´/r(mean)
end
```

We can now execute this program to compute the ratio of the average price of a domestic car vs. the average price of a foreign car, and generate a bootstrap confidence interval for the ratio.

```
.
. capture program drop muratio
. program define muratio, rclass
  1.            version 11
  2.            syntax varlist(min=2 max=2)
  3.            tempname ymu
  4.            summarize `1´, meanonly
  5.            scalar `ymu´ = r(mean)
  6.            summarize `2´, meanonly
  7.            return scalar ratio = `ymu´/r(mean)
  8. end
.
. set seed 10101
. local reps 1000
. webuse auto, clear
(1978 Automobile Data)
. tabstat price, by(foreign) stat(n mean semean)
Summary for variables: price
     by categories of: foreign (Car type)
 foreign |         N      mean  se(mean)
---------+------------------------------
Domestic |        52  6072.423  429.4911
 Foreign |        22  6384.682  558.9942
---------+------------------------------
   Total |        74  6165.257  342.8719
----------------------------------------
. g p_dom = price if foreign==0
(22 missing values generated)
. g p_for = price if foreign==1
(52 missing values generated)
. muratio p_dom p_for
. return list
scalars:
              r(ratio) =  .9510925132761489
.
. bootstrap  r(ratio), reps(`reps´) nodots ///
> saving(771bs2_9,replace): muratio p_dom p_for
Warning:  Because muratio is not an estimation command or does not set
          e(sample), bootstrap has no way to determine which observations
          used in calculating the statistics and so assumes that all
          observations are used.  This means that no observations will be
          excluded from the resampling because of missing values or other
          reasons.
          If the assumption is not true, press Break, save the data, and d
          the observations that are to be excluded.  Be sure that the data
          in memory contains only the relevant data.
Bootstrap results                                Number of obs      =
                                                 Replications       =

      command:  muratio p_dom p_for
        _bs_1:  r(ratio)
------------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Inter
-------------+----------------------------------------------------------------------
       _bs_1 |   .9510925   .1072875     8.86   0.000     .7408128    1.16
------------------------------------------------------------------------------------

. use 771bs2_9,clear
(bootstrap: muratio)
. histogram _bs_1, normal name(g771bs2_r, replace) xsize(9) ysize(7) ///
>  ti("Bootstrap distribution of ratio of means")
(bin=29, start=.68787545, width=.02437044)
.
. qui graph export 771bstrap1.pdf, replace
```



Note in the histogram that the empirical distribution is quite visibly skewed.

Combining simulation and bootstrapping

These commands are very flexible; one may combine both techniques in a single Stata program. The example in Stata's Reference Manual article on `simulate` illustrates an application where a random sample is generated, and `bootstrap` is used to generate a dataset of medians calculated by bootstrap sampling from the random sample. This procedure is called within a `simulate` program which calculates the standard deviation of these bootstrap standard errors, repeated over a number of Monte Carlo draws. The `simulate` program thus generates a point and interval estimate of the median of these simulated data, where the precision of the median estimates is derived from a bootstrapped standard error.