EC771: Econometrics, Spring 2007

Greene, Econometric Analysis (5th ed, 2003)

Appendix E:
Computation and Optimization

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 9.2 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.

For instance, let us consider simulating the performance of the estimator of sample mean, $\bar{x}$, in a context of heteroskedasticity. We could derive the analytical results for this simple experiment, but in this case let us simulate them. Take the model to be $y_i = \mu + \epsilon_i$, with $\epsilon_i \sim$

$N(0, \sigma^2)$. Let ϵ be a N(0,1) variable multiplied by a factor cz_i, where z_i varies over i. We will vary parameter c between 0.1 and 1.0 and determine its effect on the point and interval estimates of μ; as a comparison, we will compute a second random variable which is homoskedastic, with the scale factor equalling $c\bar{z}$.

Appendix 1 (also available from the course homepage) contains the log of a Stata version 9 do-file which defines the simulation experiment program and executes it over a range of c values, using actual data (the average age in each of the 50 states). Alternatively, such an experiment could be based on artificial data, generated within the experiment. The stored results are then combined into a single file, from which summary statistics may be readily computed, and graphed if desired. The `het_infl` measure contains the ratio of the standard error of the

mean from the heteroskedastic case and that for the homoskedastic case. Note that the former case involves a penalty, in terms of the mean or median of the simulations, of up to 5–6 per cent, and that the size of the penalty depends positively on c.

Note that if you run this program (or any Monte Carlo simulation exercise) more than once, you will receive slightly different results. When debugging a program, it is often useful to remove this element of randomness. This can be achieved by using Stata's `set seed n` command before any calls to the pseudo-random-number generator. Since a PRN generator will generate the same sequence of pseudo-random numbers when presented with a given seed, setting the seed will cause the sequence of PRNs to be replicable.

Calculation of power of a test

Monte Carlo simulations might be used to evaluate the power of a test. In the prior experiment, we computed point and interval estimates of the mean. Let us redo that experiment with artificial data, constructed with a known mean. In this case, we perform a conventional t-test for $H_0 : \mu = 0$, and compare the power of that test over simulations of the homoskedastic and heteroskedastic data generating process by counting the number of times that the false null is rejected. The conventional t-test is performed under the maintained hypothesis that the data are generated with a single σ^2, so we would expect to find that the test is more powerful when applied to homoskedastic data. Appendix 2 (also available from the course homepage) shows that the power (labelled $pwr1$ and $pwr2$ for the homo- and heteroskedastic cases, respectively) drops off as we increase c (which inflates the variance, and amplifies the heteroskedastic effect).

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 non-trivial. 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. For instance, Appendix 3 shows an illustration of generating bootstrap standard errors for the mean and the standard deviation of the variable `age` (the average age per state), using 1000 bootstrap samples. It also illustrates how one may generate histograms of the bootstrap distributions, and produce a single-page graph of the result.

In the output from the `bootstrap` command (technically, in the output from the `bstat` command, which is automatically invoked by `bootstrap`) the bias is presented as the difference above. The first confidence interval (labelled (N)) is based on the assumption of approximate normality of the sampling (and bootstrap) distribution, and will be reasonable if that assumption is so. The percentile (P) and bias-corrected (BC) bootstrap confidence intervals are computed without making the assumption of approximate normality, and demonstrate the sensitivity of the bootstrap estimates to that feature of the empirical distribution (for instance, those confidence intervals need not be symmetric around θ_{obs}). Note on the graph that for the estimates of the mean, the bootstrap distribution diverges to some degree from normality, with considerably more mass in the center of the distribution.

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. 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. Appendix 4 contains a program, or `ado-file`, that carries out that experiment, allowing the user to specify which variables are to be considered. In this case, we construct a ratio for the average price of a domestic car vs. the average price of a foreign car, and generate a bootstrap confidence interval for the ratio. Note in the histogram that the empirical distribution is quite visibly

skewed; this corresponds to the percentile and bias-corrected confidence intervals being wider than that derived from the assumption of approximate normality.

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 [S-Z] 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.