

Economics Department
Working Papers in Economics

Boston College

Year 2002

Facilitating Applied Economic Research
with Stata

Christopher F. Baum
Boston College,

Facilitating applied economic research with Stata

Christopher F Baum
Boston College

January 2002

Abstract

We describe the Stata software environment, and illustrate how it may be profitably employed for applied economic research. Stata stands between “point and click” statistical packages and matrix languages in terms of extensibility and ease of use, and provides web-accessible features that enhance collaborative research and instruction.

1 Introduction

Applied empirical research in economics and finance has traditionally involved the use of two types of high-level software: statistical/econometric packages, such as TSP, SAS, RATS or eViews, and matrix languages, such as MATLAB, GAUSS, S-Plus or Ox. Each type has its strengths and weaknesses. In this essay, we describe the middle ground posed by the *Stata* statistical package and its programming language, and its usefulness in rapid prototyping of sophisticated econometric routines. Stata, a rapidly evolving software environment, brings several unique features to empirical researchers, particularly for those collaborating from diverse locations, and those relying on other Stata users for assistance with the implementation of new capabilities. Although Stata is a commercial product, its evolution is a unique blend of its vendor’s efforts and the sophisticated contributions of a broad set of users, interchanged via the package’s Internet-access features.

The plan of the paper is as follows. In the next section, we discuss the fundamental design of Stata’s software environment. Section 3 describes the tools provided to Stata users for advanced software development efforts. In Section 4, we discuss the various means by which Stata is integrated with the Internet, and the implications of that integration for collaborative research and distance learning. Section 5 presents a case study of the development of a routine for the analysis of endogeneity in a regression equation, and discusses a Monte Carlo experiment evaluating the routine’s performance. Concluding remarks are offered in Section 6.

2 The Stata software environment

The Stata software environment provides a unique middle ground between statistical packages with a defined feature set and open-ended matrix languages. In the former context, a variety of econometric procedures are available, with well-formatted results, batteries of diagnostics, and predesigned graphical output. Data series may be referred to by name, and missing values and transformations such as lags and differences may be readily handled: the software does the housekeeping. But extending the context of many packages (adding estimation procedures, customizing output, or reusing the results of computed quantities) may be difficult, and almost without exception the user-provided extensions to statistical packages will require special handling. User-written procedures will not be “first-class citizens” in most statistical package environments, but will require sourcing or include statements to make them available within the package. User-written procedures often include, at best, comment statements providing on-line help. In summary, if a statistical package provides the econometric capabilities that you seek, and you are satisfied with the formatting of its output (or ability to transfer computed results to an external file), you may be quite satisfied with such an environment. In other circumstances—e.g., if you seek to utilize a very recently developed econometric procedure that has not yet been implemented as an affordable update for your package of choice by its vendors—one may find that considerable effort is required to develop and test the statistical routines necessary for an empirical research project.

Many applied researchers, and a preponderance of econometric theorists, circumvent the restrictions posed by the defined feature set of statistical packages by employing a matrix language such as MATLAB, GAUSS, S-Plus or Ox. The argued advantage of these languages is their extensibility: one may translate any complex sequence of matrix operations into the language, without the explicit loops or extensive libraries that would be required in C, C++, or Fortran 90. Unlike these compiled languages, the matrix languages are interpreted languages, supporting more rapid development and allowing for interactive use and debugging. Their disadvantages—beyond execution speed, since most (with the notable exception of Ox) are quite slow, even on powerful hardware—relate to the greater housekeeping burden that they impose on the researcher. The convenience features of the statistical packages’ languages are generally absent from matrix languages, in which one must explicitly keep track of the correspondence between named variables and columns of matrices, take account of observations lost through lags, differences, and the handling of unbalanced panel data, and provide much of the logic for the formatted output of results. While the matrix languages conceptually support reusable code, it appears to be relatively scarce in practice.¹ For example, many researchers will provide the GAUSS code used in their work on request, but those routines almost without exception contain explicit references to the particular problem they have solved in terms of the dimensions of the problem, the specific files accessed, and the specific form of the output. A building-block approach is notably absent. Another disadvantage of many of the matrix languages is their

¹ A notable exception is James P. LeSage’s Econometrics Toolbox for MATLAB (<http://www.spatial-econometrics.com/>), which provides an exhaustive set of econometric routines for MATLAB, with full documentation and an extensible development environment, at zero cost.

reliance on numerous components, each sold separately (MATLAB “toolboxes” or GAUSS “applications”), implying that most users will have a different set of available functions in their copy of the language.² The matrix languages are generally more constrained in terms of cross-platform availability than are many statistical packages.

The *Stata* software environment provides an interesting middle ground. *Stata*, a product of Stata Corporation (College Station, TX: <http://www.stata.com>), is a statistical package marketed to a diverse set of customers, with the greatest concentration in the health sciences, rather than econometrics or social science research. Like its major broadly-based competitors SAS and SPSS, *Stata* offers a wide range of statistical and econometric capabilities for the researcher who merely wants to plug in the data and execute a routine statistical analysis. But *Stata* diverges from these packages, and from econometric packages like TSP, eViews and RATS as well, by promoting its extensibility, and providing explicit support for the development and dissemination of procedures crafted by its user community. The core functionality of the package itself may be readily upgraded by its vendor between major releases, and as we will discuss in Section 4, those upgrades (as well as bugfixes) may be readily acquired and installed by the user base. Like other statistical packages, *Stata* performs the housekeeping chores: allowing the user to name variables, operate on variables with wildcard syntax, and rely on the program to keep track of missing values, lags, and the messy details of unbalanced panel data. Advanced features include the ability to transform “wide” data (such as those used in seemingly unrelated regression estimation) into “long” data (that which has been stacked by column with the “vec” operator), suitable for panel data estimators such as fixed effects and random effects. Unlike some statistical packages, *Stata* operates in a vector context, so that transformations (generating new variables and revising existing variables) are specified without explicit loops—and as with matrix languages, explicit loops in this context carry a speed penalty. The entire dataset is held in memory, so that transformations and most estimation procedures are very fast, even when hundreds of thousands of observations are defined. This implies the need for sizable memory resources, but the cost of relaxing that constraint today is quite minimal for most systems.

In contrast to the matrix languages, *Stata* provides a broad set of preformatted output routines, so that the user need not specify the format and perform housekeeping chores in order to generate all of the items needed for presentation of the results at the desired precision. All *Stata* statistical commands leave results in the program’s data structures, so that customized output is readily generated by extracting items from these data structures. Commands are either “r-class” routines, which return results in the `r()` data structures, or “e-class” (estimation) routines, which return results in the `e()` data structures. Each data structure may contain scalars, local macros, matrices, and functions: for instance, `r(mean)` contains the mean of a series after application of r-class command `summarize`, while `e(b)` is a matrix (row vector) of estimated coefficients returned by e-class command `regress`. Results are available until another r-class or e-class command is executed, and may be examined with commands `return list` or `estimates list`.

Stata provides support for users’ programming of statistical optimization routines

² Jurgen Doornik’s *Ox* language (<http://www.nuff.ox.ac.uk/Users/Doornik/doornik.html#ox>) is better placed in this regard, in that additional packages in that environment are generally freely downloadable.

through its `ml` (maximum likelihood) and `nl` (nonlinear least squares) commands.³ It also contains extensive support for Monte Carlo experiments through its `simul` command, and for bootstrap sampling and estimation via the `bstrap` command. Both `simul` and `bstrap` require some degree of programming, but in return Stata handles the many housekeeping details of organizing and saving the results of these experiments.

Unlike both competing statistical packages and the matrix languages, Stata as a product is monolithic: there are no add-ons or optional components available as separate commercial products, so that users' Stata environments are likely to be more homogeneous. Stata is fully cross-platform, with the identical look and feel and product functionality on all supported platforms: currently all versions of Windows, Macintosh OS 8.6 and 9, Mac OS X, Linux (Intel and PPC), and almost every flavor of Unix. Uniquely among the products mentioned, Stata's binary files are exchangeable across "big-endian" and "little-endian" platforms, so that no "export format" nor "transport file" is needed to utilize the binary files on every supported platform—and as discussed below, to access them via Hypertext Transport Protocol (`http`) across the Internet.

3 Software development in Stata

The software development environment created by the designers of Stata is both evolutionary and revolutionary. In its fundamental design, Stata harks back to Unix,⁴ with a very small executable kernel containing the most frequently used commands, and those commands which for reasons of efficiency must be executed in compiled code—e.g., elementary arithmetic and input-output routines. The preponderance of Stata commands (over 82% in the most recent version, 7.0, of January 2000) are provided by several hundred separate "ado-files": automatic do-files, or procedures, whose names correspond to user commands. This design promotes timely updates to the software for bugfixes, since updating the ado-files (which are plain text files) can be achieved by merely copying the corrected files to the appropriate locations (which, as we discuss below, can be performed automatically). Kernel updates are occasionally provided (on average, less than once per month), but since the kernel is quite small (1.5–2 Mb, depending on platform) updated versions can readily be transferred across the Internet.

For a Stata user, the most interesting aspect of this structure is the concept of the "adopath": a Unix-like path containing a number of directories which is searched for an ado-file containing the specified command. The official Stata directories are naturally on the path, but so are directories such as "ado/stbplus," "ado/personal" and the current directory. An extensive library of user-written commands, documented in the *Stata Technical Bulletin (STB)* over the last ten years,⁵ is freely accessible from Stata's web site. The ado-files and

³ To make effective use of the `ml` environment, the book by Gould and Sribney (1999) is essential.

⁴ There are also similarities to MATLAB's structure, in which the vendor's ".m-files," loaded on demand, define a number of the language's commands.

⁵ As of fall 2001, the *Stata Technical Bulletin (STB)* has been transformed into a quarterly, reviewed journal, *The Stata Journal* (<http://www.stata-journal.com>). It will continue to play the *STB*'s role in the distribution

associated help files for these commands may be downloaded, installed in the appropriate directory (e.g. “ado/stbplus”) and the commands defined therein will become first-class citizens in the Stata command language. A continuously expanding archive of user-written additions to Stata is also available as the Boston College Statistical Software Components (SSC) archive,⁶ which provides free access to several hundred Stata modules, which may be installed in the same manner.

The innovative concept underlying this structure is exemplified in the “level playing field” that it creates. A user-written command, whether downloaded from the *STB*, acquired from a colleague, or written oneself is indistinguishable from any “official” Stata command once it is placed on the “adopath.” This feature is similar to that implemented in MATLAB, which also has the concept of a “path” on which it will search for the “.m-file” defining a particular function. Unlike MATLAB, in which on-line help is provided in comments in the body of the “.m-file”, each Stata command has an accompanying “.hlp” file which may be accessed in a separate window.

If it has been carefully crafted, a user-written Stata command will follow the same syntax, provide on-line help, and return results in exactly the same manner as any other command.⁷ Stata’s developers have promoted the development of this quality work in the user community by providing high-level tools for ado-file development: the same tools that they themselves employ. The workings of all official commands that are implemented outside the kernel are revealed to the Stata user, making it straightforward to explore the workings of professionally-developed code and adapt it to one’s purposes. Each ado-file will generally make use of a `syntax` statement: a template specifying the required and optional features of the command’s syntax, which will enforce any rules inherent in the command’s structure. For instance, a command to calculate some function based on a single existing variable will specify that precisely one preexisting variable’s name must be provided: e.g.,

```
syntax varlist(max=1) [, table lag(integer 1) ]
```

would define a command that required a single, existing variable as an argument. The user could specify the `table` option, and an integer value for the `lag` option, which if not specified will take a default value of 1. A command to generate a new variable as a result will specify that constraint in the syntax statement, and the command parser will reject a user’s specification if an existing variable name is provided. An ado-file, defining a command, may also specify that a time-series calendar must have been defined, or that both “i” and “t” variables must have been defined for an operation involving longitudinal (panel) data. A command’s syntax statement will normally allow the optional specification of an `if` clause or an `in` condition, where the former applies the command only if some Boolean expression is satisfied, while the latter specifies that the command should be applied to some specified subset of the currently defined observations. The syntax of the example above could be extended to:

and documentation of user-authored additions to Stata.

⁶ <http://ideas.uqam.ca/ideas/data/bocbocode.html> or <http://econpapers.hhs.se/software/bocbocode/>

⁷ One quite sensible exception: a user-written command may not take the name of an “official” Stata command. A variation on the name must be chosen to prevent confusion.

```
syntax varlist(max=1) [if] [in] [, table lag(integer 1) ]
```

in order to specify that either an `if`-clause or an `in`-clause may optionally be specified. If either of these restrictions on the sample are present, Stata will handle the housekeeping of generating the subset of the sample that is to be analyzed by the user's code with a single `marksample` command.

The constructs we have discussed follow Stata's fundamental model: that an operation, such as the definition of a new variable as a function of existing variables (e.g., calculating the logarithm of income, or the first difference of the price series) will be applied in vector form to the currently defined observations (modified, where specified, by an `if` clause or `in` condition). This model encourages the development of concise, bug-free user programs and procedures. Whereas in many statistical packages one must code an explicit loop over the observations to generate a transformed variable—and perform the housekeeping to deal with lagged values, or differences—in Stata one merely defines the transformation, and it will be applied to all specified observations. This promotes generality in a program or procedure, and allows the user to avoid the use of numerous counters and indices that merely serve to define loop bounds. One can write an explicit loop in Stata code, but in most cases there is a more concise construct that will also carry a speed advantage.

A related tool for the development of clear and concise programs is the “local macro.” Macros are the “variables” of Stata programs—confusing, perhaps, since the data series themselves are also variables. Macros are just that: containers, which can hold text strings of almost arbitrary length, which can be built up by concatenation. For instance, the list of regressors for a given equation can be stored in a macro, and that macro invoked rather than explicitly repeating the variable list. This might seem to be an unnecessary replacement for a copy/paste operation—but in the context of a procedure, or `ado`-file, a local macro can be used to build up that specification automatically from an arbitrary list of specifications provided by the user. For instance, one may conduct the Hylleberg et al. (HEGY, 1990) test for seasonal unit roots in a timeseries specifying that the deterministic variables to be included may include none, a constant, a trend, or a set of seasonal trends. In our implementation of HEGY for quarterly data,⁸ the user specifies a one-word option, and the list of deterministic variables is built up from that response and included in the model specification. Since local macro names may themselves include macros, it is possible to generate a set of local macros which mimic a subscripted array (e.g. a KPSS test (Kwiatkowski et al., 1992) may return values of the KPSS statistic for 0 lags, 1 lag, 2 lags, ... M lags in local macros “`kpss0`”, “`kpss1`”, ... “`kpssM`”. These macros' values will then be accessible to the calling program, where they might be placed into a table of results. Local macros are local to the routine in which they are defined (but they may be returned from that routine), while global macros exist throughout the program.

As an illustration of the usefulness of Stata's macro language, we present a complete Stata program to generate regression estimates of a model expressing real exchange rate volatility (`vrx...`) as a function of median levels of real exchange rate volatility (`ssq...`) and income volatility (`lnip...`), across trading partners, for each of the G-7 countries in turn. The dataset is in “stacked” or panel format, with each observation identified by a country code

⁸ Available from the SSC Archive as routine `hegy4`, written by C. F. Baum and Richard Sperling.

(ccode) and time period (in this case monthly). The dependent and independent variables are computed within the routine as row standard deviations and row medians, respectively, across the remaining countries, using Stata's `egen` (extended generate) command. This program is a good illustration of the ease of integration of user-written commands with "official" Stata commands, since four of Nicholas J. Cox's `listutil` routines⁹ are employed. `wclist` counts the countries to be analyzed; `rotlist` rotates the list; and `takelist` selects each country, in turn, as local macro `dep` and the remaining six countries as local macro `model`. Finally, `prelist` generates variable names by prefixing each country code with the appropriate identifier (i.e. `lnrxS` for the log of the real exchange rate). This program can readily be adapted to work with a different set of countries' data by merely redefining the IMF-assigned country codes in the `cty` macro. The contents of a local macro are referenced by `'macroname'`, while the contents of a global macro are referenced by `$globalname`.

Exhibit 1: Stata program illustrating the use of the macro language

```
log using exhibit1, replace
use "multicountry.dta",clear
* define the list of countries
local cty 112 132 134 136 156 158 111
* count elements in country list, save as local ncty
wclist 'cty'
local ncty = `r(nw)´
* loop over countries
forvalues i = 1/`ncty´ {
* rotate the country list, store as local "now"
    rotlist 'cty',rot(-`i´)
    local now `r(list)´
* take one element from the list, store as local "dep"
    takelist 'now',pos(1)
    local dep `r(list)´
* take remainder of list, store as local "model"
    takelist 'now',pos(2/`ncty´)
    local model `r(list)´
* generate lists of log of real exchange rate (rxlist),
* exchange rate volatility (ssqlist),
* income volatility (iplist)
    prelist "`model´",pre("lnrxS") global(rxlist)
    prelist "`model´",pre("ssq") global(ssqlist)
    prelist "`model´",pre("lnip") global(iplist)
* generate cross-country std.dev. of real exchange rate,
* cross-country median exchange rate volatility,
* cross-country median income volatility
```

⁹ <http://ideas.uqam.ca/ideas/data/Softwares/bocbocodeS391301.html>

```

quietly {
    egen vrx`dep`=rsd($rxlist) if ccode=='dep'
    egen mdssq`dep`=rmean($ssqlist) if ccode=='dep'
    egen mdvol`dep`=rmean($vollist) if ccode=='dep'
}
display _n "Dependent : vrx`dep'"
* regress std.dev. of real exchange rate on
* trading partners' exchange rate and income volatility
    regress vrx`dep' muvol`dep' mussq`dep'
* generate predicted values for each country
    predict vrxhat`dep', xb
}
* save new variables to file
save g7preds, replace
log close
exit,clear

```

This program generates the appropriate data for each country's regression from the remaining countries, runs the regression, and generates the predicted values from this regression as a new variable. At termination, these new variables (and the original variables) are saved to a new binary file, `g7preds`, for further analysis (e.g., graphical presentation).

4 Integration with the Internet

The sample program above makes extensive use of utility routines not provided in "official Stata": `wclist`, `rotlist`, `takelist`, and `prelist` from the `listutil` package. In this section, we discuss the ease of incorporating user-authored components into a copy of Stata. Before touching upon that, we first consider the functionality, added to Stata in January 1999 (version 6.0), for a user with Internet access to instruct Stata to query the vendor's site for updates. By issuing the command `update query`, the user may at any time evaluate the status of her copy versus the most recent available:

Exhibit 2: Update query

```

. update query
(contacting http://www.stata.com)
Stata executable
folder: :Rumelihisari:Stata:
name of file: Stata
currently installed: 08 Aug 2001
latest available: 08 Aug 2001
Ado-file updates
folder: :Rumelihisari:Stata:ado:updates:

```

```
names of files: (various)
currently installed: 06 Sep 2001
latest available: 06 Sep 2001
Recommendation
Do nothing; all files up-to-date.
```

This dialog reveals that both of the components of Stata installed on this machine—the executable, or kernel, and the official “ado-files”—are up-to-date. If either component was available in a more recent version on the vendor’s site, the dialog would recommend that the user download the update. Ado-file updates are automatically placed in the appropriate directories; the executable is copied to the user’s hard disk, and detailed instructions are given for replacing the executable.¹⁰ This update facility does not support a “push” strategy, but rather makes it possible for the user to check the currency of her copy at any convenient time. Updates to the ado-files have been made available about every two weeks, providing both bugfixes to official Stata and significant functionality in the form of additional commands or new features of existing commands.

The ability of Stata to communicate with the vendor’s site also implies that on-line help, when connected to the Internet, can make use of both local and remote resources. The most recent search tool added to Stata, the `findit` command, will search both local on-line help, remote FAQs on the StataCorp web site, the contents of the 60+ issues of the *Stata Technical Bulletin*, and the contents of a number of additional remote sites (such as the SSC archive) that provide downloadable Stata software components. We turn next to the facility for downloading and installing user-authored additions to official Stata.

When StataCorp introduced web accessibility in Stata version 6.0, its designers envisioned that a network of user sites would be constructed and maintained by individual authors. The documentation of the `net` facility provides all of the necessary information for any user to set up an `http`-based web site, accessible from within Stata, from which individual software modules may be downloaded and automatically installed in a user’s copy of Stata. Although there are a number of user sites extant, the vast majority of user-authored materials adding functionality to Stata are indexed in one web-based archive: the Boston College Statistical Software Components Archive (commonly termed the SSC archive). This archive contains metadata, or bibliographic entries, which may be accessed and searched with any web browser. For the vast majority of entries, a copy of the code (`.ado` and `.hlp` files) are housed in the archive, so that they may be accessed and installed from within Stata (which communicates with the web server using standard HTTP protocols). This is the preferential method for acquiring these materials, since any platform-dependent issues (line ending conventions, etc.) are avoided, and the user need not be concerned with the appropriate relocation of downloaded files to the appropriate directories. To provide “one-click” access to these facilities, Stata recently added the `ssc` command, which allows the user to describe a particular routine in the archive and install it with a single command.¹¹ In contrast, The MathWorks provides a vendor site—“MATLAB

¹⁰ The replacement of the executable in the appropriate directory is scripted in the Unix and Linux versions of Stata.

¹¹ Stata’s version 7 `ssc` command is based upon the earlier `archutil` (archive utility) package of Cox and

Central”¹²—which provides a searchable archive of user-contributed software, but must be used from within a web browser (not directly within MATLAB). When files are downloaded, they are customarily in ZIP format, and must be extracted and placed in the appropriate directory on the MATLAB “path.” In my experience, these steps often surpass novice users’ capabilities, especially for those in a Windows environment. Stata’s access to user-written code, from within the package itself, may be less problematic. Exhibit 3 details a user’s enquiry and subsequent installation of the HEGY4 routine from the SSC archive.

Exhibit 3: Illustration of archive utilities

```
. ssc describe hegy4
```

```
-----  
package hegy4 from http://fmwww.bc.edu/repec/bocode/h  
-----
```

TITLE

```
'HEGY4': module to compute Hylleberg et al seasonal unit root test
```

DESCRIPTION/AUTHOR(S)

hegy4 performs the Hylleberg et al. (HEGY) test for seasonal unit roots in a quarterly timeseries. It estimates the four roots of the timeseries representation $(1-B^4)x(t) = e(t)$, where B is the backshift operator, and presents estimates of these roots as $\text{Pi}(1)..\text{Pi}(4)$. Joint tests for $\text{Pi}(3)=\text{Pi}(4)=0$, $\text{Pi}(2)=\text{Pi}(3)=\text{Pi}(4)=0$ and $\text{Pi}(1)=\text{Pi}(2)=\text{Pi}(3)=\text{Pi}(4)=0$ are calculated (the latter two proposed by Ghysels et al.) The routine will also estimate the model with seasonal trends proposed by Smith and Taylor. This is version 1.0.5 of the routine, which corrects an error in the tabulation of critical values.

Author: Christopher F Baum, Boston College
Support: email baum@bc.edu

Author: Richard Sperling, The Ohio State University
Support: email rsperling@bc.edu

INSTALLATION FILES

```
hegy4.ado  
hegy4.hlp  
-----
```

Baum (<http://ideas.uqam.ca/ideas/data/Softwares/bocbocodeS375501.html>), which provides access to the archive to Stata version 6 users as well.

¹² <http://www.mathworks.com/matlabcentral/fileexchange/index.jsp>

```

(type -ssc install hegy4- to install)

. ssc install hegy4, replace
checking hegy4 consistency and verifying not already installed...

the following files will be replaced:
  ~:ado:stbplus:h:hegy4.ado
  ~:ado:stbplus:h:hegy4.hlp

installing into ~:ado:stbplus:...
installation complete.

```

Stata's ability to operate as a web browser has an important implication for those engaged in collaborative research, on the one hand, and for instruction and distance learning on the other. Researchers in separate locations—whether across campus or several time zones away—are able to share datasets, programs, and log files without having to transfer those files by ftp or email. Stata's command to access a binary-format dataset, use *filename*, can refer to a file on one's own hard disk, on a network drive, or at a specified URL anywhere on the Internet. Since there is a single, platform-independent binary dataset format, a dataset mounted on a web server can be readily accessed by distant collaborators (including those whose Internet access is mediated by proxy servers). If their colleague writes a program (such as that presented in Exhibit 1 above) and places it on a web server, the remote user may simply give the Stata command `copy http://... localfile` to acquire a copy of the program, or may inspect it with the Stata command `type http://...` Even Stata's graphics files, which may be saved by each `graph` command, can be placed on a web server, and the Stata command `graph using http://...` allows the remote user to view the graph without having the data used to produce it. These built-in facilities, coupled with access to a web server, make it very straightforward for researchers to exchange materials related to their work, whether they be datasets, program files, graphics files, or Stata procedures (.ado-files).

The advantages of this easy interchange for the instructional process should be readily apparent. In teaching econometrics, instructors usually have to deal with the logistical difficulties of providing access to datasets to be used in assignments in a variety of formats and locations, depending on students' platforms, network access, and location (e.g. whether they will access the data with statistical software in an on-campus computer lab, or from elsewhere on the campus network, or from the public Internet). But in today's campus environment, one commonality emerges: students will inevitably have access to the Internet. Stata's access methods—whether for datasets, program updates and enhancements, or on-line remote help—all depend on http access, which will be available to the broadest group of students (including those who may be engaged in "distance learning," or working at another location while finishing their dissertation research, etc.) The commonality of Stata files and functions across a variety of hardware and operating systems also makes it likely that an instructor can disseminate the electronic materials related to her course with minimal

difficulty. The only difference between accessing a dataset from a local hard drive and over the Internet is the transfer time; even at 56 Kb modem speed, quite sizable datasets may be transferred in reasonable time. An important contributing factor is the efficiency with which Stata stores data (or can be instructed to optimize storage, via the `compress` command). Unlike, e.g., SPSS, which has a single numeric format, using the same storage allocation for the highest-precision floating-point number and a binary indicator variable, Stata has a range of data types, similar to C or Fortran, so that variables may be efficiently stored, and datasets optimized for the storage actually required. The author, who teaches econometrics at undergraduate and Ph.D. levels, has implemented a RePEc (Research Papers in Economics) series containing the datasets from a number of widely used econometrics textbooks, as well as a number of additional datasets (e.g. the Nelson–Plosser macroeconomic data), available as the series “Instructional Stata datasets for econometrics” at IDEAS.¹³ All of these datasets may be accessed by any Internet user from within a recent version of Stata.

5 Case study: testing for endogeneity

To illustrate Stata’s usefulness in applied econometric research, we present an extended example of the sort of rapid prototyping that the development environment supports, allowing a quite general routine to be developed while closely following the derivation of the underlying analytics.

The exhibit below presents the code for `dmexog.ado`, a Stata module implementing an auxiliary regression test for the endogeneity of regressors in an instrumental variables context, written by Steven Stillman and the author.¹⁴ It is available from the SSC archive, via a web browser or from within Stata via the `ssc` command. Comments on the features of this routine, and the aspects of the Stata development environment that they illustrate, are provided below. Most Stata commands may be abbreviated; for clarity, their names are spelled out in this sample code.

We present the rationale for the development and employment of this auxiliary regression test. Many econometrics texts discuss the issue of “OLS vs. IV” in the context of the well-known Hausman test, which involves estimating the model via both OLS and IV approaches and comparing the resulting coefficient vectors. A quadratic form in the differences between the two coefficient vectors—scaled by the precision matrix—gives rise to a test statistic for the null hypothesis that the OLS estimator is consistent and fully efficient. This approach, implemented by Stata’s `hausman` command, has one clear drawback: in finite samples, the precision matrix (defined as the difference between the two estimated variance-covariance matrices of the parameter estimates) may not be positive definite. A less problematic approach is the auxiliary regression framework of Davidson and MacKinnon (1993, p.236), which may always be computed. In the context of a single endogenous variable, consider the model

¹³ <http://ideas.uqam.ca/ideas/data/bocbocins.html>

¹⁴ This version of the routine has been simplified for presentation; the latest archived copy also includes support for `xtivreg` (the instrumental variables estimator applied to panel data).

$$y_1 = \beta_0 + \beta_1 y_2 + \beta_2 z_1 + \beta_3 z_2 + u_1,$$

with z_1 and z_2 assumed exogenous. Assume that z_3 and z_4 are also exogenous, and may be employed in IV estimation of this equation. The auxiliary regression approach involves estimating the reduced form (first-stage) regression for y_2 :

$$y_2 = \gamma_0 + \gamma_1 z_1 + \gamma_2 z_2 + \gamma_3 z_3 + \gamma_4 z_4 + u_2$$

We are concerned with testing that $y_2 \perp u_1$. Since by assumption each z is uncorrelated with u_1 , the first stage regression implies that this condition is equivalent to a test of $u_2 \perp u_1$. Exogeneity of the z 's implies that \hat{u}_2 —the residuals from OLS estimation of equation (5)—will be a consistent estimator of u_2 . Thus, we augment the original equation with \hat{u}_2 and reestimate this equation with OLS. A t -test of the significance of \hat{u}_2 is then a direct test of the null hypothesis (in this context, that $\theta = 0$):

$$y_1 = \beta_0 + \beta_1 y_2 + \beta_2 z_1 + \beta_3 z_2 + \theta \hat{u}_2 + v_1$$

The test may be readily generalized to multiple endogenous variables, since it merely requires the estimation of the first stage regression for each of the suspect variables, and augmentation of the original model with their residual series. The test statistic then becomes an F -test, with numerator degrees of freedom equal to the number of included endogenous variables. The test may also be applied to a subset of the regressors: those whose endogeneity may be questioned. Consider dividing a set of endogenous regressors into two subsets, Y_A and Y_B , where only the second set of variables are to be tested for endogeneity. Then a modified test involves estimating the first stage regression for each variable in Y_B in order to generate a residual series. These residual series are then used to augment the original model, and a t -test or F -test used to judge their significance.

Exhibit 4: dmexog.ado

```

*! dmexog V1.3.10    C F Baum and Steve Stillman
* with help from Mark Schaffer
* Ref: Davidson & MacKinnon, Estimation and Inference
*      in Econometrics, p.239-242

program define dmexog, rclass
    version 7.0
    syntax [anything]
    local xvarlist `anything'

    if "`e(cmd)'" ~= "ivreg" {
        error 301
    }
    if "`e(vctype)'" == "Robust" {
        di in red "test not valid with robust covariance estimates"
        exit 198
    }

```

```

if "`e(wtype)'" == "aweight" | "`e(wtype)'" == "iweight" {
    di in red "test not valid with aweights or iweights"
    exit 198
}

tempname touse depvar inst incrhs nin b varlist i word regest weight
tempname rhadd idvar

        /* mark sample */
gen byte `touse' = e(sample)
        /* dependent variable */
local depvar `e(depvar)'
        /* instrument list */
local inst `e(insts)'
        /* included RHS endog list */
local incrhs `e(instd)'
local nendog : word count `e(instd)'
        /* get regressorlist of original model */
mat `b' = e(b)
local varlist : colnames `b'
local varlist : subinstr local varlist "_cons" "", /*
*/          word count(local hascons)
* if no constant in original model, exclude from aux regr
if `hascons' == 0 {local noc = "noc"}
        /* get weights setting of original model */
local weight ""
if "`e(wexp)'" != "" {
    local weight "['e(wtype)''e(wexp)']"
}
* 1.3.7: check if xvarlist is populated, if so validate entries
local ninc 0
local rem 0
if "`xvarlist'" ~= "" {
    local nexog : word count `xvarlist'
    local rem = `nendog' - `nexog'
    local nincrhs `incrhs'
    foreach v of local xvarlist {
        local nincrhs: subinstr local nincrhs "`v'" "", /*
*/          word count(local zap)
        if `zap' ~= 1 {
            di in r_n "Error: `v' is not an endogenous variable"
            exit 198
        }
    }
}

```

```

    }
* remove nincrhs from varlist if rem>0 and load xvarlist in incrhs
    if `rem' > 0 {
        foreach v of local nincrhs {
            local varlist: subinstr local varlist "`v'" "", word
        }
        local incrhs `xvarlist'
    }
* incrhs now contains the pruned list of vars assumed exogenous
* nincrhs contains the remaining included endogenous
* varlist contains the included exogenous
    local ninc : word count `incrhs'
}
* deal with ts operators in endog list
tsrevar `incrhs', sub
local incrhs `r(varlist)'
local rhadd ""
estimates hold `regest'
foreach word of local incrhs {
    qui regress `word' `inst' `weight' if `touse'
    tempvar v_`word'
    qui predict double `v_`word'', r
    local rhadd "`rhadd' `v_`word'''"
}
if (`ninc' == 0 | `rem' == 0) {
    qui regress `depvar' `varlist' `rhadd' `weight' /*
    */ if `touse', `noc'
}
else {
    qui ivreg `depvar' `varlist' `rhadd' (`nincrhs' = `inst') /*
    */ `weight' if `touse', `noc'
}
qui test `rhadd'
return scalar df = r(df)
return scalar df_r = r(df_r)
return scalar dmexog = r(F)
return scalar p = r(p)
di in gr _n "Davidson-MacKinnon test of exogeneity: " /*
*/ in ye %9.0g return(dmexog) in gr /*
*/ in gr " F(" %2.0f in ye return(df) "," return(df_r) /*
*/ in gr ") P-value = " in ye %6.0g return(p)

end
exit

```

The code for this routine illustrates a number of the features of Stata's development environment that make it quite straightforward to generate a "rapid prototype" of an econometric routine from its analytical development in the literature. The `program` line indicates the name of the routine (which must be that of the file which contains it) and specifies that the routine is "rclass"—that is, returned macros, scalars and matrices will be denoted `r(retval)`. The `version` statement specifies that version 7 of Stata is required to execute the code. The `syntax` statement provides a template for the user's invocation of this command. Since `dmexog` is used following an estimation command, it need not have any arguments; it supports an optional `varlist` if a subset test is specified.

An initial set of checks ensures that the most recent command was `ivreg`; that `ivreg` was not invoked with the `robust` option; and that if weights were specified, they were compatible with the test. `tempname` serves to define a number of macros local to the routine. The `ivreg`'s `e()` data structure is accessed to retrieve the variables specified in that routine, which are then used to set up the appropriate auxiliary regressions. If the user has specified the optional `varlist` when invoking `dmexog`, each of the variables must be checked to ensure that they not only exist but were among the included endogenous regressors in the prior command. The `incrhs` macro then contains the variables from which residuals are to be generated; those residuals are stored as temporary variables `'v' 'word'` in a loop over words in `incrhs`. The auxiliary regression is then executed, as either an OLS `regress` or an IV `ivreg`, and Stata's `test` command used to generate the appropriate test statistic. `test` returns the statistic, its p-value, and degrees of freedom in the `r()` data structure; those elements are then formatted for presentation in `dmexog`'s output. The `return scalar` statements are used to make computed values from the routine available to the calling routine; e.g. `return scalar dmexog` will cause the value of the F-statistic to be accessible as `r(dmexog)` in the calling routine, while `return scalar p` will place its p-value in `r(p)`.

We now include an example of the use of this routine, as presented in the on-line help file for `dmexog`. We must first access an appropriate dataset: in this case, the Mroz dataset, as provided with Wooldridge's (2000) econometrics text, containing data on 428 working married women. An instrumental variables estimate of the log of the worker's wage is computed, using experience and education as potentially endogenous variables, instrumented by mother's, father's and husband's levels of education (so that the equation is overidentified). In the interest of brevity, the regression output is suppressed. The `dmexog` test is applied, and indicates that the null hypothesis of exogeneity (i.e., that OLS would be a consistent estimator of the equation) cannot be rejected. A second invocation of `dmexog` considers only experience as potentially exogenous, assuming that education is endogenous; this null hypothesis cannot be rejected.

Exhibit 5: use of the `dmexog` routine

```
. quietly ivreg lwage (exper educ = motheduc fatheduc huseduc)
. dmexog
```

```
Davidson-MacKinnon test of exogeneity:  1.531276  F( 2,423)
                                           P-value =  .2175
```

```
. dmexog exper
```

```
Davidson-MacKinnon test of exogeneity:  .5944261  F( 1,424)
                                           P-value =  .4411
```

We may be interested in gauging the performance of this test versus that of the well-known Hausman test in this context. We have conducted extensive simulation experiments, using Stata's `simul` command, over a range of sample sizes and constructed correlations between regressor and error. Although we will not reproduce the program listing and results here in the interest of brevity, they are available upon request. Our findings are quite robust: the `dmexog` test exhibits comparable power, over both specified factors, to the standard Hausman test for the detection of endogeneity among the regressors when applied to cross-sectional data. Since the `dmexog` test can be calculated for all samples, and is more straightforward for the user, one may rely on it without concern for its efficacy. Interestingly, in longitudinal (panel) data, the auxiliary regression test appears to be considerably more powerful. Although further research is needed to identify the source of this power differential, it points out the value of working in an environment where such comparisons may readily be programmed and tabulated.

6 Conclusions

Applied economic researchers should be concerned with the efficiency of their computational environment in terms of the amount of their own time required to perform their work. High-level tools such as matrix languages are often not very efficient from a computational standpoint, but the speed at which CPU cycles can be delivered at a given price is constantly declining. Since researchers' processing speed is not increasing, an environment which permits researchers to economize on their own time will be the most effective. The Stata programming environment poses significant advantages over those provided by matrix languages and traditional statistical packages in terms of its ease of use, extensibility, documentation of research strategy, and explicit support for collaborative research. From a programming language standpoint, Stata's features encourage the development of reusable, robust and well-documented code, as evidenced by the large collection of reliable extensions to the program that are in the public domain. Those seeking a workable environment for applied economic research should gain familiarity with Stata's potential.

Acknowledgements

C. F. Baum is an associate professor of economics at Boston College, where he co-directs the Minor in Scientific Computation in the College of Arts and Sciences. He is an associate editor of *Computational Economics* and *The Stata Journal*, and serves on the Advisory Council of the Society for Computational Economics. Baum founded and manages the Boston College Statistical Software Components archive at RePEc (<http://repec.org>), the largest Web repository of Stata code. He acknowledges useful discussions with Petia Petrova, Nicholas J. Cox, William Gould, Stanislav Kolenikov, Francesco Zanetti, an anonymous reviewer, and the editor of this volume. The standard disclaimer applies.

References

- [1] Baum, C. F., 2000. sts15: Tests for stationarity of a time series. *Stata Technical Bulletin* 57, 36–39.
- [2] Baum, C. F. and R. Sperling, 2000. sts15.1: Tests for stationarity of a time series: Update. *Stata Technical Bulletin* 58, 35–36.
- [3] Davidson, R. and MacKinnon, J., *Estimation and Inference in Econometrics*, 1993, New York: Oxford University Press.
- [4] Gould, W., and W. Sribney, 1999. *Maximum likelihood estimation with Stata*. College Station, TX: Stata Press.
- [5] Hylleberg, S., Engle, R. F., Granger, C. W. J. and B. S. Yoo, 1990. Seasonal integration and cointegration. *Journal of Econometrics*, 44, 215–238.
- [6] Kwiatkowski, D., Phillips, P. C. B., Schmidt, P. and Y. Shin, 1992. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of Econometrics*, 54, 159–178.
- [7] Wooldridge, J., *Introductory Econometrics: A Modern Approach*, 2000, New York: South-Western College Publishing.