AgEcon SEARCH
RESEARCH IN AGRICULTURAL & APPLIED ECONOMICS

The World's Largest Open Access Agricultural & Applied Economics Digital Library

This document is discoverable and free to researchers across the globe due to the work of AgEcon Search.

Help ensure our sustainability.

Give to AgEcon Search

AgEcon Search
http://ageconsearch.umn.edu
aesearch@umn.edu

*Papers downloaded from **AgEcon Search** may be used for non-commercial purposes and personal study only. No other use, including posting to another Internet site, is permitted without permission from the copyright owner (not AgEcon Search), or as allowed under the provisions of Fair Use, U.S. Copyright Act, Title 17 U.S.C.*

No endorsement of AgEcon Search or its fundraising activities by the author(s) of the following work or their employer(s) is intended or implied.

The Stata Journal

Editors

H. Joseph Newton
Department of Statistics
Texas A&M University
College Station, Texas
editors@stata-journal.com

Nicholas J. Cox
Department of Geography
Durham University
Durham, UK
editors@stata-journal.com

Associate Editors

Christopher F. Baum, Boston College
Nathaniel Beck, New York University
Rino Bellocco, Karolinska Institutet, Sweden, and University of Milano-Bicocca, Italy
Maarten L. Buis, University of Konstanz, Germany
A. Colin Cameron, University of California–Davis
Mario A. Cleves, University of Arkansas for Medical Sciences
William D. Dupont, Vanderbilt University
Philip Ender, University of California–Los Angeles
David Epstein, Columbia University
Allan Gregory, Queen's University
James Hardin, University of South Carolina
Ben Jann, University of Bern, Switzerland
Stephen Jenkins, London School of Economics and Political Science
Ulrich Kohler, University of Potsdam, Germany
Frauke Kreuter, Univ. of Maryland–College Park
Peter A. Lachenbruch, Oregon State University
Jens Lauritsen, Odense University Hospital
Stanley Lemeshow, Ohio State University
J. Scott Long, Indiana University
Roger Newson, Imperial College, London
Austin Nichols, Urban Institute, Washington DC
Marcello Pagano, Harvard School of Public Health
Sophia Rabe-Hesketh, Univ. of California–Berkeley
J. Patrick Royston, MRC Clinical Trials Unit, London
Philip Ryan, University of Adelaide
Mark E. Schaffer, Heriot-Watt Univ., Edinburgh
Jeroen Weesie, Utrecht University
Ian White, MRC Biostatistics Unit, Cambridge
Nicholas J. G. Winter, University of Virginia
Jeffrey Wooldridge, Michigan State University

Stata Press Editorial Manager

Lisa Gilmore

Stata Press Copy Editors

David Culwell, Shelbi Seiner, and Deirdre Skaggs

The *Stata Journal* publishes reviewed papers together with shorter notes or comments, regular columns, book reviews, and other material of interest to Stata users. Examples of the types of papers include 1) expository papers that link the use of Stata commands or programs to associated principles, such as those that will serve as tutorials for users first encountering a new field of statistics or a major new technique; 2) papers that go "beyond the Stata manual" in explaining key features or uses of Stata that are of interest to intermediate or advanced users of Stata; 3) papers that discuss new commands or Stata programs of interest either to a wide spectrum of users (e.g., in data management or graphics) or to some large segment of Stata users (e.g., in survey statistics, survival analysis, panel analysis, or limited dependent variable modeling); 4) papers analyzing the statistical properties of new or existing estimators and tests in Stata; 5) papers that could be of interest or usefulness to researchers, especially in fields that are of practical importance but are not often included in texts or other journals, such as the use of Stata in managing datasets, especially large datasets, with advice from hard-won experience; and 6) papers of interest to those who teach, including Stata with topics such as extended examples of techniques and interpretation of results, simulations of statistical concepts, and overviews of subject areas.

The *Stata Journal* is indexed and abstracted by *CompuMath Citation Index*, *Current Contents/Social and Behavioral Sciences*, *RePEc: Research Papers in Economics*, *Science Citation Index Expanded* (also known as *SciSearch*), *Scopus*, and *Social Sciences Citation Index*.

For more information on the *Stata Journal*, including information for authors, see the webpage

http://www.stata-journal.com

Subscriptions are available from StataCorp, 4905 Lakeway Drive, College Station, Texas 77845, telephone 979-696-4600 or 800-STATA-PC, fax 979-696-4601, or online at

http://www.stata.com/bookstore/sj.html

Subscription rates listed below include both a printed and an electronic copy unless otherwise mentioned.

U.S. and Canada		Elsewhere	
Printed & electronic		**Printed & electronic**	
1-year subscription	$115	1-year subscription	$145
2-year subscription	$210	2-year subscription	$270
3-year subscription	$285	3-year subscription	$375
1-year student subscription	$ 85	1-year student subscription	$115
1-year institutional subscription	$345	1-year institutional subscription	$375
2-year institutional subscription	$625	2-year institutional subscription	$685
3-year institutional subscription	$875	3-year institutional subscription	$965
Electronic only		**Electronic only**	
1-year subscription	$ 85	1-year subscription	$ 85
2-year subscription	$155	2-year subscription	$155
3-year subscription	$215	3-year subscription	$215
1-year student subscription	$ 55	1-year student subscription	$ 55

Back issues of the *Stata Journal* may be ordered online at

http://www.stata.com/bookstore/sjj.html

Individual articles three or more years old may be accessed online without charge. More recent articles may be ordered online.

http://www.stata-journal.com/archives.html

The *Stata Journal* is published quarterly by the Stata Press, College Station, Texas, USA.

Address changes should be sent to the *Stata Journal*, StataCorp, 4905 Lakeway Drive, College Station, TX 77845, USA, or emailed to sj@stata.com.

Copyright © 2015 by StataCorp LP

Copyright Statement: The *Stata Journal* and the contents of the supporting files (programs, datasets, and help files) are copyright © by StataCorp LP. The contents of the supporting files (programs, datasets, and help files) may be copied or reproduced by any means whatsoever, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the *Stata Journal*.

The articles appearing in the *Stata Journal* may be copied or reproduced as printed copies, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the *Stata Journal*.

Written permission must be obtained from StataCorp if you wish to make electronic copies of the insertions. This precludes placing electronic copies of the *Stata Journal*, in whole or in part, on publicly accessible websites, fileservers, or other locations where the copy may be accessed by anyone other than the subscriber.

Users of any of the software, ideas, data, or other materials published in the *Stata Journal* or the supporting files understand that such use is made without warranty of any kind, by either the *Stata Journal*, the author, or StataCorp. In particular, there is no warranty of fitness of purpose or merchantability, nor for special, incidental, or consequential damages such as loss of profits. The purpose of the *Stata Journal* is to promote free communication among Stata users.

The *Stata Journal* (ISSN 1536-867X) is a publication of Stata Press. Stata, Stata, Stata Press, Mata, Mata, and NetCourse are registered trademarks of StataCorp LP.

The Stata Journal (2015)
15, Number 1, pp. 45–76

More power through symbolic computation: Extending Stata by using the Maxima computer algebra system

Giovanni L. Lo Magno
Department of Economics, Business, and Statistics
University of Palermo
Palermo, Italy
lomagno.gl@virgilio.it

Abstract. Maxima is a free and open-source computer algebra system that can perform symbolic computations such as solving equations, determining derivatives of functions, obtaining Taylor series, and manipulating algebraic expressions. In this article, I present the Maxima Bridge System, which is a collection of software programs that allows Stata to interface with Maxima so that Maxima can be used for symbolic computation to transfer data from Stata to Maxima and to retrieve results from Maxima. The cooperation between Stata and Maxima provides an environment for statistical analysis in which symbolic computation can be easily used together with all the facilities supplied by Stata. In this environment, symbolic computation algorithms can be used to manage the complexity of algebra and calculus, whereas numerical computation can be used when speed matters. I also discuss the software architecture of the Maxima Bridge System, and I present several examples to illustrate how to develop new Stata commands that exploit the capabilities of Maxima.

Keywords: pr0059, maxima, maximaget, maximaput, Maxima Bridge System, symbolic computation, computer algebra system

1 Introduction

A computer algebra system (CAS) is a software program that can perform symbolic computations; namely, it can manipulate symbols for mathematical objects such as ratios, numbers with arbitrary numerical precision, equations, integrals, and well-known mathematical constants (Grabmeier, Kaltofen, and Weispfenning 2003). Unlike software programs used for numerical computations, such as Stata, CAS uses symbols as the main inputs and outputs. For example, the computation of $\Gamma(1/2)$ provides the following output in Stata,

```
. display exp(lngamma(1/2))
1.7724539
```

while the evaluation of the same expression in a CAS gives $\sqrt{\pi}$ as output. This latter algebraic expression is exactly what would be displayed on the monitor. Stata stores the numerical information it manages in machine floating-point numbers, namely, sequences

© 2015 StataCorp LP

of bits that represent both integer and noninteger numbers. Thus the previous result is internally stored in Stata as the following 64-bit sequence:

0011111111111100010110111111100010010001101101001110111101101011

In a CAS, the evaluation of the same expression will be stored in a more sophisticated manner, which could be described as "the square root of the well-known mathematical constant π". Data stored by Stata are approximations of the real value $\sqrt{\pi}$ because an exact representation in the IEEE754 format of that number would require an infinite number of bits (IEEE 2008; Linhart 2008). Mathematical information stored in a CAS is not affected by issues of numerical precision, and it can therefore be said to be "exact".

Computer algebra overcomes the risk of obtaining incorrect results from algebraic manipulations, which easily occur when computations are performed with traditional pencil-and-paper methods. In addition, computer algebra provides easy access to advanced mathematical techniques; resolves problems that would otherwise be intractable for humans (when they are too complex or time consuming); allows one to obtain complex formulas from a simple description of the theory on which they are based; and facilitates the testing of conjectures and enables the researcher to perform experimentation with mathematical objects (Cohen, Davenport, and Heck 1993). For these reasons, computer algebra is used in many fields of science, including the life sciences (Barnett 2002), physics, and engineering (Boyle and Caviness 1990). Zeilberger (2004) introduced several examples of applications to statistics and automatically obtained theorems for the calculus of the moments of combinatorial random variables through computer algebra. Bollen and Bauldry (2010) also presented examples, in which they assessed the identifiability of structural equation models (Hoyle 2012) using a general method requiring the solutions of complex, nonlinear systems of equations.

There are currently special symbolic software programs for statistics. For example, Rose and Smith created mathStatica (Rose and Smith 2002; Stokes 2012; Vinod 2003), an add-on for the commercial Mathematica CAS (Wolfram 2003) that makes it possible to perform complex symbolic computations in statistics. These computations include expected values for random variables (defined by arbitrary probability density functions); a derivation of the distribution of the product of random variables; derivations of maximum likelihood estimators (Rose and Smith 2000); and derivations of exact density functions of order statistics (Rose and Smith 2005). Another software package used in symbolic statistical computations is A Probability Programming Language, or APPL (Glen, Evans, and Leemis 2001), which is based on the Maple CAS (Dodson 2004) and is oriented to the manipulation of random variables. These software packages relieve the statistician from the burden of complex and error-prone algebraic manipulations and provide students with a powerful learning tool (Rose 2007).

Maxima is a free and open-source CAS written in Lisp, and it can be downloaded from http://maxima.sourceforge.net (Joyner 2006; Li and Racine 2008). Maxima permits access to the underlying Lisp implementation on which it runs, and the user can use both Lisp and Maxima instructions in the Maxima environment. It is a command-line software program. Thus interaction with Maxima is effected through the terminal; that is, the user inputs the commands from the keyboard, while the output is obtained in plain-text style. Several graphical user interfaces (GUIs) for use with Maxima are freely available, all of which provide comfortable point-and-click interfaces for interacting with Maxima and for obtaining a well-formatted, textbook-style output. I recommend using wxMaxima, which is a free GUI for use with Maxima (http://sourceforge.net/projects/wxmaxima/).

In this article, I present Maxima Bridge System (MBS), a collection of software programs that enables Stata to use Maxima as a symbolic computational engine. With MBS, the user can launch Maxima commands from within Stata in an interactive session or create new Stata commands that have been written by mixing Stata and Maxima instructions. Programs with Stata and Maxima instructions combined are called ado-Maxima programs in the MBS jargon.

The components of MBS are varied:

- Maxima Bridge: a stand-alone application, written in C++, that links Stata and Maxima, thus allowing the interaction between the two software programs;
- Maxima Plugin: a Stata plugin, written in C++, that communicates via a local TCP/IP connection with Maxima Bridge to dispatch commands to Maxima and retrieve subsequent output;
- `maxima`: a Stata command that basically functions as a wrapper for Maxima Plugin;
- `maximaget`: a Stata command used to transfer data from Maxima to Stata;
- `maximaput`: a Stata command used to transfer data from Stata to Maxima; and
- MBS utilities package: a collection of Maxima commands, written in Lisp, that eases the interaction between Stata and Maxima and assists with various programming tasks.

To my knowledge, MBS is the first software to integrate a CAS into Stata. Analogous systems for the statistical software R (R Development Core Team 2014) are the Ryacas package (Goedman et al. 2012), which allows for integration with the Yacas CAS (http://yacas.sourceforge.net), and rSymPy (Grothendieck 2012), which interfaces R with the Python library for symbolic computation, SymPy (http://www.sympy.org).

MBS and its source code can be downloaded from

https://sourceforge.net/projects/maximabridgesystem/

The ado-files and plugins of MBS can be downloaded from

http://www.stata-journal.com/software/sj15-1/pr0059

This article is organized as follows: In section 2, I discuss how MBS works and introduce its capabilities. In section 3, I further discuss the software architecture of MBS. In section 4, I highlight the differences between Stata and Maxima regarding the treatment of floating-point numbers. In sections 5 through 8, I describe the syntax of the Stata commands of MBS and provide various examples of use for these commands. In section 9, I describe the Maxima functions from the MBS utilities package. In section 10, I propose various applications of MBS. Finally, in section 11, I conclude.

The aim of this article is not to introduce Maxima. To learn about Maxima, please refer to the reference manual and several tutorials, which are available at http://maxima.sourceforge.net/documentation.html.

2 A glance at MBS

Before discussing the full syntax relating to the Stata commands provided by MBS, I will outline a sample MBS session using the `maxima`, `maximaget`, and `maximaput` commands. Here I provide the basics relating to the launch of a Maxima command from Stata, transferring data from Stata to Maxima, and retrieving data from Maxima to Stata.

To enable the interaction between Stata and Maxima, one must launch Maxima Bridge. Thereafter, the Maxima process[1] must also be running, and a client must be connected to Maxima Server with Stata Server listening (Maxima Server and Stata Server will be discussed in section 3). These conditions can be verified by looking at the status bar of the Maxima Bridge application.

An interactive Maxima session can be launched by simply typing `maxima` in Stata.

```
. maxima
-- maxima (type end to exit) ------------------------------------------------
 . val: integrate(sqrt(x), x, 0, 1);
                                        2
(%o1)                                   -
                                        3
(%i2)
 . end
-------------------------------------------------------------------------------
```

After launching the `maxima` command, one can use only Maxima commands. To exit from the Maxima session, one can simply type `end`.

1. A Maxima process is automatically launched and used by Maxima Bridge: Maxima Bridge communicates with this process to issue Maxima commands and retrieve the subsequent output.

In the previous Maxima session, the value for the definite integral (from 0 to 1) of $f(x) = \sqrt{x}$ was calculated and assigned to the Maxima symbol `val`. This value can be imported into Stata using the `maximaget` command.

```
. maximaget val
(%o2)                                done
(%i3)
. scalar dir val
       val =  .66666667
```

The scalar, created in Stata in the previous example, contains the floating-point approximation of 2/3.

In the next example, a scalar is created in Stata, and its value is assigned to the Maxima symbol `stval` by using `maximaput`. Thereafter, a Maxima session is invoked to check that the data have been correctly transferred.

```
. scalar stval = 21.3
. maximaput stval
(%o3)                                21.3
(%i4)
. maxima
— maxima (type end to exit) ————————————————————————————————————————
 . stval;
(%o4)                                21.3
(%i5)
 . end
————————————————————————————————————————————————————————————————————
```

Maxima can also be invoked in inline mode, namely, by typing the word `maxima` immediately followed by Maxima instructions, as follows:

```
. maxima val: integrate(sqrt(x), x, 0, 1);
                                      2
(%o5)                                 -
                                      3
(%i6)
```

In this mode, no Maxima session will remain open, and Stata is ready to accept ordinary Stata commands after the command has been executed.

An important feature of the inline mode is the possibility of mixing Maxima instructions with Stata macros.

```
. global sup "5"
. maxima integrate(sqrt(x), x, 0, $sup);
                                        3/2
                                     2 5
(%o6)                                ------
                                       3
(%i7)
```

In the above example, the code is executed by Maxima as if the user typed `maxima integrate(sqrt(x), x, 0, 5);`.

Maxima requires that instructions end with a semicolon (`;`) or a dollar sign (`$`) (a dollar sign at the end of a Maxima instruction suppresses its subsequent output). If neither a semicolon or a dollar sign is typed at the end of the string that has been passed to `maxima`, then a semicolon is automatically appended.

3 MBS architecture

Stata communicates with Maxima in the MBS environment through a local TCP/IP network. The network infrastructure enabling the connection between the two software programs is provided by Maxima Bridge, which is a stand-alone application acting as a bridge between Stata and Maxima (figure 1). To issue a command to Maxima, Stata connects to Maxima Bridge to dispatch a string containing the command. Next, Maxima Bridge reads the string and repeats it to Maxima. Then, Maxima executes the command and sends the output back to Maxima Bridge. Finally, Maxima Bridge sends the received output to Stata. Thus the entire MBS works only if Maxima Bridge is running correctly.

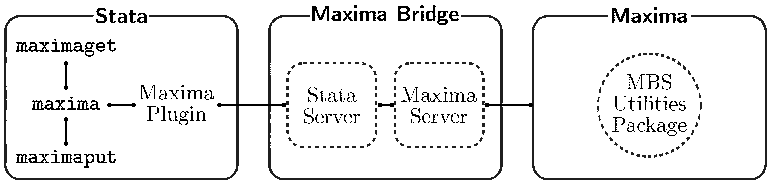

Figure 1. The MBS architecture

When Maxima Bridge starts, a Maxima process is launched in the background. Once started, Maxima will immediately connect to Maxima Bridge, and it will run as a client (however, Maxima does not run as a server). Maxima Bridge manages two different servers: Stata Server and Maxima Server. Stata Server opens a network port (whose default number is 4059), which is used by Stata to communicate with Maxima Bridge, while Maxima Server opens a port (default number 4060) to allow for communication between Maxima Bridge and Maxima. The Stata server port is left open to allow Stata to communicate with Maxima Bridge.

Maxima Bridge can be considered as independent software from MBS in that it is useful even when not used as a "bridge" between Stata and Maxima. It also functions as a GUI for Maxima, and its interface can be used to issue commands to Maxima and obtain output. The GUI of Maxima Bridge can be extended through plugins,[2] which are written in C++. A Maxima Bridge plugin provides a convenient GUI for issuing Maxima commands. For more information regarding Maxima Bridge and to learn how to develop new plugins, see the online help for Maxima Bridge.

2. Such plugins, which are called Maxima Bridge plugins in MBS jargon, should not be confused with Stata plugins.

The MBS components running inside Stata include Maxima Plugin and the following Stata commands: `maxima`, `maximaget`, and `maximaput`. Maxima Plugin is at the heart of MBS because it allows communication between Stata and Maxima Bridge. Every command sent to Maxima is issued through Maxima Plugin. Furthermore, Maxima output is retrieved by Maxima Plugin itself and generally echoed to Stata. Maxima Plugin is written in C++. MBS provides three versions of this plugin for the major operating systems Linux, Mac, and Windows; they are called `maximaplugin_linux.plugin`, `maximaplugin_mac.plugin`, and `maximaplugin_windows.plugin`, respectively. `maxima` automatically chooses the correct version to load according to the operating system currently in use. The Maxima Plugin will be discussed in more detail in section 5.

Maxima Plugin could be directly used, but it is not very user friendly. Thus MBS provides the `maxima` command, which is a convenient wrapper for the plugin. `maxima` can be used to issue commands to Maxima in an interactive session or to include Maxima instructions in an ado-program.

After a command has been issued via Maxima Plugin or the `maxima` command, the Stata GUI freezes until all output has been received. In other words, Stata cannot be used when Maxima is calculating. Every output sent from Maxima to Stata is a string with an ASCII code 4 as its terminating character, which is used to mark the end of the string transmission. This terminating character is automatically appended to the output because Maxima Bridge sets this character as the prompt suffix for Maxima once Maxima has been launched. Thus, every time a prompt is obtained in Maxima, this terminating character is sent to the stream, which is redirected to Stata. When Stata receives the terminating character, it "understands" that the output has been received in its entirety, and it then disconnects from Stata Server, prints the output, and restores its GUI.

If the terminating character is not received, Stata will freeze interminably. For example, this can happen when the user forgets to close the quotes in a command, as is the case with the command string `print("hello world`. When Maxima receives this string, it expects a second quotation mark, so it does not process the string. Consequently, no output is produced, the prompt is not displayed, and no terminating character is obtained. In such cases, the user could complete the string by typing "`");`" from the Maxima Bridge console, thus unfreezing Stata. Unfortunately, there are no easy solutions in other cases. To learn how to deal with such cases, see the online help of Maxima Bridge.

The `maximaget` and `maximaput` commands can be used to retrieve data from Maxima and transfer data to Maxima, respectively. `maximaget` orders Maxima to write the requested data in a temporary file and then reads this file; `maximaput` writes the data in a temporary file and then orders Maxima to read it. This file is written in binary format using 64-bit precision, thus ensuring precision and speed. The `maximaget` and `maximaput` commands will be discussed in section 7 and section 8, respectively.

When the Maxima process is launched by Maxima Bridge, it loads a package of Maxima functions called the MBS utilities package. This package provides a set of utility functions that eases the interaction between Stata and Maxima. The functions are written in

Lisp, and their code is contained in the `maximabridge-init.lisp` file, which is located in the `config` subfolder under the directory in which Maxima Bridge has been installed. An example of these functions is `statamissing()`, a function that provides the floating-point number corresponding to the Stata missing value, specified as the argument of this function (for example, `statamissing("s")` provides 9.0301602172344116E+307, which corresponds to `.s`). The remaining functions relating to the MBS utilities package will be discussed in subsequent sections.

4 Numerical issues

When transferring data from Stata to Maxima and vice versa, the user should be aware of numerical issues that can arise from the different ways that the two software programs handle numerical data. The main issue involves missing values (see [U] **12.2.1 Missing values**). These are internally represented by large positive numbers in Stata (where they affect the evaluation of mathematical expression when they are involved as operands), but they are absent from Maxima.

The following result of an arithmetic operation in Stata, in which one of the operands is a missing value, is itself a missing value:

```
. scalar num = 21
. scalar den = .
. scalar statares = num / den
. scalar dir statares
  statares =          .
```

When imported into Maxima, the value for `den` would simply be read as a large positive number. The result of the same operation involving the same binary numbers used previously would therefore be different.

```
. maximaput num
(%o1)                                21.0
(%i2)
. maximaput den
(%o2)                      8.9884656743115795E+307
(%i3)
. maxima maximares: num / den;
(%o3)                      2.3363275514325615E-307
(%i4)
```

Also note the way in which Stata and Maxima behave differently when they are requested to perform arithmetic operations that are undefined (such as division by 0), operations giving rise to overflow (such as 10^{350}), and operations that do not return real values (such as $\sqrt{-4}$). In all these cases, Stata returns the system missing value (`.`). However, Maxima behaves differently; a division by zero returns an error message, and an overflow can arise only when Maxima performs calculations involving numbers that are internally represented as floating-point numbers. The square root of a neg-

ative number is not a real number in Maxima, so it can not possess a floating-point representation (which is readable by Stata).

5 Maxima Plugin

Maxima Plugin is the MBS program that enables Stata to connect to Maxima Bridge. It is a Stata plugin[3] written in C++ and compiled for Linux, Mac, and Windows. For those three operating systems, MBS provides these plugins: `maximaplugin_linux.plugin`, `maximaplugin_mac.plugin`, and `maximaplugin_windows.plugin`, respectively. To directly use the plugin, one loads the correct version for the operating system in use.

The plugin accepts two arguments: the first is the name of the macro containing the Maxima code to be executed, and the second is the Stata Server port. The following code provides an example of use in Linux:

```
. local code "taylor(exp(x), x, 0, 9);"
. program maximaplugin, plugin using(maximaplugin_linux.plugin)
. plugin call maximaplugin, "_code" "--port" "4059"
                  2    3    4    5     6      7      8        9
                 x    x    x    x     x      x      x        x
(%o1)/T/ 1 + x + -- + -- + -- + --- + --- + ---- + ----- + ------ + . . .
                 2    6    24   120   720   5040   40320   362880
(%i2)
```

Because `code` is a local macro, the name that was passed to the Maxima Plugin was `_code`.[4] The 4059 port number was passed to the plugin, preceded by the "`--port`" option name. If the port number is unspecified, 4059 is assumed to be the default value. If the Maxima code passed to the Maxima Plugin does not terminate with a semicolon or a dollar sign, a semicolon is automatically appended to the command string.

Maxima Plugin is internally used by `maxima`. This plugin is not designed to be used by average users; it is more convenient to use `maxima` because it is a convenient wrapper for the plugin. I discuss the `maxima` command in the following section.

6 The maxima command

The maxima `command` permits the user to execute Maxima commands from within Stata and to obtain the subsequent output in the Results window. Briefly, `maxima` is a command providing the user with a friendly interface between Stata and Maxima.

3. See http://www.stata.com/plugins/.
4. Macro names that begin with an underscore are indeed local macros (see [P] **macro**).

This command functions only when Maxima Bridge is running correctly. Maxima Bridge works as a bridge between Stata and Maxima: 1) it receives Maxima commands from `maxima` and sends them to Maxima for execution; and 2) it receives output from Maxima, which is then sent back to `maxima`.

The `maxima` command relies on Maxima Plugin, a Stata plugin written in C++ that is loaded each time `maxima` is invoked (see section 5). Maxima Plugin is the heart of maxima, enabling the TCP/IP connection with Maxima Bridge. Because Stata plugins written in C++ must be compiled, they are platform dependent. Thus the MBS provides three versions of the Maxima Plugin, one for each of the major operating systems (Linux, Mac, and Windows). Behind the scenes, `maxima` always loads the appropriate plugin for the computer system from which it has been launched.

6.1 Syntax

The syntax relating to the `maxima` command is

`maxima` [*maxima_cmd*]

The `maxima` syntax permits the user to call Maxima in two different modes: interactive and inline.

When the user types `maxima` and then presses the *Enter* key, a Maxima interactive session is opened, and everything typed thereafter is sent to Maxima for execution. When an interactive session is open, only Maxima commands can be used. Stata macros are not expanded when they are typed inside an interactive session. The special word `end` is used to exit from the interactive session and return to ordinary Stata mode.

`maxima` is invoked in inline mode when Maxima commands are typed after the word `maxima`. These commands are sent to Maxima for execution; after the command output has been returned to Stata, the Stata console is once again ready to accept ordinary Stata commands. Stata macros that appear in *maxima_cmd* are expanded; thus the inline mode is useful for integrating Maxima commands into Stata ado-programs.

6.2 Examples

In this section, I discuss examples of several advanced topics relating to `maxima`. Other basic examples of the interactive and inline uses of the command were provided in section 2.

The first time `maxima` is called, it loads the Maxima Plugin version that is appropriate to the operating system in use.

```
. display c(os)
Unix
. discard
. program dir
. maxima trigsimp((sin(x))^2 + (cos(x))^2);
(%o2)                                  1
(%i3)
. program dir
  ado      567  maxima
     ________
          567
maxima.maximaplugin     |maximaplugin_linux.plugin|
```

Maxima manages a system variable called `linel`, which contains the width (in characters) of the console display. The default value is 79, but when `maxima` is initially loaded, it automatically sets `linel` to the current value of `c(linesize)` (see [P] **creturn**) to ensure the maximum exploitation of the width of the Results window of the Stata GUI.

The `linel` value used to log all the examples presented here is 79. This value enables one to exploit all the horizontal space allowed by the formatting style of the *Stata Journal*.

The user can change the `linel` value, as is shown in the following example:

```
. discard
. display c(linesize)
104
. maxima linel;
(%o4)                                                  104
(%i5)
. maxima %pi, bfloat, fpprec: 300;
(%o5) 3.14159265358979323846264338327950288419716939937510582097494459230781640
> 62862089986280348253421170679821480865132823066470938446095505822317253594081284811174502841\
027019385211055596446229489549303819644288109756659334461284756\
> 48233786783165271201909145648566923460348610454326648213393607260249141\
27b0
(%i6)
. maxima linel: 79;
(%o6)                                  79
(%i7)
. maxima %pi, bfloat, fpprec: 300;
(%o7) 3.14159265358979323846264338327950288419716939937510582097494459230781\
64\
062862089986280348253421170679821480865132823066470938446095505822317253594\
081\
284811174502841027019385211055596446229489549303819644288109756659334461284\
756\
482337867831652712019091456485669234603486104543266482133936072602491412\
7b0
(%i8)
```

```
. maxima linel: 40;
(%o8)              40
(%i9)
. maxima %pi, bfloat, fpprec: 300;
(%o9) 3.14159265358979323846264338327950\
28841971693993751058209749445923078164\
06286208998628034825342117067982148086\
13282306647093844609550582231725359408\
28481117450284102701938521105559644622\
48954930381964428810975665933446128475\
48233786783165271201909145648566923460\
48610454326648213393607260249141270b0
(%i10)
```

When used in interactive mode, `maxima` does not allow expansion of macros; the latter are expanded only if they appear in the command string passed to `maxima` when `maxima` is invoked in inline mode.

```
. global test "mess"
. maxima
— maxima (type end to exit) ——————————————————————————————
 . "this is a $test"
(%o10)      this is a $test
(%i11)
 . end
——————————————————————————————————————————————————————————

. maxima "this is a $test";
(%o11)       this is a mess
(%i12)
```

7 The maximaget command

The `maximaget` command enables the transfer of data from Maxima to Stata. The retrieved data can be Maxima scalars, lists, or matrices. `maximaget` can put the transferred data into Stata scalars, matrices, or variables.

The name of the Stata object, which will be created or replaced, can be omitted in the syntax. In this case, `maximaget` attempts to use the name of the retrieved Maxima data for that of the Stata target data.

Unless otherwise specified (see below for the `toscalar`, `tomatrix`, and `tovar` options), a Maxima scalar transfers to a Stata scalar, while a Maxima list or matrix transfers to a Stata matrix.

7.1 Syntax

The syntax relating to the `maximaget` command is

`maximaget` *maxima_name* [*if*] [*in*] [`,` `fromscalar` `fromlist` `frommatrix` `toscalar` `tomatrix` `tovar` `name(`*stata_name*`)` `replace`]

maxima_name specifies the Maxima data to be transferred from Maxima to Stata. These data can be scalars, lists, or matrices. If `name()` is unspecified, `maximaget` attempts to use *maxima_name* as the name for the target Stata data. In this case, *maxima_name* must be a valid Stata name (see [U] **11.3 Naming conventions**).

`if` and `in` qualifiers are allowed only when importing data from Maxima to Stata variables. These options can therefore be used only when the `tovar` option is specified

7.2 Options

`fromscalar`, `fromlist`, and `frommatrix` specify the data type that `maximaget` is requested to retrieve from Maxima; at most, one of the three options is allowed. When one of `fromscalar`, `fromlist`, and `frommatrix` is specified, `maximaget` tries to retrieve data from Maxima only if data are scalars, lists, or matrices, respectively. If the data are not of the specified types, an error message is returned. If any of these three options are unspecified, `maximaget` attempts to retrieve the requested data provided the data are scalars, lists, or matrices. Specifying one of these is recommended when using `maximaget` within an ado-program because it protects against the risk of retrieving undesired data. However, when `maximaget` is used in interactive mode, the possibility of omitting the specification of one of these options lightens the syntax, and the user can immediately check whether something is amiss.

`toscalar`, `tomatrix`, and `tovar` specify the type of target Stata data; a maximum of one of these three options is permitted. `toscalar` constricts `maximaget` to import the Maxima data into Stata scalars, `tomatrix` permits importation of data into Stata matrices, and `tovar` permits importation of data into Stata variables. If any of these three options are unspecified and `maximaget` is requested to import a Maxima scalar, `maximaget` will attempt to transfer it into a Stata scalar. If any of these three options are unspecified and `maximaget` is requested to import a Maxima list or matrix, `maximaget` will attempt to transfer it into a Stata matrix.

`name(`*stata_name*`)` specifies the name of the Stata data (scalar, matrix, or variable) that will contain the data retrieved from Maxima. When this option is unspecified, `maximaget` will try to use *maxima_name* to name the target Stata data. *stata_name* must follow the Stata naming conventions.

`replace` replaces an existing variable with data retrieved from Maxima. If this option is unspecified, `maximaget` does not allow the contents of an existing variable to be replaced by data retrieved from Maxima. `replace` can be used only when the `tovar` option is specified.

7.3 Examples

In the following example, a list called `mlist` has been created in Maxima and then imported into Stata by `maximaget`; because the target type is unspecified and the `name()` option is not used, the data have been inserted into a Stata matrix called `mlist` by default.

```
. maxima mlist: [11, 22, 33, 44];
(%o1)                          [11, 22, 33, 44]
(%i2)
. maximaget mlist
(%o2)                                  done
(%i3)
. matrix list mlist
mlist[1,4]
    c1  c2  c3  c4
r1  11  22  33  44
```

In the next example, the same list has been retrieved from Maxima, and the `name()` option is used to specify `mymatrix` as the name of the target Stata matrix.

```
. maximaget mlist, name(mymatrix)
(%o3)                                  done
(%i4)
. matrix list mymatrix
mymatrix[1,4]
    c1  c2  c3  c4
r1  11  22  33  44
```

The following instruction reattempts to retrieve `mlist` from Maxima by using the `fromscalar` option. An error message is returned because `mlist` is a list and not a scalar.

```
. maximaget mlist, fromscalar
(%o4)                                  false
(%i5)
File open error (-601). Probably data does not exist in Maxima or is not of the
>  specified type
r(198);
```

To load `mlist` into a Stata variable, we create a dataset with four observations and use `maximaget` with the `tovar` option.

```
. set obs 4
obs was 0, now 4
. maximaget mlist, tovar name(var1)
(%o5)                                   done
(%i6)
. list
```

	var1
1.	11
2.	22
3.	33
4.	44

The following instructions provide an example of the use of `in` together with the `tovar` option:

```
. set obs 5
obs was 4, now 5
. maximaget mlist in 2/5, tovar name(var2)
(%o6)                                   done
(%i7)
. list
```

	var1	var2
1.	11	.
2.	22	11
3.	33	22
4.	44	33
5.	.	44

8 The maximaput command

The `maximaput` command transfers numerical data from Stata to Maxima. The transferred Stata data can be scalars, matrices, or variables, and they can be exported to Maxima scalars, matrices, or lists. If the name of the Maxima object that will contain the transferred data is unspecified, `maximaput` will attempt to use the name of the source Stata object. In this case, the data name must also be valid for Maxima according to its naming conventions.

The type of the target data can be specified by `toscalar`, `tomatrix`, or `tolist`. If none of these options have been specified, `maximaput` behaves according to the following rules: if the source Stata data are scalars, the target Maxima data are scalars; if the source Stata data are matrices, the target Maxima data are matrices; and if the source Stata data are variables, the target Maxima data are lists. It is possible to use Stata

matrices as source data and Maxima scalars as target data. In this case, the target scalar will contain the very first element of the specified matrix. Analogously, if the source is a Stata variable and the target is a Maxima scalar, the target object will contain the first observation of the Stata variable. If the source is a matrix and the target is a list, the latter will contain the elements of the matrix concatenated by rows.

8.1 Syntax

The syntax relating to the `maximaput` command is

`maximaput` *stata_name* [*if*] [*in*] [`,` `fromscalar` `frommatrix` `fromvariable` `toscalar` `tolist` `tomatrix` `name(`*maxima_name*`)`]

stata_name specifies the Stata data to be transferred from Stata to Maxima. These data can be scalars, matrices, or variables. If the `name()` option is not specified, `maximaput` will attempt to use *stata_name* as the name of the target Maxima data. In this case, *stata_name* must be a valid Maxima name.

`if` and `in` are permitted only when exporting a Stata variable. They can be used to limit the observations requested for export. `if` is used to specify a condition, while `in` permits the range to be specified.

8.2 Options

`fromscalar`, `frommatrix`, and `fromvariable` specify the data type that `maximaput` is requested to export to Maxima. At most, one of these three options is permitted. When one of `fromscalar`, `frommatrix`, and `fromvariable` is specified, `maximaput` will attempt to export the specified Stata data only if they are scalars, matrices, or variables, respectively. If *stata_name* is not of the specified types, an error message will be returned. If any of these three options are unspecified, `maximaput` will try to export the requested data, provided they exist. Specifying one of these options is recommended when `maximaput` is used within an ado-program because this syntax option prevents the exporting of undesirable data. However, when `maximaput` is used in interactive mode, the possibility of omitting the specification of one of these options lightens the syntax, and the user can immediately check whether anything is amiss.

`toscalar`, `tolist`, and `tomatrix` specify the type of the Maxima target data. At most, one of these three options is allowed. `toscalar` forces `maximaput` to export the specified Stata data to Maxima scalars; `tolist` forces the exporting of data to Maxima lists; and `tomatrix` compels the exporting of data to Maxima matrices. If any of these three options are unspecified, `maximaput` will export data to Maxima scalars, matrices, or lists if the source data are Stata scalars, matrices, or variables, respectively.

`name(`*maxima_name*`)` specifies the name of the Maxima data (scalar, list, or matrix) containing the data exported from Stata. When this option is unspecified, `maximaput` will attempt to use *stata_name* to name the target Maxima data. *maxima_name* must be a valid Maxima name according to Maxima naming conventions.

8.3 Examples

In the following example, a matrix is created in Stata and then exported to Maxima:

```
. matrix M = (1, 2 \ 3, 4 \ 5, 6)
. maximaput M
                                [ 1.0  2.0 ]
                                [          ]
(%o1)                           [ 3.0  4.0 ]
                                [          ]
                                [ 5.0  6.0 ]
(%i2)
```

Because the `name()` option was not used and the target type was unspecified, the data were exported to a Maxima matrix with the same name as the Stata matrix.

In the following example, the `tolist` option is used to export the same matrix to a list:

```
. maximaput M, tolist
(%o2)                  [1.0, 2.0, 3.0, 4.0, 5.0, 6.0]
(%i3)
```

The next example shows how to use `maximaput` to export the contents of a variable to a Maxima list by using the `if` and `name()` options.

```
. sysuse auto, clear
(1978 Automobile Data)
. maximaput price if price > 12000, name(x)
(%o3)           [14500.0, 15906.0, 13594.0, 13466.0, 12990.0]
(%i4)
```

The user can export a scalar to a matrix. In this case, the target data will be in a 1×1 matrix, in which the unique element is the value of the exported scalar.

```
. scalar stscalar = 21.3
. maximaput stscalar, tomatrix name(mscalar)
(%o4)                              [ 21.3 ]
(%i5)
. maxima matrixp(mscalar)
(%o5)                                true
(%i6)
```

If a variable and a scalar have the same name in Stata and `maximaput` is asked to export the data with this name, the command resolves this ambiguity by exporting the variable, unless the `fromscalar` option has been specified.

```
. clear
. set obs 5
obs was 0, now 5
. generate mydata = 1234
. scalar mydata = 5.6
. maximaput mydata
(%o6)              [1234.0, 1234.0, 1234.0, 1234.0, 1234.0]
(%i7)
. maxima listp(mydata)
(%o7)                                 true
(%i8)
. maximaput mydata, fromscalar
(%o8)                                  5.6
(%i9)
. maxima scalarp(mydata)
(%o9)                                 true
(%i10)
```

9 The MBS utilities package

The MBS utilities package provides these Maxima functions: `stataf()`, `statafm()`, `statamissing()`, `killtemp()`, and `wxmconnect()`. They are all written in Lisp and can be used only if this package is loaded in Maxima. When Maxima Bridge launches Maxima, it forces Maxima to load the MBS utilities package in the background, thus ensuring that all the functions of the package are available in MBS. In this section, I briefly present all the functions of the MBS utilities package.

9.1 stataf()

The `stataf()` function can be used to define a new Maxima function called "derived function" on the basis of another function called "base function". The purpose of the derived-function is to wrap the base function to slightly modify its behavior so that the floating-point number corresponding to the Stata system missing value (`.`) is returned when a call to the function generates an error (for example, an error generated by a division by zero or an overflow).

`stataf()` requires two arguments. The first is the name of the derived function, and the second is a list of the arguments of the base function. The following example clarifies how `stataf()` can be used:

```
. maxima divide(x, y) := x/y;
                                                x
(%o1)                           divide(x, y) := -
                                                y
(%i2)
. maxima res: divide(3, 0);
expt: undefined: 0 to a negative exponent.
#0: divide(x=3,y=0)
 -- an error. To debug this try: debugmode(true);
(%i3)
. maxima st_divide(x, y) := stataf(divide, [x, y]);
(%o3)              st_divide(x, y) := stataf(divide, [x, y])
(%i4)
. maxima res: st_divide(3, 0);
(%o4)                        8.9884656743115795E+307
(%i5)
. maximaget res
(%o5)                                  done
(%i6)
. scalar list res
       res =          .
```

9.2 statafm()

The `statafm()` function is very similar to `stataf()`, requiring the same arguments as `stataf()` (see section 9.1). The only difference is that `statafm()` returns the floating-point number corresponding to the Stata system missing value (`.`) even when only one of the values passed as arguments of the function is a missing value.

The following example illustrates the use of `statafm()`:

```
. maxima sum(x, y) := x + y;
(%o1)                          sum(x, y) := x + y
(%i2)
. maxima stm_sum(x, y) := statafm(sum, [x, y]);
(%o2)                stm_sum(x, y) := statafm(sum, [x, y])
(%i3)
. scalar m = .g
. maximaput m, fromscalar
(%o3)                        9.0038268217042019E+307
(%i4)
. maxima res: stm_sum(5, m);
(%o4)                        8.9884656743115795E+307
(%i5)
. maximaget res
(%o5)                                  done
(%i6)
. scalar list res
       res =          .
```

9.3 statamissing()

The `statamissing()` function is used to generate floating-point numbers corresponding to Stata missing values (see [U] **12.2.1 Missing values**). The only argument of this function is a string containing the name of the requested missing value.

In the following example, `statamissing()` is used to generate the "`.g`" missing value and the "`.`" system missing value:

```
. maxima missing_g: statamissing("g");
(%o1)                       9.0038268217042019E+307
(%i2)
. maxima missing_system: statamissing(".");
(%o2)                       8.9884656743115795E+307
(%i3)
. maximaget missing_g
(%o3)                                  done
(%i4)
. maximaget missing_system
(%o4)                                  done
(%i5)
. scalar list missing_g missing_system
 missing_g =         .g
missing_system =          .
```

9.4 killtemp()

Temporary names and local macros are used in Stata, especially in ado-programs, to avoid conflicts with global data. Similarly, MBS provides a system for creating temporary data in Maxima. This system is based on the convention of using names starting with an underscore (_) to mark Maxima data as temporary (for example, "`_mymaximadata`"). Once temporary Maxima data are correctly named, they can be automatically deleted using `killtemp()`. The `killtemp()` function deletes all Maxima data that have a name beginning with an underscore. The function does not require an argument.

`killtemp()` can be used in ado-Maxima programs in which creating temporary Maxima data can be useful. An important difference with the analogous system used in Stata to manage temporary objects is that temporary objects in MBS are not deleted automatically and must be explicitly deleted by invoking `killtemp()`. A call to `killtemp()` is required in the ado-program to delete all the temporary data that have been created thus far.

Several ado-Maxima programs in section 10 are examples using the `killtemp()` function. The following example demonstrates the basic use of this function:

```
. maxima a: 1$ _b: 2$ _c: 3$ d: 4$ _e: 5$
(%i6)
. maxima values;
(%o6)                          [a, _b, _c, d, _e]
(%i7)
. maxima killtemp();
(%o7)                                done
(%i8)
. maxima values;
(%o8)                               [a, d]
(%i9)
```

9.5 wxmconnect()

The `wxmconnect()` function connects the Maxima process to wxMaxima, a free and open-source front end for Maxima, and allows the user to take advantage of the GUI of wxMaxima and interact with Maxima in a more comfortable environment. After the Maxima process is connected to wxMaxima, `wxmconnect()` can be used to reconnect the Maxima process to Maxima Bridge. The only argument required by `wxmconnect()` is the network port of the software (wxMaxima or Maxima Bridge) to which the Maxima process will be connected.

The use of `wxmconnect()` and its technical details are explained more thoroughly in the online help for Maxima Bridge.

10 Applications

In this section, I present several examples of using MBS for practical applications. These applications include many features made available by Maxima: differentiation calculus, linear programming, arbitrary numerical precision, and 3D plotting. All the proposed examples are intentionally simple because the goal here is to introduce various potentialities that can arise when using Stata and Maxima together.

10.1 Calculating the moments of a random normal variable

To obtain the moments of a random normal variable, one can exploit its moment-generating function (MGF). The MGF of a normal variable with expected value μ and standard deviation σ is

$$M(t) = e^{\mu t} e^{\frac{1}{2}\sigma^2 t^2} \tag{1}$$

The derivative of order r of (1), with respect to t and evaluated at $t = 0$, is the noncentral moment of order r of the normal variable with parameters μ and σ.

Typically, software that specializes in numerical computation, such as Stata, does not provide algorithms for analytical differentiation. However, we can use Maxima to calculate moments of a normal variable via the differentiation of its MGF. The following is the code of an `egen` function called `_gnormalmoment`. This function accepts expected values and standard deviations of normal variables as arguments, and it generates a new Stata variable containing the corresponding values of the r-order moments.

```
. program _gnormalmoment
 1. gettoken vartype 0 : 0
 2. gettoken newvar 0 : 0
 3. gettoken equalsign 0: 0
 4. syntax varlist(numeric min=2 max=2) [if] [in], r(integer)
 5. local mu : word 1 of `varlist´
 6. local sd : word 2 of `varlist´
 7. marksample touse
 8. quietly {
 9. maxima killtemp();
10. maximaput `mu´ if `touse´, fromvar tolist name(_mulist)
11. maximaput `sd´ if `touse´, fromvar tolist name(_sdlist)
12. maxima _G(t, mu, sd) := exp(mu*t)*exp((1/2)*(sd^2)*(t^2));
13. maxima _gdiff: diff(_G(´t, ´mu, ´sd), ´t, `r´);
14. maxima _m(mu, sd) := ´´(at(_gdiff, t=0));
15. maxima _st_m(mu, sd) := stataf(_m, [mu, sd]);
16. maxima _res: map(_st_m, _mulist, _sdlist);
17. maximaget _res if `touse´, fromlist tovar name(`newvar´)
18. maxima killtemp();
19. }
20. end
```

Lines 1 through 6 parse the syntax. The `marksample` command, which is used in line 7, avoids the transfer of missing values to Maxima. The remaining lines are MBS commands, which are enclosed between two calls to `killtemp()`. As stated in section 9.4, `killtemp()` deletes all existing temporary Maxima variables whose names start with an underscore. The first invocation to `killtemp()` avoids the risk of conflict with possibly existing temporary data, while the second deletes temporary data that have been created in `_gnormalmoment`. Because every Maxima object created in `_gnormalmoment` is temporary datum, no garbage data remain in the Maxima memory after the above program is executed.

Line 12 defines the MGF function (1), and line 13 specifies its r-order derivative. In this latter line, the Maxima quote operator (‘) is used before several Maxima symbols to prevent their evaluation. The `_m` function, which is defined in line 14, evaluates the r-order derivative of the MGF at $t = 0$. In line 15, the `_st_m` function is created as a wrapper to `_m` by using the `stataf()` function from the MBS utilities package. `stataf()` ensures that a Stata system missing value (`.`) is returned whenever a call to `_m` generates an error. Finally, line 16 creates a list containing the calculated values of the moments, and line 17 imports these values to a new variable generated in Stata.

The following is an example using `_gnormalmoment`, in which order-two moments are calculated for a set of five observations with randomly generated expected values and standard deviations:

```
. clear all
. set obs 5
obs was 0, now 5
. set seed 9999
. generate mu = runiform()
. generate sd = runiform()
. egen m2 = normalmoment(mu sd), r(2)
. list
```

	mu	sd	m2
1.	.5706419	.4555794	.53318483
2.	.7122409	.4310627	.6931022
3.	.9121964	.7898812	1.4560145
4.	.3174115	.1223136	.11571071
5.	.8693877	.8267023	1.4392716

10.2 Nondiscretionary input in data envelopment analysis

Data envelopment analysis (DEA) is a nonparametric, quantitative method that assesses the relative efficiency in how decision-making units (DMUs)—firms, schools, and hospitals—transform input into output (see Charnes, Cooper, and Rhodes [1978] and Cook and Seiford [2009]). The main DEA output is a list of efficiency scores, which are numbers between zero and one that indicate the performance of each DMU. A DMU is deemed efficient when its efficiency score is one.

DEA analysis requires the solution to a linear program, namely, the mathematical problem of optimizing a linear function that is subject to linear equality and linear inequality constraints. Ji and Lee (2010) provide the `dea` Stata command for performing basic DEA analysis, but this command is not as flexible as the linear programming solver provided by Maxima. For example, the `dea` command cannot manage a case with various inputs that are nondiscretionary; that is, it cannot be used when some of the inputs are beyond the control of the DMUs (for example, the location of firms or the advertising expenditure decided at the corporate level) (Banker and Morey 1986).

In this section, I explain how to write a rudimentary DEA solver program, called `fdea`, that can handle the case when one of the inputs is nondiscretionary. This program uses MBS and exploits the linear programming solver of Maxima. The `fdea` command is intentionally simple because it is presented only for explanatory purposes. Thus it is provided with certain limitations: 1) only two discretionary inputs, one nondiscretionary input and one output, are permitted; 2) only first-stage DEA analysis is performed, so efficiency slacks are not identified; 3) `if` and `in` options are not allowed; and 4) if the command passes Stata missing values to Maxima, they will not be correctly interpreted by Maxima.

Let n be the number of DMUs; x_{d1i}, x_{d2i}, x_{fi}, and y_i denote the first discretionary input, the second discretionary input, the nondiscretionary input, and the output for DMU i, respectively; and t_j^* denotes the efficiency score for DMU j. The efficiency score for DMU j is obtained from the solution to the following linear program (Banker and Morey 1986).

$$
\begin{aligned}
& t_j^* = \min t \\
& \text{subject to} \\
& \sum_{i=1}^{n} l_i x_{d1i} \leq t x_{d1j} \\
& \sum_{i=1}^{n} l_i x_{d2i} \leq t x_{d2j} \\
& \sum_{i=1}^{n} l_i x_{fi} \leq x_{fj} \\
& \sum_{i=1}^{n} l_i y_i \geq y_j \\
& t, l_i \geq 0,\ i \in (1, 2, \ldots, n)
\end{aligned}
$$

This linear program is translated into the following code of the `fdea` ado-Maxima program:

```
. program fdea
  1. syntax newvarname, xd1(varname numeric) xd2(varname numeric)
> xf(varname numeric) y(varname numeric)
  2. maxima kill(all)
  3. maximaput `xd1´, fromvar tolist name(xd1)
  4. maximaput `xd2´, fromvar tolist name(xd2)
  5. maximaput `xf´, fromvar tolist name(xf)
  6. maximaput `y´, fromvar tolist name(y)
  7. maxima load(simplex); nonegative_lp: true;
  8. local n = _N
  9. maxima l: makelist(concat(l, i), i, 1, `n´);
 10. maxima scores: [];
 11. maxima for j:1 thru `n´ do                          ///
 12. (                                                   ///
 13.     cxd1: l . xd1 <= t*xd1[j],                      ///
 14.     cxd2: l . xd2 <= t*xd2[j],                      ///
 15.     cxf: l . xf <= xf[j],                           ///
 16.     cy: l . y >= y[j],                              ///
 17.     first: minimize_lp(t, [cxd1, cxd2, cxf, cy]), ///
 18.     scores: append(scores, [first[1]])              ///
 19. );
 20. maximaget scores, fromlist tovar name(`varlist´)
 21. end
```

The code lines from 3 through 6 export the 2 discretionary input variables, the nondiscretionary input variable, and the output variable, which are specified, respectively, to Maxima in the `xd1()`, `xd2()`, `xf()`, and `y()` options. Line 7 loads the simplex Maxima package, which implements the 2-phase standard simplex method for solving linear programming problems in Maxima. In the same line, the `nonegative_lp` variable is set to "true" to indicate to the solver that all decision variables are assumed to be nonnegative. The linear program is solved for each DMU in lines 11 through 19. Here the `minimize_lp` Maxima function (used in line 17) minimizes the t subject to the linear

constraints specified in the second argument of the function. Finally, line 20 exports the calculated efficiency scores to Stata.

The following is an example of the use of `fdea`, illustrated using a randomly generated dataset where the `x1` and `x2` variables are two discretionary inputs, the `x3` variable is the only nondiscretionary input, the `y` variable is the unique output, and `score` is a variable containing the efficiency scores generated by `fdea`:

```
. clear all
. set obs 5
obs was 0, now 5
. set seed 4321
. generate x1 = runiform()
. generate x2 = runiform()
. generate x3 = runiform()
. generate y = runiform()
. quietly fdea score, xd1(x1) xd2(x2) xf(x3) y(y)
. list
```

	x1	x2	x3	y	score
1.	.3195768	.6264579	.3398471	.8968329	1
2.	.8935631	.8397377	.1994603	.1145874	.09283615
3.	.6512877	.560277	.6148463	.826956	.99954164
4.	.6101616	.5916896	.0149962	.4950392	1
5.	.8236098	.1597123	.1503211	.2799092	1

The `fdea` program is a current draft of a possibly more-advanced general command that could be developed. Nevertheless, it demonstrates how easy it is to develop a DEA solver to fit a specific need. The advantage of programming a DEA solver in the MBS environment is that we can obtain a tool that is perfectly integrated into Stata. Thus the ado-Maxima DEA solver can be combined with all the facilities provided by Stata to perform subsequent statistical analyses, such as Monte Carlo experiments[5] or sensitivity analysis.

10.3 Arbitrary numerical precision

In this section, I present a case of numerical computation where Stata fails and a way to overcome this using Maxima. Suppose, for example, that the tricky expression $(e^x - 1)/x$ must be evaluated for a very small value of x.

5. A command such as `fdea` could be used in Monte Carlo experiments like those performed by Pedraja-Chaparro, Salinas-Jiménez, and Smith (1999).

```
. set obs 3
obs was 0, now 3
. set seed 9999
. generate double x = runiform()
. replace x = 2^(-150) in 2
(1 real change made)
. generate double stata = (exp(x)-1)/x
. list
```

	x	stata
1.	.57064196	1.3483105
2.	7.006e-46	0
3.	.91219641	1.6331846

The value of $(e^x - 1)/x$ for $x = 2^{-150}$, calculated in the previous example for the second observation, is 0. This numerical computation is far from the correct one, which is $\lim_{x\to 0}(e^x - 1)/x = 1$. A better result can be obtained by using Maxima with the following code:

```
. quietly maximaput x
. quietly maxima fpprec: 50;
. quietly maxima f(x) := (exp(bfloat(x))-1)/x;
. quietly maxima maxima: map(f, x);
. quietly maximaget maxima, fromlist tovar
. list
```

	x	stata	maxima
1.	.57064196	1.3483105	1.3483105
2.	7.006e-46	0	1
3.	.91219641	1.6331846	1.6331846

Here the number of significant digits for the arithmetic relating to big-float numbers was set to 50 by changing the value of the `fpprec` Maxima variable. Thereafter, the `f()` function was defined to calculate $(e^x - 1)/x$. In the definition of `f()`, an x value is converted to a big-float number via the `bfloat` function; in most cases, expressions with at least one big-float number are entirely evaluated as big-float (big-float numbers in Maxima can be conceived of as "contagious"). Thus the evaluation of `f()` will be performed to 50 digits of precision. The evaluation of $(e^x - 1)/x$ for $x = 2^{-150}$ (see the second observation in the output generated by list) is now 1, which is a more accurate result.

Note that in the previous case, 0 can also be obtained in Maxima for a smaller value of x, say, 2^{-250}. To obtain 1 again, the user must increase the value of `fpprec`; a value of 100 for `fpprec` would be sufficient.

10.4 Exploiting gnuplot

Gnuplot is a free command-line software program that creates 2D and 3D graphs of mathematical functions and data. It comes with Maxima and is one of the graphics engines used by Maxima for rendering graphs. Gnuplot is accessible from Maxima through a suite of commands that operates as a convenient interface to gnuplot. In this section, I present two useful examples of using gnuplot. The first is an ado-Maxima program for drawing 3D graphs, while the second is a program for extracting coordinates of points from a scatterplot for which the image is available but not the source data.

The following is the code of `scatter3d`, an ado-Maxima program that plots 3D graphs:

```
. program scatter3d
  1. syntax varlist(min=3 max=3 numeric) [if] [in]
  2. marksample touse
  3. local x: word 1 of `varlist´
  4. local y: word 2 of `varlist´
  5. local z: word 3 of `varlist´
  6. quietly {
  7. maxima killtemp();
  8. maximaput `x´ if `touse´, fromvar tolist name(_x)
  9. maximaput `y´ if `touse´, fromvar tolist name(_y)
 10. maximaput `z´ if `touse´, fromvar tolist name(_z)
 11. maxima load(draw);
 12. maxima draw3d(point_type=plus, color=black, xlabel="x", ylabel="y",
> zlabel="z", points(_x,_y,_z));
 13. maxima killtemp();
 14. }
 15. end
```

This program takes three numerical variables as input (representing the 3D coordinates) and plots a 3D scatterplot of these variables by using `draw3d()`, a function of the `draw` Maxima package, which is loaded in line 11. `draw3d()` is called in line 12. Although the full syntax of `draw3d()` is very sophisticated, that which is used in the previous example is easy to understand. `draw3d()` is requested to plot a scatterplot of `_x`, `_y`, and `_z` by using the symbol `plus` as the point style, choosing `black` as the color for the points, and setting the labels for all the axes.

As an example, we create the scatterplot of a simulated 3-dimensional normal variable with vector **0** as an expected value and a desired covariance matrix. Here the task of obtaining a simulated dataset of such a three-dimensional vector variable is assigned to Stata via the `drawnorm` (see [D] **drawnorm**) command.

```
. clear
. set obs 1000
obs was 0, now 1000
. matrix cov = (1, 0.7, -0.4 \ 0.7, 1, 0.2 \ -0.4, 0.2, 1)
. set seed 9999
. drawnorm x y z, cov(cov)
. scatter3d x y z
```

The 3D output, produced by gnuplot through the previous commands, is reported in figure 2, where three different views of the same point cloud have been provided from different angles. When a graph is obtained in gnuplot, the user can interact with it through the mouse to zoom or change the angle view.

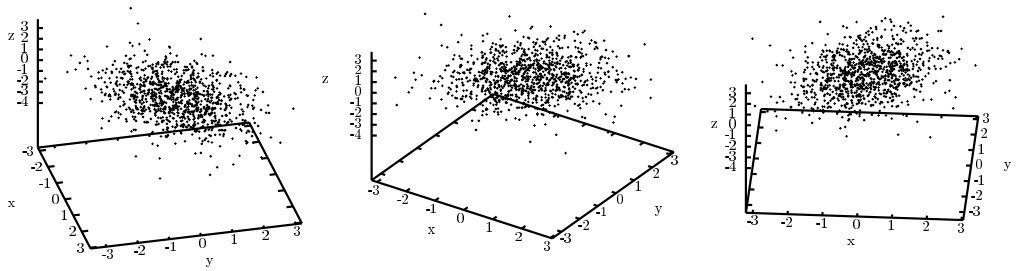

Figure 2. Three scatterplots of the same simulated three-dimensional normal variable, viewed from three different angles, obtained using gnuplot

The next example exploits an interactive feature of gnuplot, consisting of retrieving the coordinates of a point from a graph on which the user has clicked. The `takecoords` ado-Maxima program, which I present here, exploits this feature to retrieve the coordinates of the data points of a scatterplot. This program is especially useful for retrieving data from a scatterplot when the source data are not available.

The following is the code of `takecoords`:

```
. program takecoords
  1.     syntax newvarlist(min=2 max=2) using/ , x1(real) y1(real) x2(real)
> y2(real)
  2.     if _N != 0 {
  3.         display as error "you must start with an empty dataset"
  4.         exit 18
  5.     }
  6.     local xVar: word 1 of `varlist´
  7.     local yVar: word 2 of `varlist´
  8.     confirm file "`using´"
  9.     tempfile tempFileName
 10.     maxima killtemp();
 11.     maxima load(draw);
 12.     maxima _img: image(read_xpm("`using´"), `x1´, `y1´, `x2´-`x1´,
> `y2´-`y1´);
 13.     maxima draw2d(xy_file = "`tempFileName´", _img);
 14.     maxima killtemp();
 15.     local dummyInput
 16.     display "Press ENTER at the end of the interaction with the plot",
> _request(_dummyInput)
 17.     infile `xVar´ `yVar´ using "`tempFileName´"
 18. end
```

In line 12, the program reads an `.xpm` image whose position in the file system is specified through the `using` option of `takecoords`. This image is displayed by the `draw2d()` Maxima function called in line 13. Furthermore, `draw2d()` saves the coordinates of the

user's clicks on the displayed images in a temporary file specified by the `xy_file` option. In line 17, the `infile` Stata command reads the content of that temporary file and loads the coordinates into two variables that have been specified by the user.

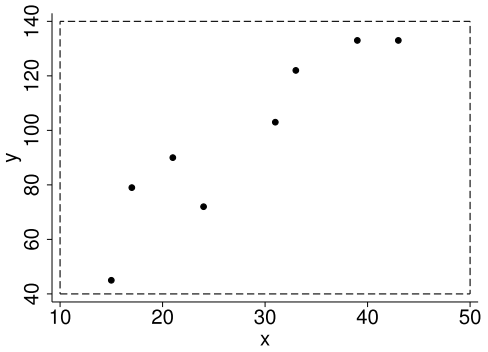

Figure 3. An example scatterplot to use with `takecoords`

Let's suppose that the data source of the scatterplot of figure 3 is unavailable and that we want to extract the data from this picture. First, we crop the figure to preserve only the part within the dashed box. This operation is necessary to correctly map the screen area provided by gnuplot with the plot area of the example scatterplot. The cropped image must be converted to the `.xpm` format. Second, `takecoords` can be called by specifying 1) the two `x` and `y` variables that will be filled with the retrieved point coordinates; 2) the complete path to the `.xpm` scatterplot image; and 3) the coordinates of the bottom-left points $(10; 40)$ and the far-right and top points $(50; 140)$ of the plot area corresponding to the dashed box in figure 3.

```
. clear
. takecoords x y using "/home/giovanni/figure3_cropped.xpm", x1(10) y1(40)
> x2(50) y2(140)
  (output omitted)
* Now the user middle-clicks on the points of the scatterplot
Press ENTER at the end of the interaction with the plot .
(8 observations read)
. list, sep(0)
```

	x	y
1.	15.10363	44.68052
2.	17.08479	78.75887
3.	21.03634	89.98685
4.	24.11575	71.63292
5.	31.07133	102.7136
6.	33.03096	121.6724
7.	39.02826	132.6637
8.	42.96904	132.9266

During the interactive session in gnuplot, the user must save the coordinates by clicking on the points of the scatterplot and then pressing the *x* key. Thereafter, the user must press the *Enter* key in Stata to inform `takecoords` that the temporary file can be read and that the `x` and `y` variables can be created.

11 Concluding remarks

MBS extends the computational capabilities of Stata in several ways:

- by enabling the computation of something that cannot easily be obtained in Stata because, for example, the computation would require algebra-intensive algorithms;
- by providing algorithms that facilitate the development of powerful computing solutions by allowing a direct application of the theory, for example, via differential and integral calculus; and
- by enabling high-precision calculations using big-float numbers.

Although a CAS can be a powerful tool for dealing with many mathematical problems, a CAS can occasionally provide incorrect computations, or it may not have the capability to undertake certain tasks. This article lacks an assessment of the capabilities of Maxima. However, the interested researcher could use the 131 mathematical problems proposed by Wester (1994) as a comprehensive CAS test suite for evaluating Maxima.

Apart from its intrinsic utility, MBS is also an example of a software system that enables Stata to communicate with another software application.[6] The interested software developer might draw inspiration from MBS to create new and powerful integrations between Stata and other software programs: MBS is open-source software, and its code can be freely examined.

12 References

Banker, R. D., and R. C. Morey. 1986. Efficiency analysis for exogenously fixed inputs and outputs. *Operations Research* 34: 513–521.

Barnett, M. P. 2002. Computer algebra in the life sciences. *ACM SIGSAM Bulletin* 36: 5–32.

Bollen, K. A., and S. Bauldry. 2010. Model Identification and Computer Algebra. *Sociological Methods and Research* 39: 127–156.

Boyle, A., and B. F. Caviness, eds. 1990. *Future Directions for Research in Symbolic Computation: Report of a Workshop on Symbolic and Algebraic Computation, April*

6. Also see Fiedler (2013) for a Stata plugin that makes the Python language available within Stata.

29–30, 1988, Washington, DC. Philadelphia: Society for Industrial and Applied Mathematics.

Charnes, A., W. W. Cooper, and E. Rhodes. 1978. Measuring the efficiency of decision making units. *European Journal of Operational Research* 2: 429–444.

Cohen, A. M., J. H. Davenport, and A. J. P. Heck. 1993. An overview of computer algebra. In *Computer Algebra in Industry*, ed. A. M. Cohen, 1–52. Chichester, UK: Wiley.

Cook, W. D., and L. M. Seiford. 2009. Data envelopment analysis (DEA)—Thirty years on. *European Journal of Operational Research* 192: 1–17.

Dodson, C. T. J. 2004. Review of Maple 9.5. *MSOR Connections* 4: 1–2.

Fiedler, J. 2013. python: Stata module for using the Python language within Stata. Statistical Software Components S457688, Department of Economics, Boston College. http://econpapers.repec.org/software/bocbocode/s457688.htm.

Glen, A. G., D. L. Evans, and L. M. Leemis. 2001. APPL: A probability programming language. *American Statistician* 55: 156–166.

Goedman, R., G. Grothendieck, S. Højsgaard, and A. Pinkus. 2012. *Ryacas: R interface to the Yacas computer algebra system*. R package version 0.2-11. http://cran.r-project.org/web/packages/Ryacas/index.html.

Grabmeier, J., E. Kaltofen, and V. Weispfenning, eds. 2003. *Computer Algebra Handbook: Foundations, Applications, Systems*. New York: Springer.

Grothendieck, G. 2012. *rSymPy: R Interface to SymPy Computer Algebra System*. R package version 0.2-1.1.
http://cran.r-project.org/web/packages/rSymPy/index.html.

Hoyle, R. H., ed. 2012. *Handbook of Structural Equation Modeling*. New York: Guilford Press.

IEEE. 2008. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008* 1–70.

Ji, Y.-B., and C. Lee. 2010. Data envelopment analysis. *Stata Journal* 10: 267–280.

Joyner, D. 2006. OSCAS: maxima. *ACM Communications in Computer Algebra* 40: 108–111.

Li, J., and J. S. Racine. 2008. Maxima: An open source computer algebra system. *Journal of Applied Econometrics* 23: 515–523.

Linhart, J. M. 2008. Mata Matters: Overflow, underflow and the IEEE floating-point format. *Stata Journal* 8: 255–268.

Pedraja-Chaparro, F., J. Salinas-Jiménez, and P. Smith. 1999. On the quality of the data envelopment analysis model. *Journal of the Operational Research Society* 50: 636–644.

R Development Core Team. 2014. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org.

Rose, C. 2007. "Oh Zeus, free me!"—Teaching mathematical statistics with Mathematica/mathStatica. Proceedings of the 56th Session of the International Statistical Institute. Lisbon, Portugal.

Rose, C., and M. D. Smith. 2000. Symbolic maximum likelihood estimation with Mathematica. *Journal of the Royal Statistical Society, Series D* 49: 229–240.

———. 2002. *Mathematical Statistics with Mathematica*. New York: Springer.

———. 2005. Computational order statistics. *Mathematica Journal* 9: 790–802.

Stokes, B. 2012. mathStatica 2.5. *Journal of Statistical Software* 47: 1–12.

Vinod, H. D. 2003. Review of mathStatica (V.1): An add-on to Mathematica. *Journal of Applied Econometrics* 18: 485–491.

Wester, M. 1994. A review of CAS mathematical capabilities. *Computer Algebra Nederland Nieuwsbrief* 13: 41–48.

Wolfram, S. 2003. *The Mathematica Book*. 5th ed. Champaign, IL: Wolfram Media.

Zeilberger, D. 2004. Symbolic moment calculus I: Foundations and permutation pattern statistics. *Annals of Combinatorics* 8: 369–378.

About the author

Giovanni Luca Lo Magno created stand-alone software that extends the capabilities of Stata. He is the author of Stata Automatic Report, which is a macro for Microsoft Word for Windows that allows the integration between Stata and Word. He is also interested in researching gender differences in employment.