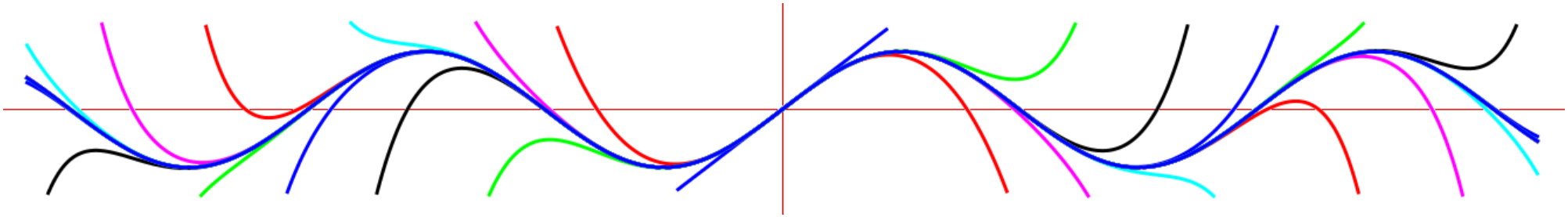
X Italian Stata User Group Meeting

*Florence, 14-15 November 2013*

# Maxima Bridge System: a Software Interface between Stata and the Maxima Computer Algebra System



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

# What is a computer algebra system (CAS)?

A CAS is a software program which enables the user to manipulate mathematical expression in a symbolic form

**Useful for dealing with:**

- equations
- derivatives of functions
- integrals
- arbitrary numerical precision

# Maxima: a free and open-source CAS

- highly featured CAS
- programmable and extendible
- written in LISP
- runs in Windows, Mac, Unix and GNU/Linux
- downloadable from *http://maxima.sourceforge.net*

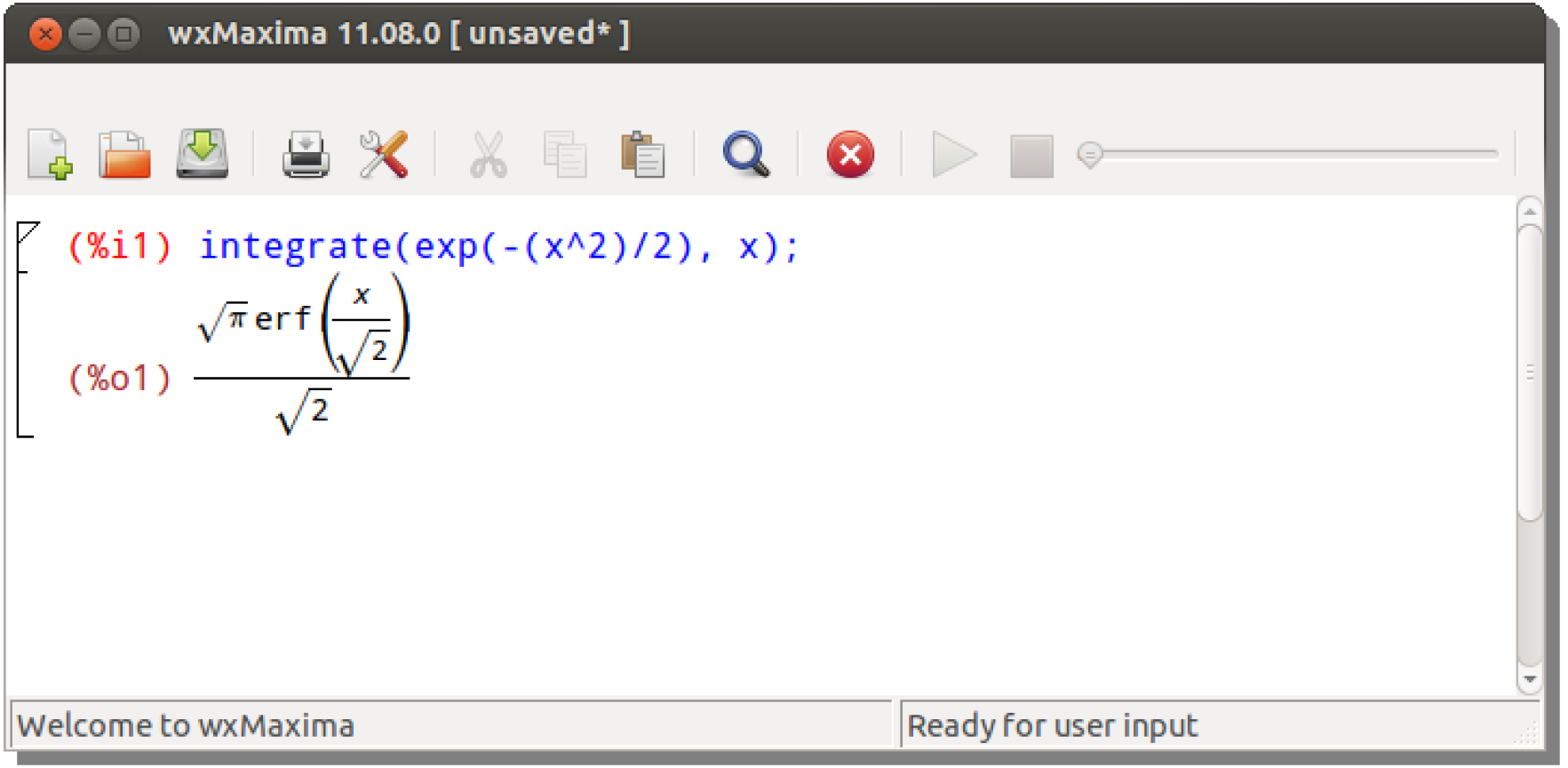
# Using Maxima from the terminal

Maxima is at heart a command-line software program

giovanni@giovanni-ubuntupc: ~

```
giovanni@giovanni-ubuntupc:~$ maxima

Maxima 5.24.0 http://maxima.sourceforge.net
using Lisp GNU Common Lisp (GCL) GCL 2.6.7 (a.k.a. GCL)
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
(%i1) integrate(exp(-(x^2)/2), x);
                                                 x
                                 sqrt(%pi) erf(-------)
                                               sqrt(2)
(%o1)                            ----------------------
                                        sqrt(2)
(%i2)
```

# wxMaxima: a graphical user interface (GUI) for Maxima




Downloadable for free from *http://andrejv.github.io/wxmaxima*

# Numerical computations *vs.* symbolic computations

## Calculating *arcsin(1)*

### Stata

```
. display asin(1)
1.5707963
```

### Maxima

```
(%i1) asin(1);
                      %pi
(%o1)                 ---
                       2
```

# Numerical computations *vs.* symbolic computations

## Obtaining the decimal expansion for $\pi$

### Stata

```
.  display %26.24f _pi
3.14159265358979311599796 3
```

### Maxima

```
(%i1) %pi, bfloat, fpprec: 300;
(%o1) 3.14159265358979323846264338327950288419716939937510\
5820974944592307816406286208998628034825342117067982148086\
5132823066470938446095505822317253594081284811174502841027\
0193852110555964462294895493038196442881097566593344612847\
5648233786783165271201909145648566923460348610454326648213\
39360726024914127b0
```

# Numerical computations *vs.* symbolic computations

## Verifying the inverse of a Hilbert (1894) matrix

### Stata

```
. matrix A = 1, 1/2 \ 1/2, 1/3
. matrix I = A * invsym(A)
. matrix list I
symmetric I[2,2]
             r1           r2
r1            1
r2  -3.708e-17            1
. display C[2,1] == 0
0
```

### Maxima

```
(%i1) A: matrix([1, 1/2], [1/2, 1/3]) $

(%i2) I: A .  invert(A);
                [ 1   0 ]
(%o2)           [       ]
                [ 0   1 ]
(%i3) is(I[2,1]=0);
(%o3)             true
```

# What is Maxima Bridge System (MBS)?

MBS is a software system which enables Stata to interface with Maxima

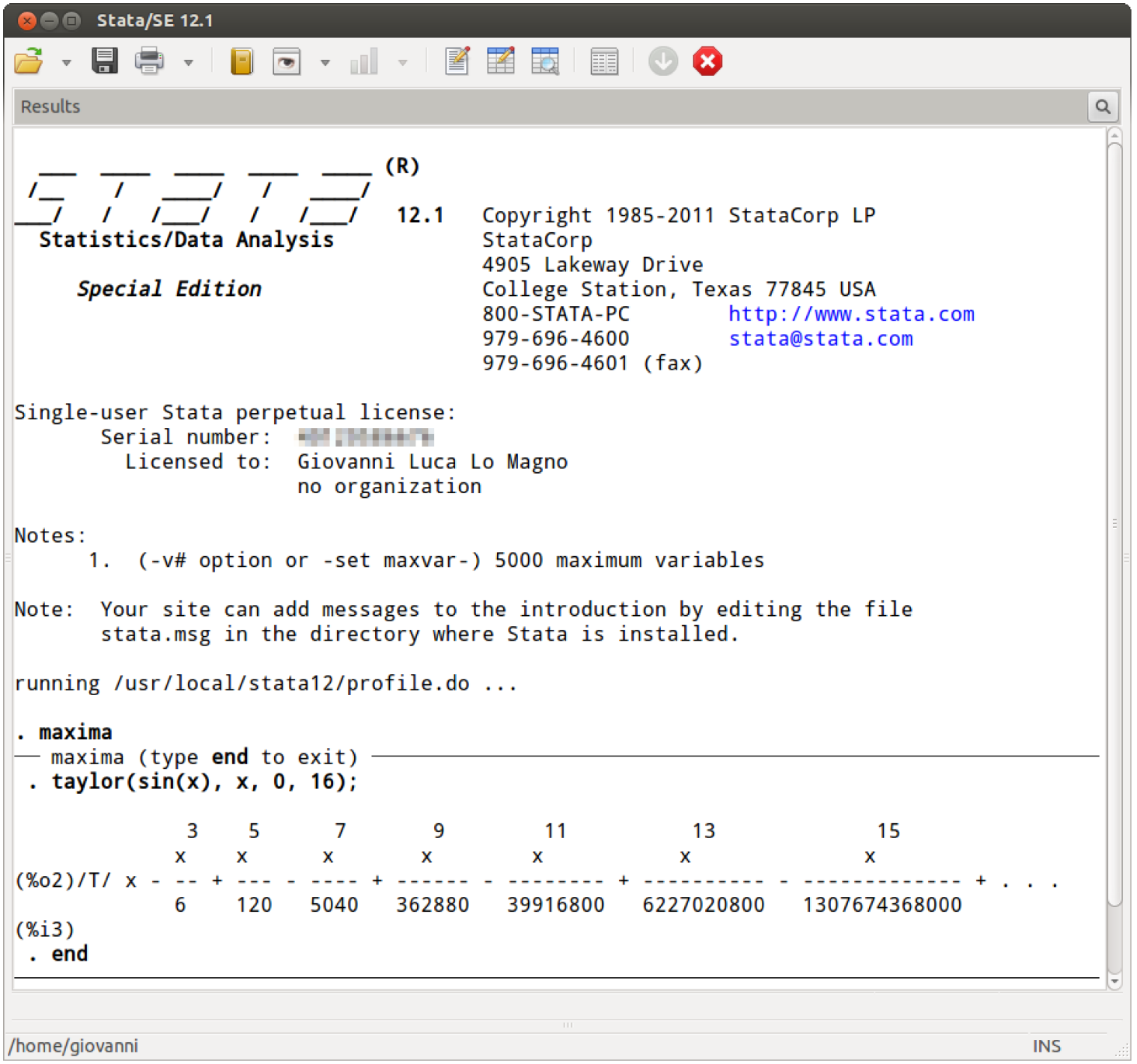
**MBS permits:**

- the use of Maxima from Stata
- the writing of ado-program incorporating Maxima commands

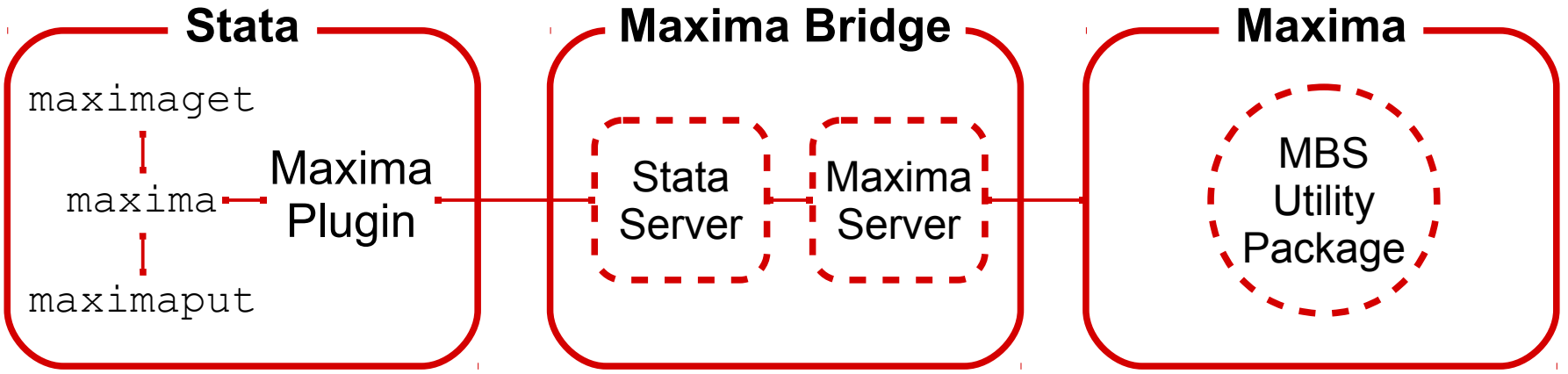
**MBS comprises:**

- a Stata plugin
- ado-programs (`maxima`, `maximaget`, `maximaput`)
- a stand-alone software program (Maxima Bridge)
- Maxima functions (MBS Utility Package)

# Calling Maxima from Stata



# MBS architecture




- **Maxima Plugin**: manages the communication with Maxima
- **`maxima`**: a wrapper for the Stata Plugin
- **`maximaget`**: imports data from Maxima
- **`maximaput`** exports data to Maxima

- **Stata Server**: communicates with Stata
- **Maxima Server**: communicates with Maxima

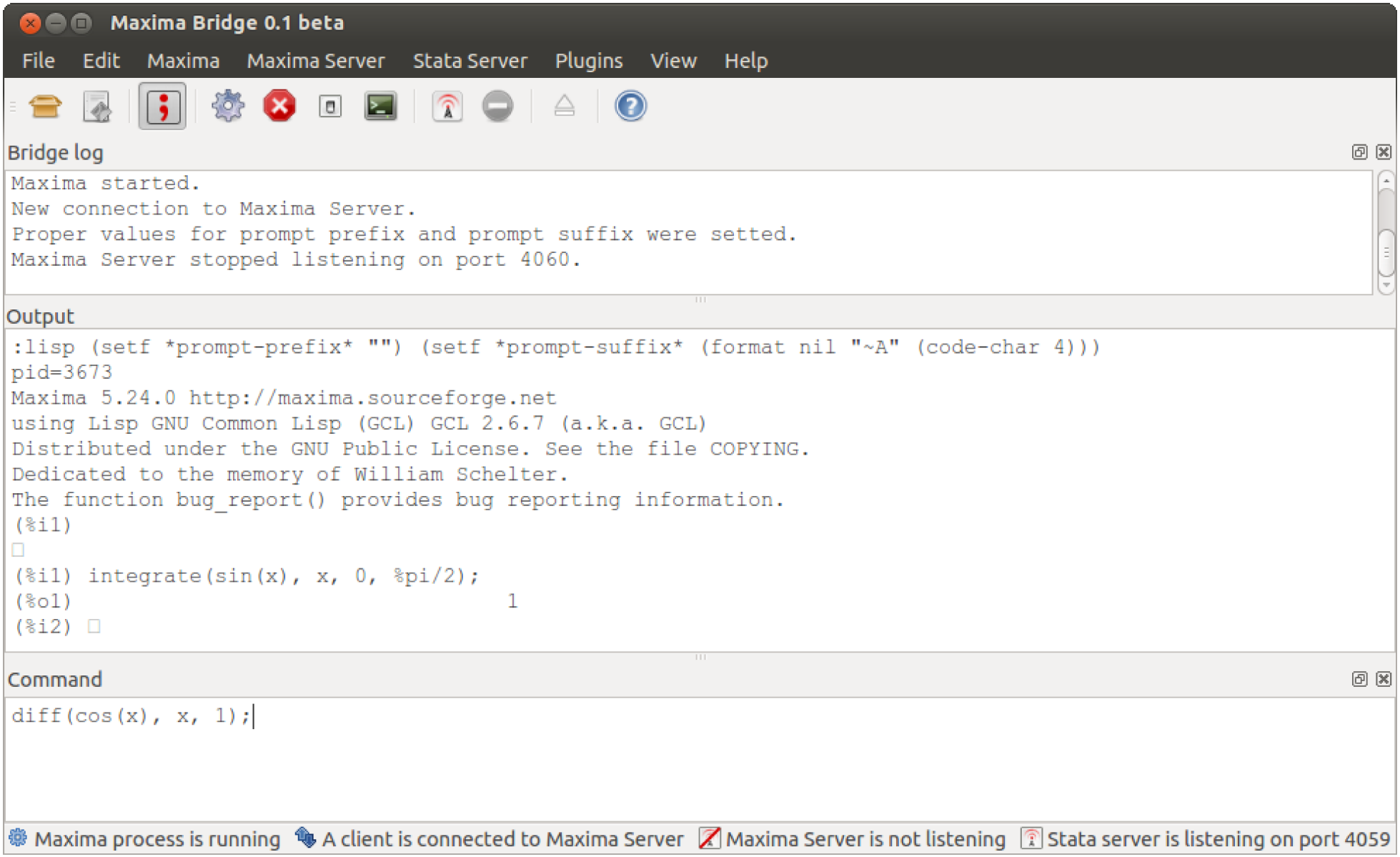
- **MBS Utility Package**: Maxima functions that facilitate the cooperation between Stata and Maxima

# Maxima Bridge

A **bridge** between Stata and Maxima


- Connects Stata and Stata by using a local TCP/IP network
- Essential for the communication between Stata and Maxima
- Stand-alone software program
- Written in C++

# A screenshot of Maxima Bridge 0.1 beta



# Using Maxima from Stata: the `maxima` command

## Interactive mode

```
. maxima
-- maxima (type end to exit) --------------------------------
 . integrate(sqrt(x), x, 0, 1);

                                  2
(%o1)                             -
                                  3

(%i2)
 . end
-------------------------------------------------------------
```

## Inline mode

```
. maxima integrate(sqrt(x), x, 0, 1);

                                  2
(%o2)                             -
                                  3

(%i3)
```

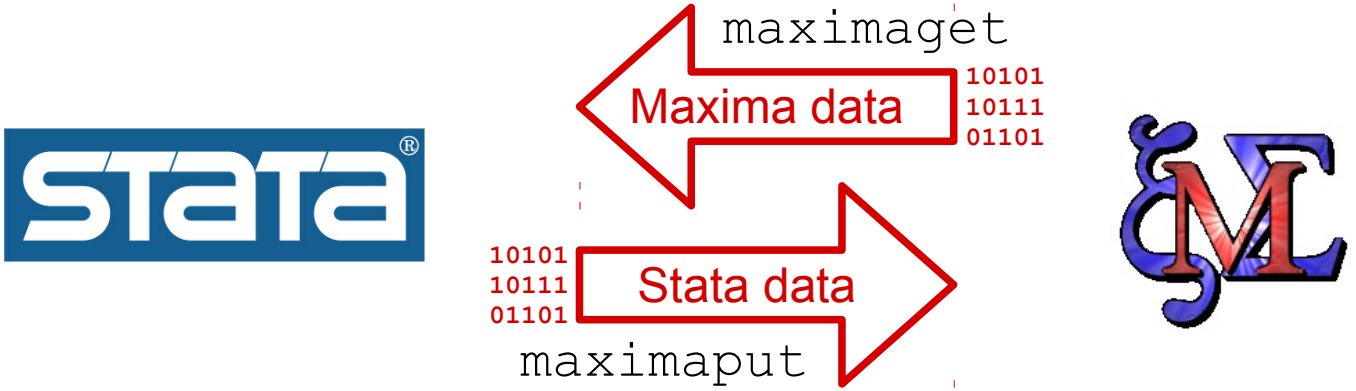
# Macro expansion when the `maxima` command is called

Macro expansion only works in *inline* mode

```
. global test "mess"

. maxima

-- maxima (type end to exit) -------------------------------
 . print("this is a $test");

this is a $test
(%o1)                    this is a $test
(%i2)
 . end
-----------------------------------------------------------

. maxima print("this is a $test");

this is a mess
(%o2)                     this is a mess
(%i3)
```

# Data exchange between Stata and Maxima

MBS provides two commands for data exchange:
`maximaget` and `maximaput`

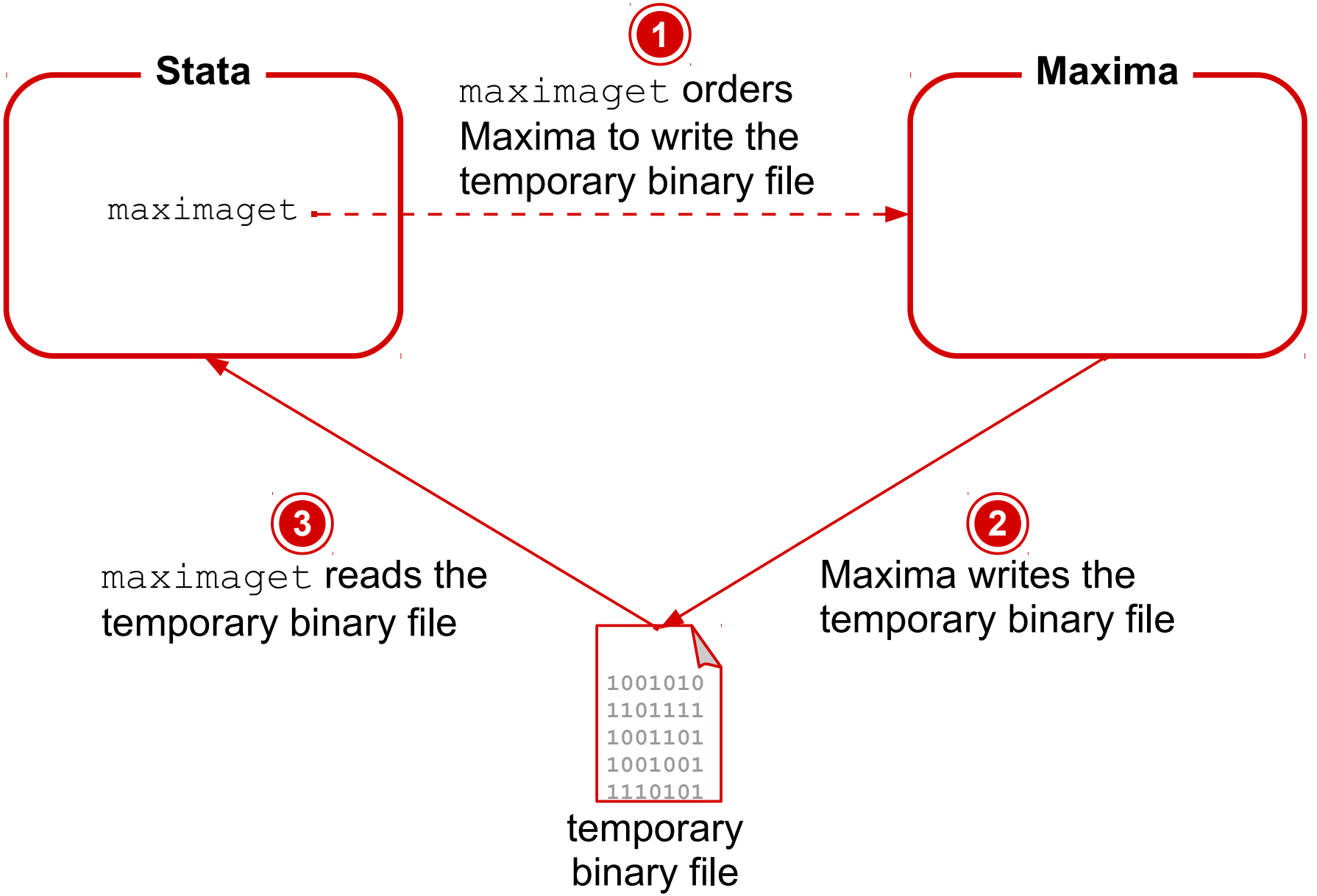


# Importing Maxima data: the `maximaget` command

```
. maxima M: matrix([1, 2], [3, 4]);

                                [ 1  2 ]
(%o1)                           [      ]
                                [ 3  4 ]
(%i2)

. maximaget M

(%o2)                             done
(%i3)

. matrix list M

M[2,2]
    c1  c2
r1   1   2
r2   3   4
```

# How Maxima data are imported into Stata

The Maxima data to be imported is stored in a temporary binary file

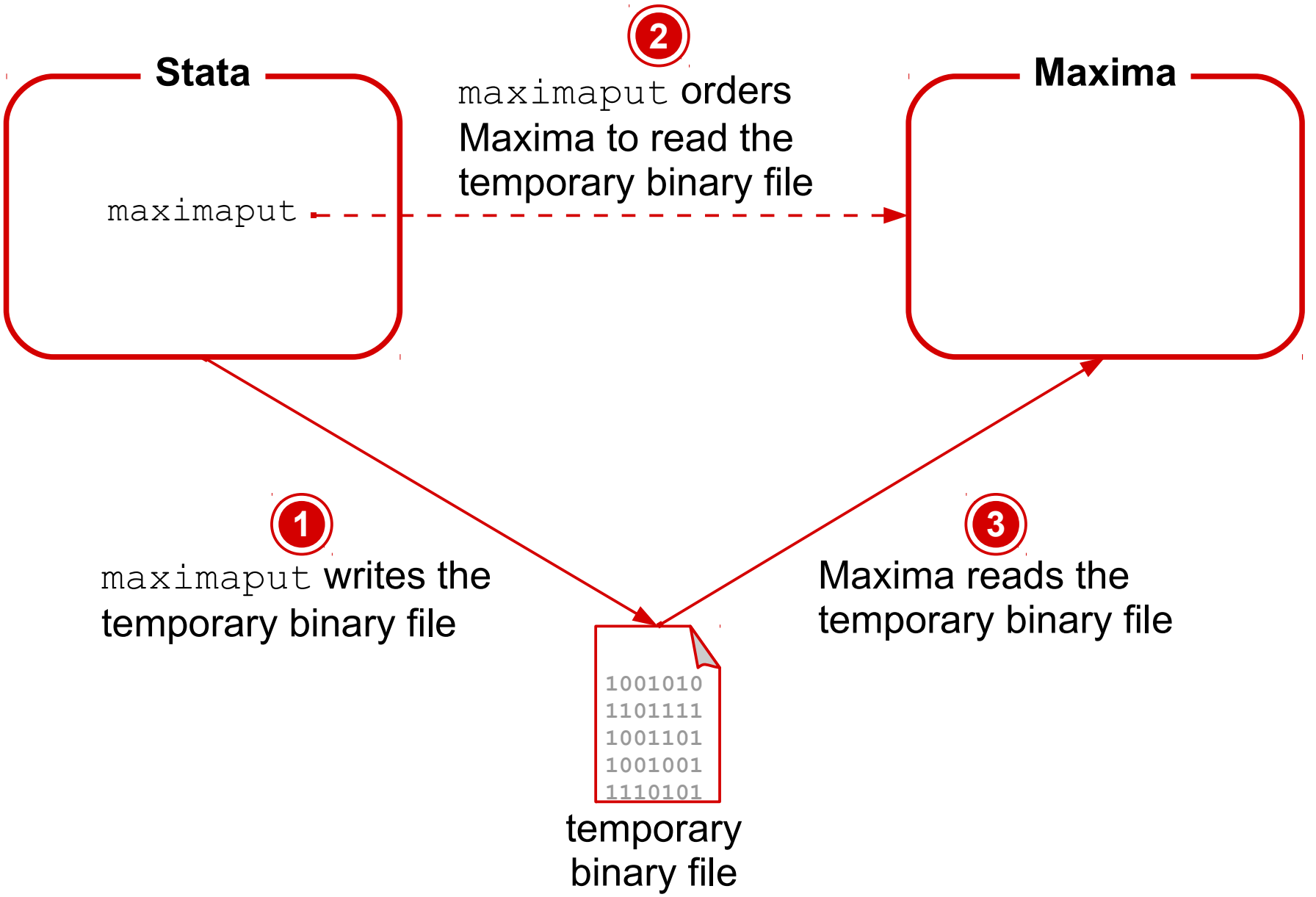


# Exporting Stata data: the `maximaput` command

```
. sysuse auto
(1978 Automobile Data)

. correlate price weight length
(obs=74)

             |    price   weight   length
-------------+---------------------------
       price |   1.0000
      weight |   0.5386   1.0000
      length |   0.4318   0.9460   1.0000


. matrix C = r(C)

. maximaput C

(%o1)
    [       1.0          0.53861146260048  0.43183124497159 ]
    [                                                       ]
    [ 0.53861146260048         1.0         0.94600864341781 ]
    [                                                       ]
    [ 0.43183124497159  0.94600864341781         1.0        ]
(%i2)
```

# How Stata data are exported to Maxima

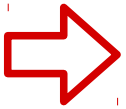
The Stata data to be exported is stored in a temporary binary file



# Syntax of `maximaget` and `maximaput`

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

## Import facilities



**From Maxima**

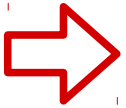
- scalar
- list
- matrix

**To Stata**

- scalar
- matrix
- variable

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

## Export facilities



**From Stata**

- scalar
- matrix
- variable

**To Maxima**

- scalar
- list
- matrix

# MBS Utility Package

Maxima functions to facilitate the cooperation beween Stata and Maxima

- `statamissing`
- `stataf`
- `statafm`
- `killtemp`
- `wxmconnect`

- written in Lisp
- defined in the *maximabridge-init.lisp* file, which is located under the Maxima Bridge installation folder
- automatically loaded by Maxima Bridge

# MBS Utility Package: the `statamissing` function

Returns a Stata missing value

**Syntax**

`statamissing(`*`missing_type`*`)`

`missing_type` is a string, which can be `.`, `a`, `b`, …, `z`:

**Example**

```
. maxima missvalue: statamissing("g");

(%o1)                9.0038268217042019E+307
(%i2)

. quietly maximaget missvalue

. scalar list missvalue
 missvalue =          .g
```

Missing values are internally stored as large positive value

# What if a number is divided by zero?

**Stata**: a system missing value (.) is returned

```
. display 3/0
.
```

**Maxima**: an error is returned

```
. maxima 3/0;

expt: undefined: 0 to a negative exponent.
 -- an error. To debug this try: debugmode(true);
(%i2)
```

Stata functions and expressions generally return a missing value when an error occurs

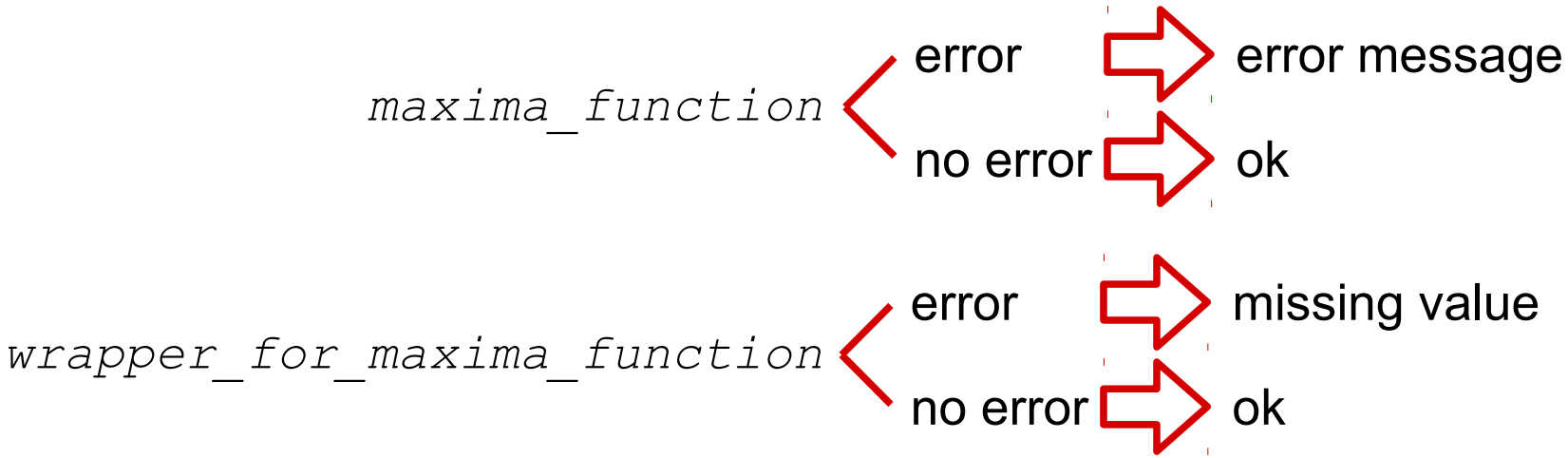
# MBS Utility Package: the `stataf` function

Given a Maxima function `maxima_function` with an argument list `argument_list`, `stataf` returns a wrapper function for `maxima_function` which returns a missing value when an error internally occurs

**Syntax**

`stataf(`*`maxima_function`*`, `*`argument_list`*`)`

- *`maxima_function`* must be a function
- *`argument_list`* must be a list



# MBS Utility Package: the `stataf` function

## Example

```
. maxima divide(x, y) := x/y $

(%i2)

. maxima st_divide(x, y) := stataf(divide, [x, y]) $

(%i3)

. maxima result: st_divide(3, 0);

(%o3)                8.9884656743115795E+307
(%i4)

. maximaget result

(%o4)                          done
(%i5)

. scalar list result
    result =                .
```

# MBS Utility Package: the other functions

**Short descriptions:**

- **`statafm`**
  Similar to stataf, but `statafm` returns a missing value if one of the arguments, which are passed to the wrapper function, is a missing value.
- **`killtemp`**
  Deletes all Maxima variables whose name starts with an underscore (_). These variables are called *temporary variables* in the MBS *jargon* and their function is similar to temporary data in Stata ado-programs.
- **`wxmconnect`**
  Hacks wxMaxima connecting the Maxima process to wxMaxima. Once the connection has been established, the user can exploit the wxMaxima GUI to interact with Maxima. For more information, see Appendix

# The confluent hypergeometric limit function

This funcion is a particular hypergeometric function:

$$ {}_0F_1(a;x) = \sum_{i=0}^{\infty} \frac{x^i}{(a)_i i!} $$

where $(a)_i$ is the Pochhammer symbol for the rising factorial:

$$ (a)_i = a(a+1)\dots(a+i-1) = \frac{\Gamma(a+i)}{\Gamma(a)} $$

**Practical utility in statistics**:
Useful for obtaining the unbiased estimator for the expected value of the lognormal variable (Bradu and Mundlak, 1970; Shen and Zhu, 2008)

**Stata**:

- no built-in function
- hard to implement

**Maxima**:

```
hypergeometric([], [a], x)
```

# ado-Maxima programs

**Stata commands**  **Maxima commands**

# An initial example of ado-Maxima program

An ado-Maxima program for calculating values
for the confluent hypergeometric limit function

syntax parsing

MBS code

```
program confluenthl
 1. syntax varname(numeric), a(real) generate(name)
 2. tempname par_a
 3. scalar `par_a' = `a'
 4. quietly {
 5. maximaput `parameter_a', fromscalar toscalar
 6. maximaput `varlist', fromvar tolist
 7. maxima confluenthl(x) := hypergeometric([],
    [`par_a'], x);
 8. maxima result: map(confluenthl, `varlist');
 9. maximaget result, fromlist tovar
    name(`generate')
10. }
11. end
```

*This program is only for illustrative purposes. It has several flaws and bugs which are not easy to identify for unskilled MBS users*

# Using the initial ado-Maxima program example

```
. set obs 4
obs was 0, now 4

. set seed 1234

. gen x = uniform()

. confluenthl x, a(3) generate(y)

. list

     +----------------------+
     |         x          y |
     |----------------------|
  1. | .4770199   1.1687954 |
  2. | .3306433   1.1148715 |
  3. | .3445907   1.1199265 |
  4. | .3577346   1.1247062 |
     +----------------------+
```

# Improving the ado-Maxima program

- ☑ create an *egen* function (`_gconfluenthl`)
- ☑ allow for `[if]` and `[in]` options

```
program _gconfluenthl
 1. gettoken vartype 0 : 0
 2. gettoken newvar 0 : 0
 3. gettoken equalsign 0: 0
 4. syntax varname(numeric) [if] [in], a(real)
 5. marksample touse
 6. tempname par_a
 7. scalar `par_a' = `a'
 8. quietly {
 9. maximaput `par_a', fromscalar toscalar
10. maximaput `varlist' if `touse', fromvar tolist
11. maxima confluenthl(x) := hypergeometric([], [`par_a'],
    x);
12. maxima result: map(confluenthl, `varlist');
13. maximaget result if `touse', fromlist tovar
    name(`newvar')
14. }
15. end
```

# Using the improved ado-Maxima program example

```
. egen y2 = confluenthl(x) in 2/4, a(3)
(1 missing value generated)

. list

     +-----------------------------------+
     |         x            y         y2 |
     |-----------------------------------|
  1. | .4770199   1.1687954          . |
  2. | .3306433   1.1148715   1.1148715 |
  3. | .3445907   1.1199265   1.1199265 |
  4. | .3577346   1.1247062   1.1247062 |
     +-----------------------------------+
```

# Moments for a random normal variable (r.n.v).

**Random normal variable**

$$X \sim N(\mu; \sigma)$$

**Moment generating function**

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

**Non-central moment of order *n***

$$E(X^n) = \left. \frac{d^n M(t)}{dt^n} \right|_{t=0}$$

**Example**

$$\begin{aligned} E(X^{20}) = {} & 654729075\,\sigma^{20} + 6547290750\,\mu^2\,\sigma^{18} + 9820936125\,\mu^4\,\sigma^{16} \\ & + 5237832600\,\mu^6\,\sigma^{14} + 1309458150\,\mu^8\,\sigma^{12} + 174594420\,\mu^{10}\,\sigma^{10} \\ & + 13226850\,\mu^{12}\,\sigma^8 + 581400\,\mu^{14}\,\sigma^6 + 14535\,\mu^{16}\,\sigma^4 + 190\,\mu^{18}\,\sigma^2 \\ & + \mu^{20} \end{aligned}$$

**Stata**

- no built-in function
- hard to implement

**Maxima**

Requires some code writing

# Example: obtaining the moment of order 20 for a r.n.v.

**Moment generating function for a random normal variable (r.n.v)**

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

```
G(t, %mu, %sigma) := exp(%mu*t)*exp((1/2)*(%sigma^2)*(t^2));
```

**Derivative of order 20 of the moment generating function**

$$\frac{d^{20} M(t)}{dt^{20}}$$

```
gdiff: diff(G(t, %mu, %sigma), t, 20);
```

**Non-central moment of order 20**

$$E(X^{20}) = \left.\frac{d^{20} M(t)}{dt^{20}}\right|_{t=0}$$

```
m20(%mu, %sigma) := ''(at(gdiff, t=0));
```

**Example of use**

```
(%i4) m20(0.5, 0.7);
(%o4)                  5858584.318678359
(%i5)
```

# Ado-Maxima program for the moments of a r.n.v.

```
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. quietly {
 8. maximaput mu if in, fromvar tolist name(mulist)
 9. maximaput sd if in, fromvar tolist name(sdlist)
10. maxima G(t, mu, sd) :=
    exp(mu*t)*exp((1/2)*(sd^2)*(t^2));
11. maxima gdiff: diff(G(t, mu, sd), t, r);
12. maxima m(mu, sd) := (at(gdiff, t=0));
13. maxima res: map(m, mulist, sdlist);
14. maximaget res if in, fromlist tovar name(newvar)
15. }
16. end
```

*This program is only for illustrative purposes. It has several flaws and bugs which are not easy to identify for unskilled MBS users*

# Example of use of _gnormalmoment

```
. clear all

. set obs 4
obs was 0, now 4

. set seed 9999

. generate mu = uniform()

. generate sd = uniform()

. egen m20 = normalmoment(mu sd), r(20)

. list

     +-----------------------------------+
     |        mu          sd          m20 |
     |-----------------------------------|
  1. | .5706419   .8693877   3.508e+08 |
  2. | .7122409   .4555794   32177.066 |
  3. | .9121964   .4310627   81869.779 |
  4. | .3174115   .7898812    17795804 |
     +-----------------------------------+
```

# Critical issues regarding ado-Maxima programming

**It is important to understand how to deal with**:

- undesired evaluation of Maxima symbols
- conflict with pre-existing Maxima symbols
- Maxima symbols as by-products
- overflows
- missing values
- errors returned by Maxima functions

A complete discussion of this matter will be provided in a forthcoming paper

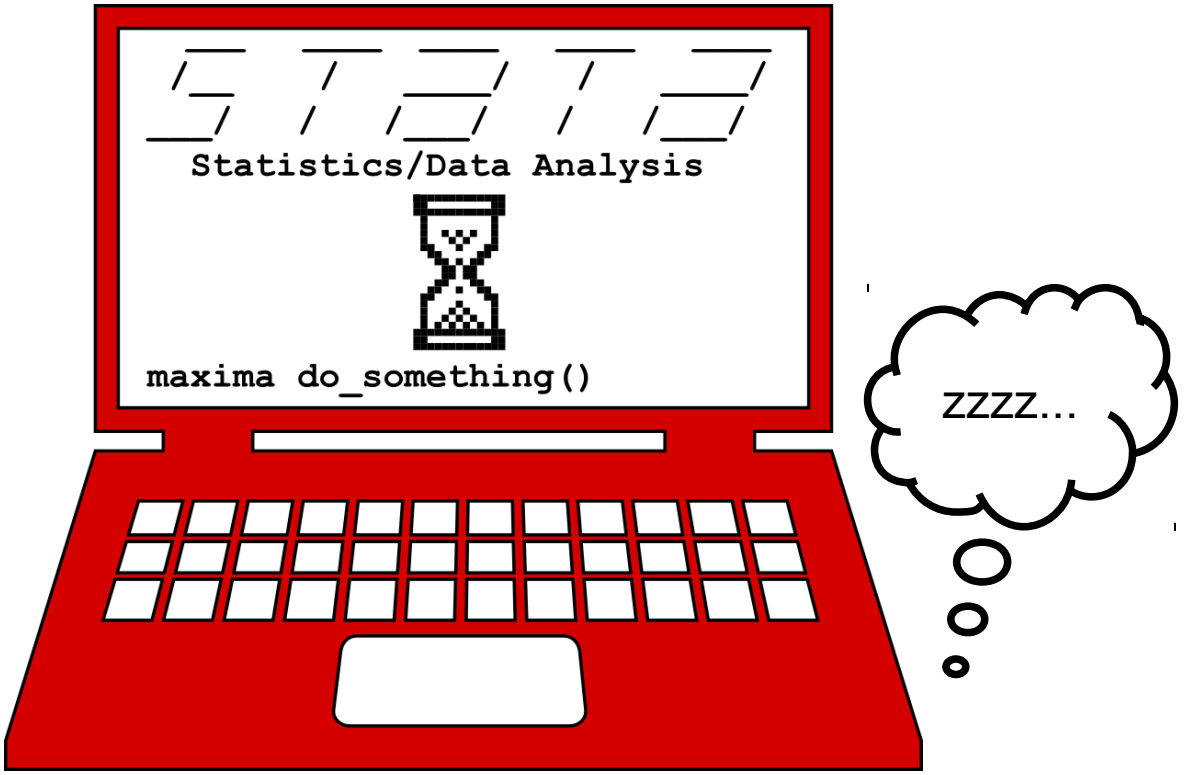
# Concluding remarks: what else MBS can do

- **Big float precision**
  Computations when precision matters
  (see Appendix B)
- **Systems of non-linear equations**
  Assessing identification of SEM models by means of the method by Bollen and Bauldry (2010a, 2010b)
  (see Appendix C)
- **Linear programming**
  More flexibility with respect to the `dea` command by Ji and Lee (2010) for data envelopment analysis (DEA)
- **Hundreds of other Maxima commands**
  Many other things you can do

*Thank you for your attention*

# Appendix A – What to do when the Stata GUI freezes

## When the Stata GUI freezes?




Stata connects to Maxima Bridge to issue Maxima instructions; it stays connected until all output is obtained. The Stata GUI will freeze during this period. Stata disconnects itself when it receives the **terminating character** (ASCII 4), which MBS uses to mark the end of the output.

# Appendix A – What to do when the Stata GUI freezes

## What can cause the Stata GUI to freezes indefinitely?

**Badly-written instructions**, which Maxima interprets as:

- incomplete ⇨ Maxima waits indefinitely for completion
- wrong ⇨ Maxima returns an anomalous error message

⇩

In both cases Maxima does not send the terminating character

# Appendix A – What to do when the Stata GUI freezes

**An example command which causes the Stata GUI to freeze indefinitely**

```
. maxima print("hello
```

(the correct one should be `maxima print("hello");`)

**A possible solution to unfreeze the Stata GUI**

After the Stata GUI has frozen, type the following from Maxima Bridge:

```
");
```

# Appendix A – What to do when the Stata GUI freezes

## Some solutions for unfreezing the Stata GUI


1. Restart Maxima
2. Interrupt Maxima
3. Disconnect the Stata client

## Recommendations:

- attempt to interrupt Maxima
- avoid solution 3 if you are not expert; it may cause Maxima to behave erratically
- see the online help regarding Maxima Bridge for more information

# Appendix B – Big floats and precision

## Number representation in the computer

### IEEE754 double precision floating point number (64 bit)

$$\sqrt{\pi} =$$

0011111111111100010110111111100010010001101101001110111101101011

```
. display %28.26f sqrt(_pi)
   1.77245385090551588e+00
```

### Big floats

Representations of numbers with arbitrary precision

```
(%i1) sqrt(%pi), bfloat, fpprec: 300;
(%o1) 1.772453850905516027298167483341145182797549456122387\
128213807789852911284591032181374950656738544665416226823 62\
428257066623615286572442260252509370960278706846203769865 31\
051228499251730289508262289320953792679628001746390153514 79\
720516700190185234018585446974494912640313921775525906216 40\
541933250091b0
(%i2)
```

# Appendix B – Big floats and precision

## Various remarks regarding IEEE754 double precision representation

### Range

```
. display %28.26f mindouble()
  -8.98846567431157854le+307

. display %28.26f maxdouble()
   8.98846567431157854le+307
```

### Not all decimal numbers can be exactly represented

For example, 0.1 has no exact representation in the IEEE754 standard...

```
. display %28.26f 0.1
0.10000000000000005551115l2
```

… and its “ideal” binary representation is a repeating number:

```
0011111110111001100110011001100l1001...
```

# Appendix B – Big floats and precision

## Using big floats

### Defining a big float

```
(%i1) mybigfloat: 1317.4b5;
(%o1)                          1.3174b8
```

### Converting to a big float

```
(%i2) bfloat(%pi);
(%o2)                 3.14159265358979b0
```

### Setting digit precision

```
(%i3) fpprec: 40;
(%o3)                              40
(%i4) bfloat(%pi);
(%o4)   3.1415926535897932384626433832795028841970b0
```

# Appendix B – Big floats and precision

## Performing calculations with big floats

In most cases, expressions with big floats returns a big float

```
(%i5) 1 + 2.0b0;
(%o5)                          3.0b0
```

Occasionally, the conversion to a big float have to be explicitily invoked

```
(%i6) %pi + 2.0b0;
(%o6)                      %pi + 2.0b0
(%i7) bfloat(%pi + 2.0b0);
(%o7)   5.14159265358979323846264338327950288419 7b0
```

# Appendix B – Big floats and precision

## Numerical accuracy of in Stata

**A tricky expression for Stata**

$$\frac{e^x - 1}{x}$$ ⇨ *(exp(x)-1)/x*

***(exp(x)-1)/x* when *x* is very small**

$$\lim_{x \to 0} \frac{e^x - 1}{x} = 1$$

```
. quietly set obs 3

. set seed 9999

. generate double x = runiform()

. quietly replace x = 2^(-150) in 2

. generate double stata = (exp(x)-1)/x

. list

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

# Appendix B – Big floats and precision

## Double precision and big float precision for *(exp(x)-1)/x* in Maxima

### Double precision

```
(%i1) x: 2^(-150)$

(%i2) float((exp(x)-1)/x);
(%o2)                          0.0
```

### Big float

```
(%i3) fpprec: 16;
(%o3)                           16
(%i4) (exp(bfloat(x))-1)/x;
(%o4)                        0.0b0
(%i5) fpprec: 50;
(%o5)                           50
(%i6) (exp(bfloat(x))-1)/x;
(%o6)                        1.0b0
```

# Appendix B – Big floats and precision

## Improving precision in Stata through Maxima: 50 digits

```
. quietly maxima fpprec: 50;

. quietly maxima f50(x):= (exp(bfloat(x))-1)/x;

. quietly maxima maxima50: map(f50, x);

. quietly maximaget maxima50, fromlist tovar

. list

     +-----------------------------------+
     |          x       stata    maxima50 |
     |-----------------------------------|
  1. |  .57064196   1.3483105   1.3483105 |
  2. |  7.006e-46           0           1 |
  3. |  .91219641   1.6331846   1.6331846 |
     +-----------------------------------+
```

# Appendix B – Big floats and precision

## What if *x* approaches 0?

```
. quietly replace x = 2^(-250) in 2

. quietly replace stata = (exp(x)-1)/x

. quietly maximaput x, fromvar

. quietly maxima fpprec: 50;

. quietly maxima f50(x):= (exp(bfloat(x))-1)/x;

. quietly maxima maxima50: map(f50, x);

. quietly maximaget maxima50, fromlist tovar replace

. list

     +-----------------------------------+
     |          x       stata    maxima50 |
     |-----------------------------------|
  1. | .57064196   1.3483105   1.3483105 |
  2. | 5.527e-76           0           0 |
  3. | .91219641   1.6331846   1.6331846 |
     +-----------------------------------+
```

# Appendix B – Big floats and precision

## More precision: 100 digits

```
. quietly maximaput x, fromvar

. quietly maxima fpprec: 100;

. quietly maxima f100(x):= (exp(bfloat(x))-1)/x;

. quietly maxima maxima100: map(f100, x);

. quietly maximaget maxima100, fromlist tovar

. list

     +-----------------------------------------------------+
     |          x        stata     maxima50    maxima100 |
     |-----------------------------------------------------|
  1. |  .57064196    1.3483105    1.3483105    1.3483105 |
  2. |  5.527e-76            0            0            1 |
  3. |  .91219641    1.6331846    1.6331846    1.6331846 |
     +-----------------------------------------------------+
```

# Appendix C – Assessing identification of SEM models

## Bollen and Bauldry's method (2010a, 2010b)

Bollen and Bauldry's method is a broadly applicable method which requires the solving of systems of non-linear equations

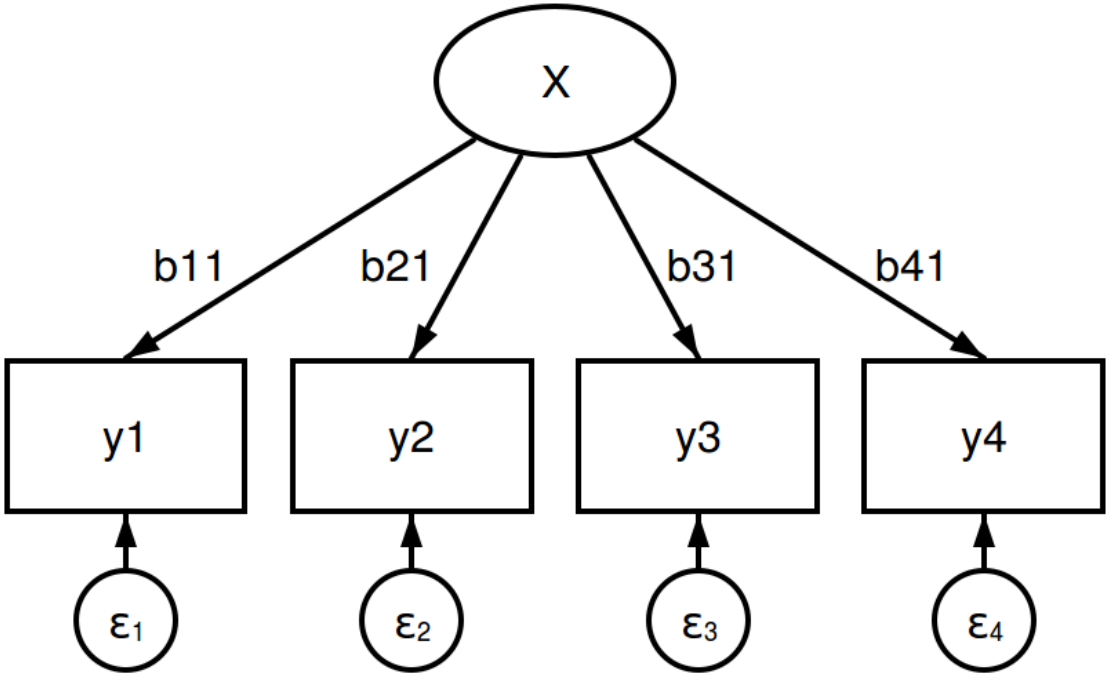
Very difficult for humans: a CAS approach is suggested

Stata's SEM builder + MBS

can facilitate the work

# Appendix C – Assessing identification of SEM models

## A very simple example: can this model be identified?




We know this model is not identified; we will pretend not to know this and apply Bollen and Bauldry's method

# Appendix C – Assessing identification of SEM models

## The example model in matrix form

$$\mathbf{Y} = \mathbf{a} + \mathbf{G}X + \zeta$$

Observed endogenous variables

$$\mathbf{Y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$$

Intercepts

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}$$

Path coefficients

$$\mathbf{G} = \begin{pmatrix} g_{11} \\ g_{21} \\ g_{31} \\ g_{41} \end{pmatrix}$$

Erratic components

$$\zeta = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \end{pmatrix}$$

Variance of the latent exogenous variable

$$\mathbf{\Phi} = \mathrm{Var}(X) = \begin{pmatrix} f_{11} \end{pmatrix}$$

Variance matrix for the erratic components

$$\mathbf{\Psi} = \mathrm{Var}(\zeta) = \begin{pmatrix} p_{11} & 0 & 0 & 0 \\ 0 & p_{22} & 0 & 0 \\ 0 & 0 & p_{33} & 0 \\ 0 & 0 & 0 & p_{44} \end{pmatrix}$$

All random variables are normally distributed with 0 as expected value

# Appendix C – Assessing identification of SEM models

## Variance matrix of observed endogenous variables

$$\boldsymbol{\Sigma} = \mathbf{G}\boldsymbol{\Phi}\mathbf{G}' + \boldsymbol{\Psi}$$

$$= \begin{pmatrix} s_{11} & s_{21} & s_{31} & s_{41} \\ s_{21} & s_{22} & s_{32} & s_{42} \\ s_{31} & s_{32} & s_{33} & s_{43} \\ s_{41} & s_{42} & s_{43} & s_{44} \end{pmatrix}$$

$$= \begin{pmatrix} p_{11} + f_{11}g_{11}^2 & f_{11}g_{11}g_{21} & f_{11}g_{11}g_{31} & f_{11}g_{11}g_{41} \\ f_{11}g_{11}g_{21} & p_{21} + f_{11}g_{21}^2 & f_{11}g_{21}g_{31} & f_{11}g_{21}g_{41} \\ f_{11}g_{11}g_{31} & f_{11}g_{21}g_{31} & p_{33} + f_{11}g_{31}^2 & f_{11}g_{31}g_{41} \\ f_{11}g_{11}g_{41} & f_{11}g_{21}g_{41} & f_{11}g_{31}g_{41} & p_{44} + f_{11}g_{41}^2 \end{pmatrix}$$

# Appendix C – Assessing identification of SEM models

## The system of equations

$$\begin{cases} p_{11} + f_{11} g_{11}^2 = s_{11} \\ f_{11} g_{11} g_{21} = s_{21} \\ p_{21} + f_{11} g_{21}^2 = s_{22} \\ f_{11} g_{11} g_{31} = s_{31} \\ f_{11} g_{21} g_{31} = s_{32} \\ p_{33} + f_{11} g_{31}^2 = s_{33} \\ f_{11} g_{11} g_{41} = s_{41} \\ f_{11} g_{21} g_{41} = s_{42} \\ f_{11} g_{31} g_{41} = s_{43} \\ p_{44} + f_{11} g_{41}^2 = s_{44} \end{cases}$$

9 unknowns: $g_{11},\ g_{21},\ g_{31},\ g_{41},\ f_{11},\ p_{11},\ p_{22},\ p_{33},\ p_{44}$

10 knowns: $s_{11},\ s_{2,1},\ s_{2,2},\ s_{3,1},\ s_{3,2},\ s_{3,3},\ s_{4,1},\ s_{4,2},\ s_{4,3},\ s_{4,4}$

#knowns > #unknowns

# Appendix C – Assessing identification of SEM models

## Proof by Bollen and Bauldry (2010a)

Given a subset of equations with the following characteristics:

- it contains all the unknowns
- the number of equations equals the number of unknowns

if a unique algebraic solution is found for this subset of equations, this is sufficient to establish identification

# Appendix C – Assessing identification of SEM models

## Two examples of subset of equations to consider

$$\begin{cases} p_{11} + f_{11}g_{11}^2 = s_{11} \\ f_{11}g_{11}g_{21} = s_{21} \\ p_{21} + f_{11}g_{21}^2 = s_{22} \\ f_{11}g_{11}g_{31} = s_{31} \\ f_{11}g_{21}g_{31} = s_{32} \\ p_{33} + f_{11}g_{31}^2 = s_{33} \\ f_{11}g_{11}g_{41} = s_{41} \\ f_{11}g_{21}g_{41} = s_{42} \\ p_{44} + f_{11}g_{41}^2 = s_{44} \end{cases}$$

$$\begin{cases} p_{11} + f_{11}g_{11}^2 = s_{11} \\ f_{11}g_{11}g_{21} = s_{21} \\ p_{21} + f_{11}g_{21}^2 = s_{22} \\ f_{11}g_{11}g_{31} = s_{31} \\ f_{11}g_{21}g_{31} = s_{32} \\ p_{33} + f_{11}g_{31}^2 = s_{33} \\ f_{11}g_{11}g_{41} = s_{41} \\ f_{11}g_{31}g_{41} = s_{43} \\ p_{44} + f_{11}g_{41}^2 = s_{44} \end{cases}$$

The total number of subsets of equations to consider is 6

# Appendix C – Assessing identification of SEM models

## How to hack Stata's SEM Builder to “sniff” model description from it


The “SEM estimation options” dialog is opened after the user has clicked on the “estimate button” ( ) from SEM Builder

An object called `__SEMG` (instance of the `__sg_class`) is created to transfer model informations to the “SEM estimation options” dialog

An ado-program can be create to “intercept” the data exchanged through the `__SEMG` object

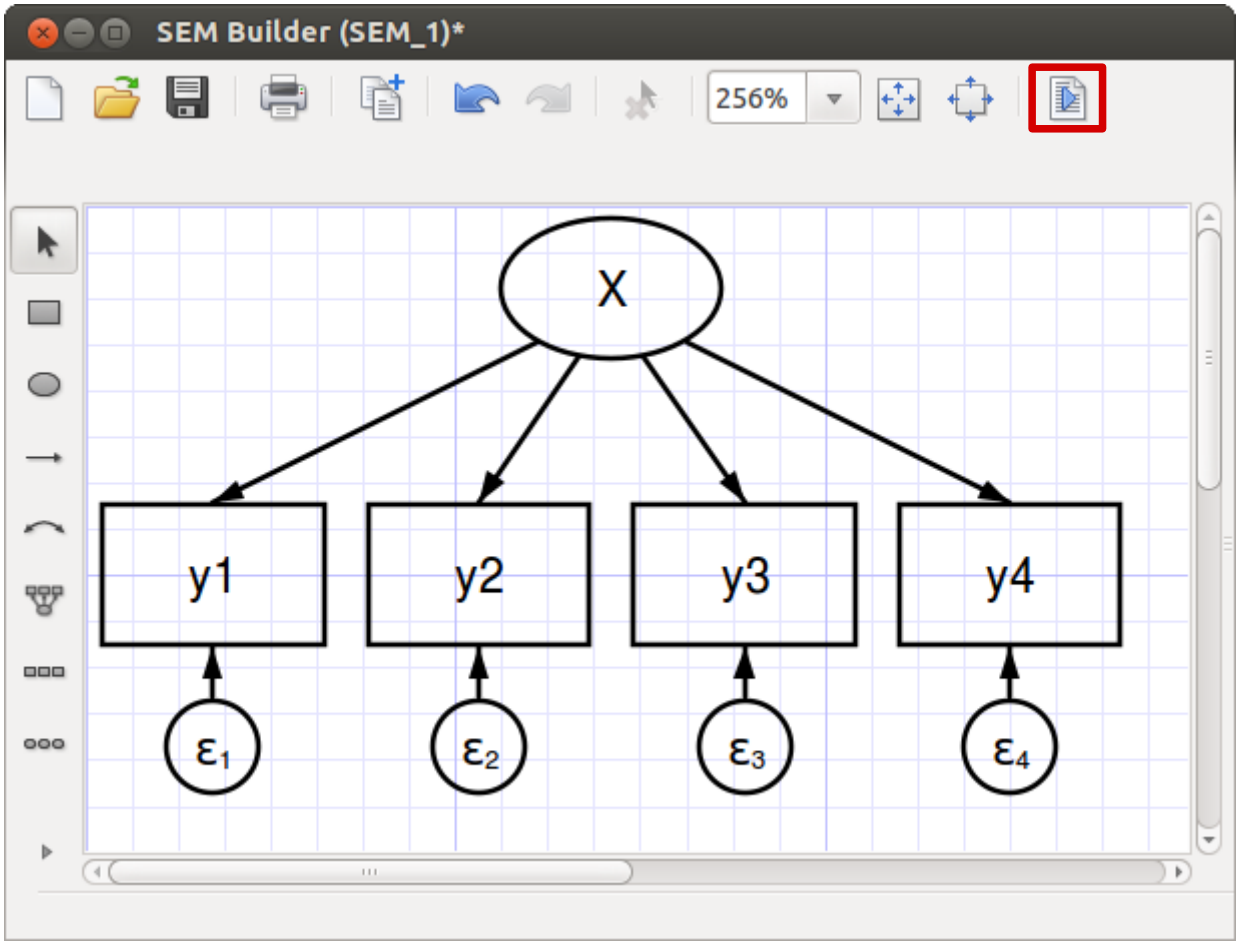
# Appendix C – Assessing identification of SEM models

## Step 1: create a dummy dataset

```
. clear all

. generate y1 = .

. generate y2 = .

. generate y3 = .

. generate y4 = .

. * this dataset has no observations

. count
    0
```

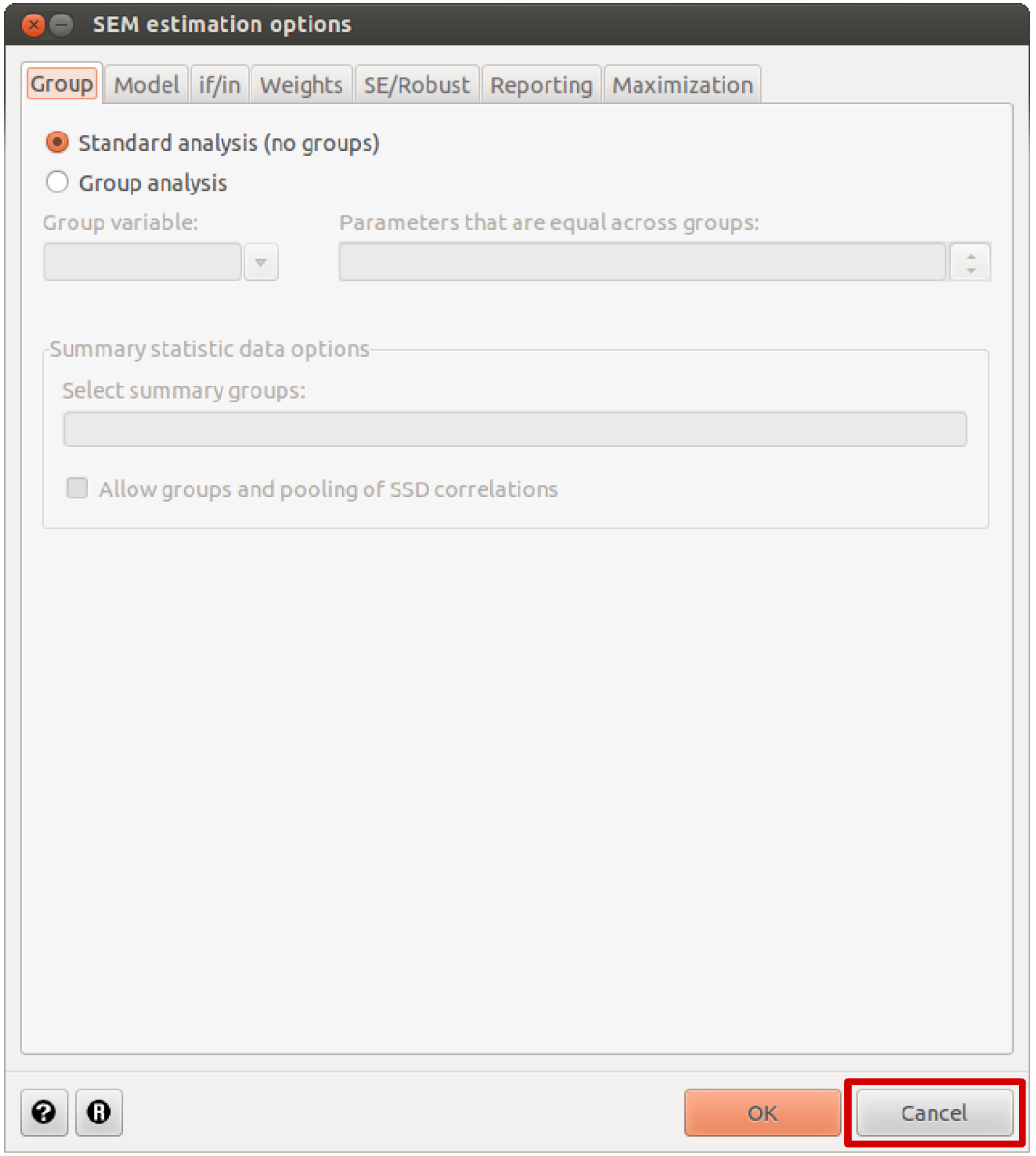
# Appendix C – Assessing identification of SEM models

## Step 2: draw the SEM model and click on the “estimate” button



# Appendix C – Assessing identification of SEM models

## Step 3: click on Cancel from the “SEM estimation options” dialog



# Appendix C – Assessing identification of SEM models

## Step 4: “sniff” information about the SEM model

```
. classutil describe __SEMG, recurse

(output omitted)

. classutil describe __SEMG.dbsettings.cmd
string        .__SEMG.dbsettings.cmd = "sem (X -> y1) (X ->
y2) (X -> y3) (X -> y4)"

. classutil describe __SEMG.dbsettings.options
string        .__SEMG.dbsettings.options = " latent(X )
nocapslatent"

. classutil describe __SEMG.dbsettings.covstruct
string        .__SEMG.dbsettings.covstruct = ""
```

# Appendix C – Assessing identification of SEM models

## The `semidentify` command

- uses MBS to assess identification of SEM models
- `semidentify` "sniffs" `__SEMG`: must be called after step 1, 2, 3 e 4
- is an experimental command with a lot of defects and bugs
- is not currently on public release, but it can be obtained from the author of this presentation (at user's risk)

# Appendix C – Assessing identification of SEM models

## The `semidentify` command: example of use

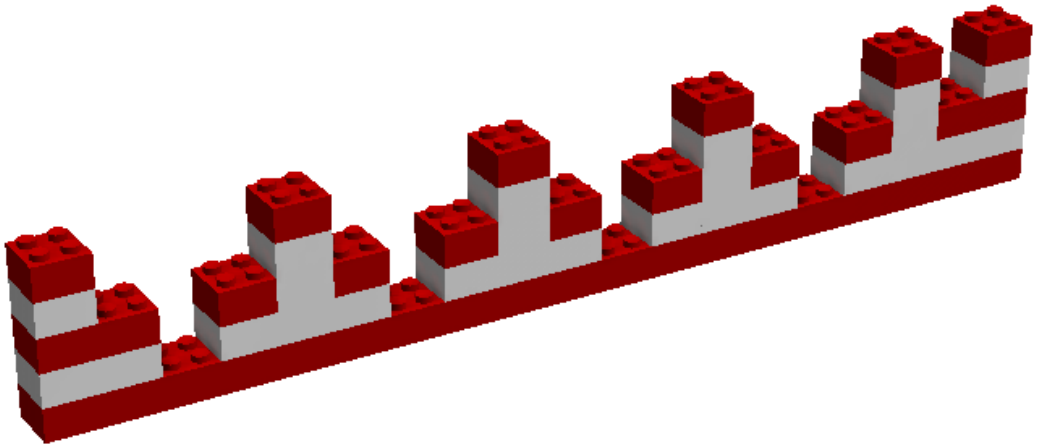
```
. semidentify sniffer
Sniffed paths: sem (X -> y1) (X -> y2) (X -> y3) (X -> y4)
Sniffed options:  latent(X ) nocapslatent
Sniffed covariance restrictions:

(very long output omitted)

6 subsets of equations found:

Subset 1: Solve | Show
Subset 2: Solve | Show
Subset 3: Solve | Show
Subset 4: Solve | Show
Subset 5: Solve | Show
Subset 6: Solve | Show
```

You can click on the blue links to solve or show a particular system of equations

`semidentify` found 6 subsets of equations to consider.
No solution was found for the 6 systems of equations:
**the model is not identified**

# Appendix D – Maxima Bridge plugins

## What is a Maxima Bridge plugin?

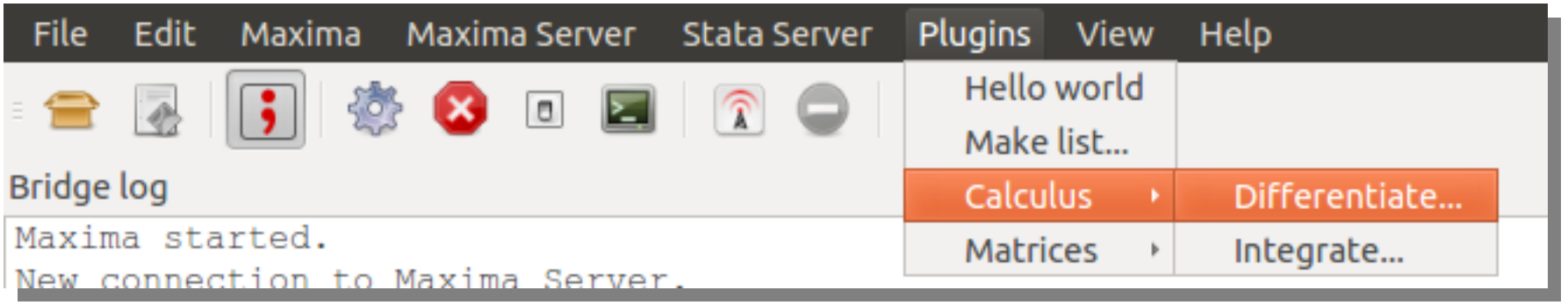


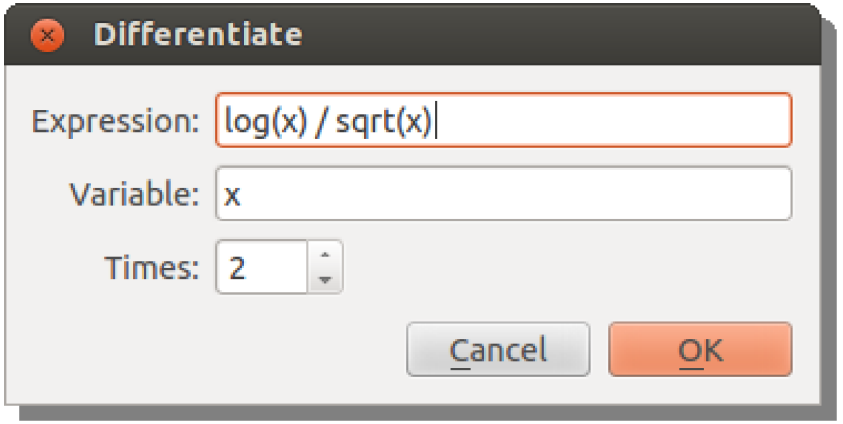
A software add-on which adds new features to Maxima Bridge

- generally, a dialog window which enable the issuing of a Maxima command
- a single file: *.dll* in Windows, *.dyllib* in Mac, *.so* in Linux
- the user can develop new plugins

# Appendix D – Maxima Bridge plugins

## An example plugin: Differentiate




```
diff(log(x) / sqrt(x), x, 2);
```

# Appendix D – Maxima Bridge plugins

## How to develop, load and distribute new plugins

### Developing

- plugins can be written in C++ by using the Qt4 programming framework (https://qt-project.org)
- details are provided in the online help for Maxima Bridge

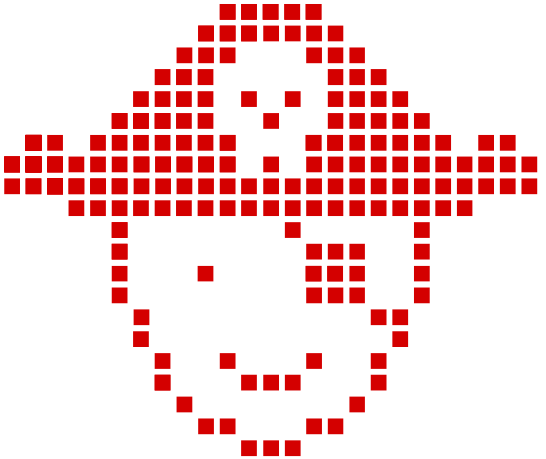
### Loading

- place the new plugin in the “plugins” folder and restart Maxima Bridge
- the structure of the “plugins” menu mimics the tree for the “plugins” folder

### Distribution

Remember: plugins are platform-dependent

# Appendix E – Hacking wxMaxima to exploit its GUI



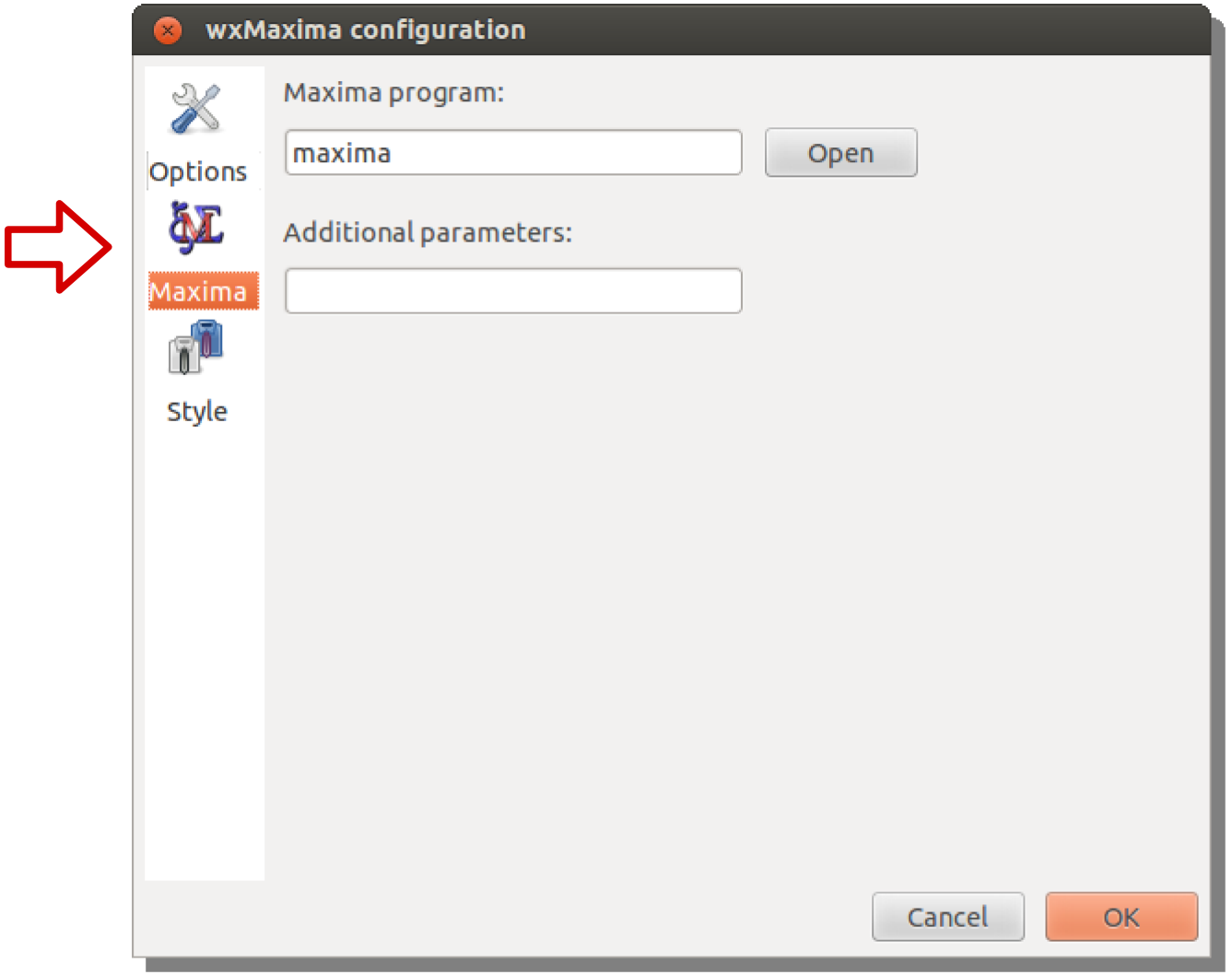
`wxmconnect` permits to temporarily use wxMaxima as a GUI

- wxMaxima can be hacked to accept the connection of the Maxima process
- thereafter, is possible to reconnect the Maxima process to Maxima Bridge

# Appendix E – Hacking wxMaxima to exploit its GUI

## Step 1

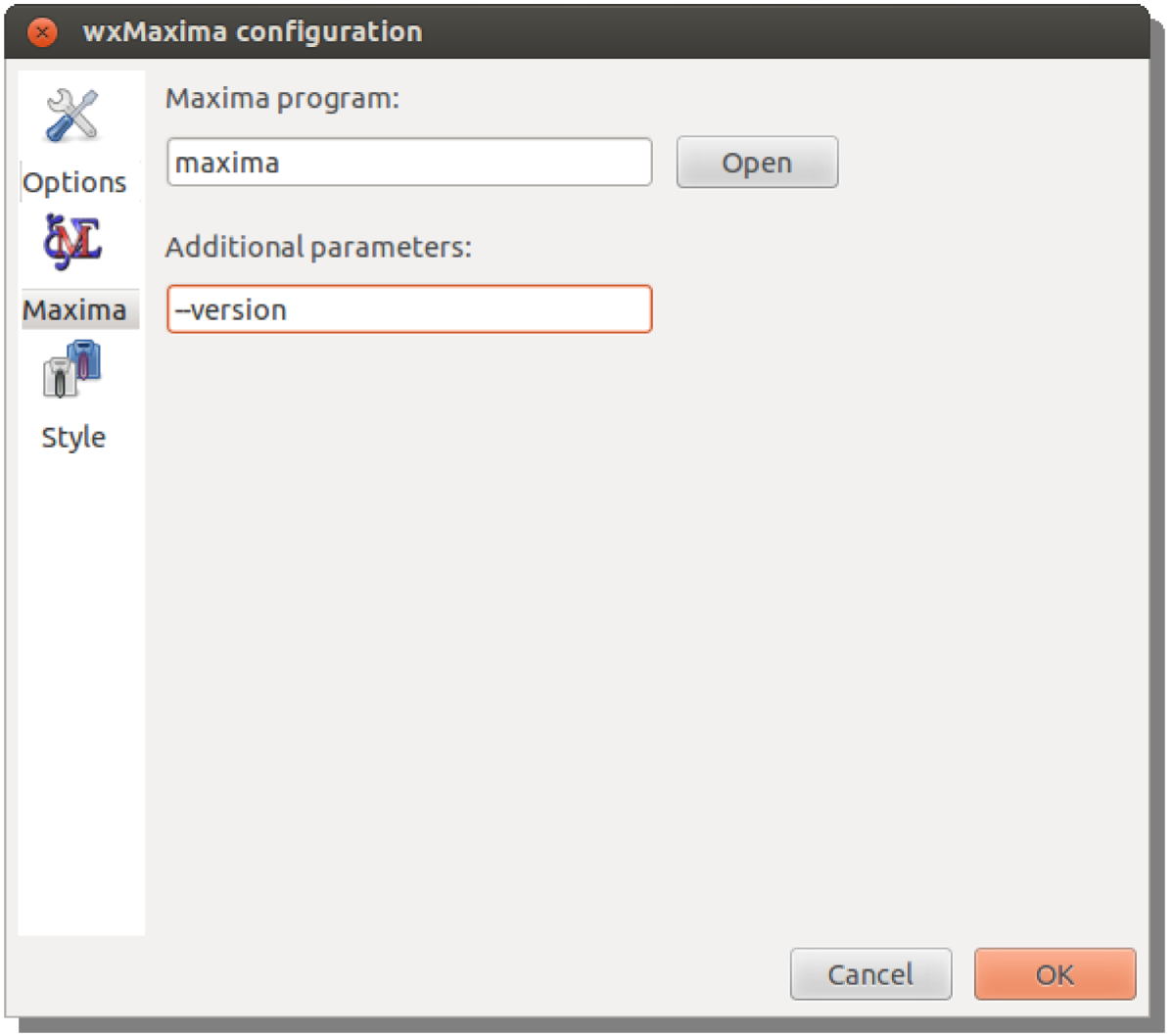
From wxMaxima: edit → configure



# Appendix E – Hacking wxMaxima to exploit its GUI

## Step 2

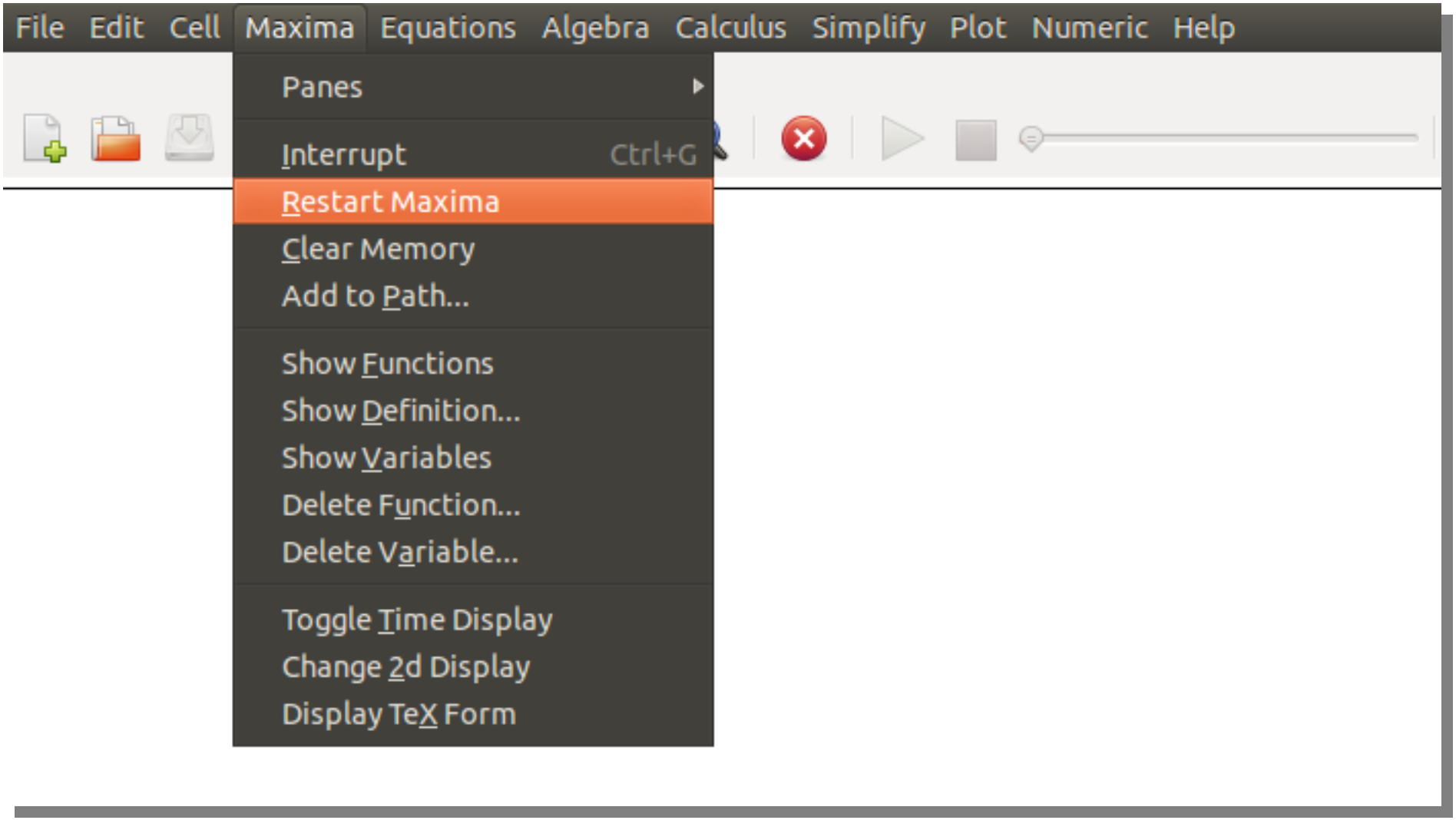
Type “--version” in the “Additional parameters” field



# Appendix E – Hacking wxMaxima to exploit its GUI

## Step 3

Restart Maxima

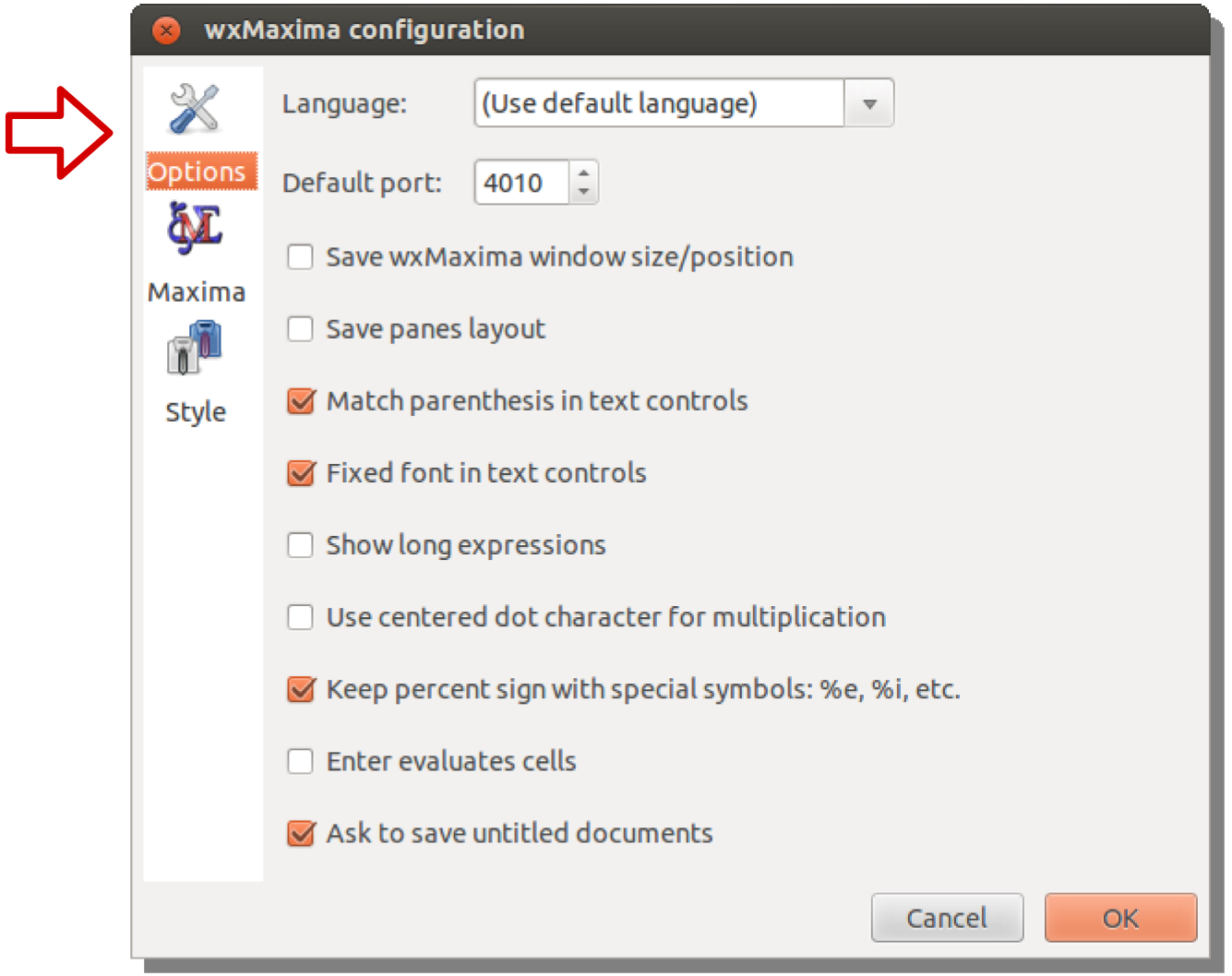



Now wxMaxima is waiting for an incoming connection: the port is open

# Appendix E – Hacking wxMaxima to exploit its GUI

## Step 4

Identify the port number which is used by wxMaxima

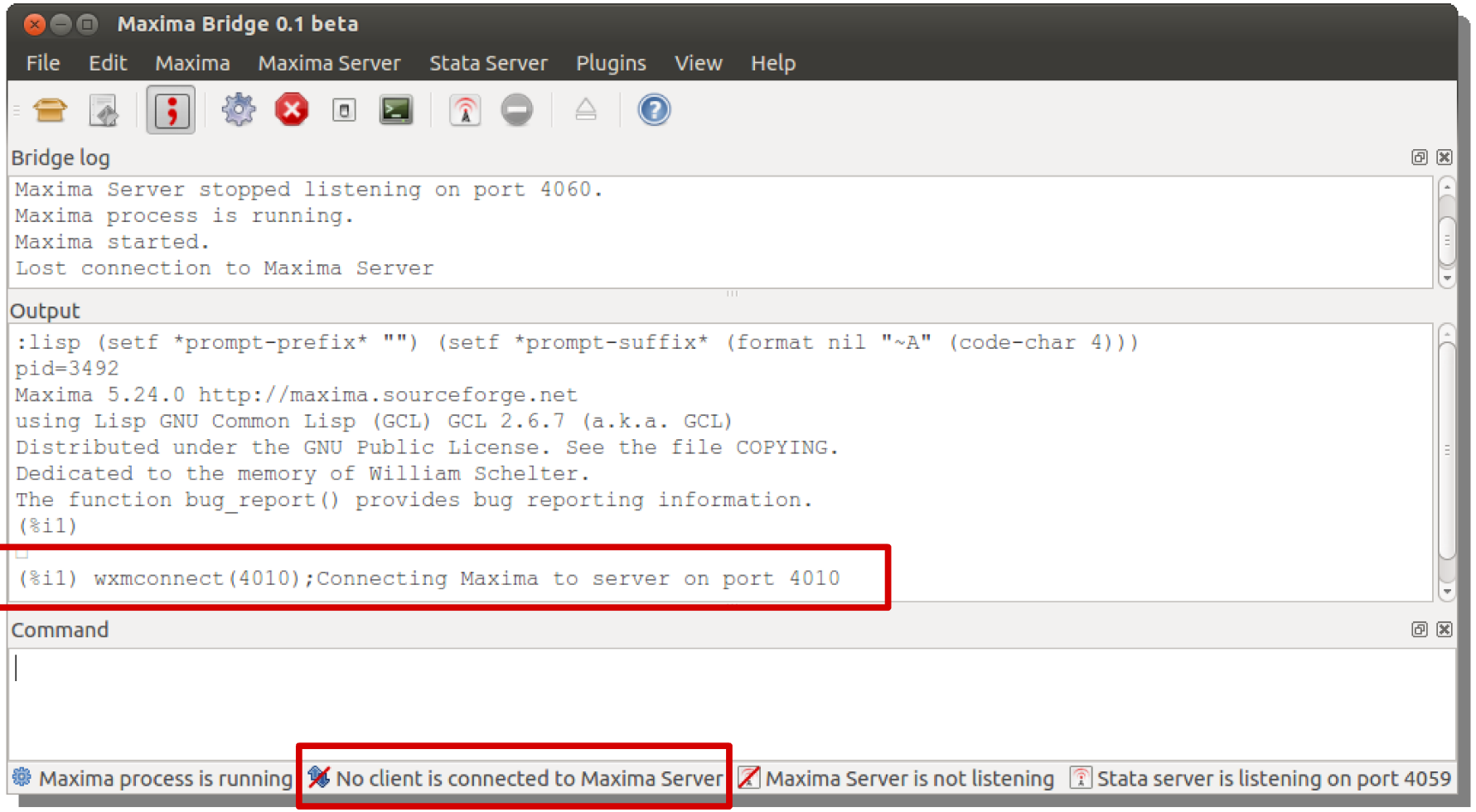



Warning: the default port is 4010 but wxMaxima may automatically use a port greater than 4010 when that port is used by another application

# Appendix E – Hacking wxMaxima to exploit its GUI

## Step 5

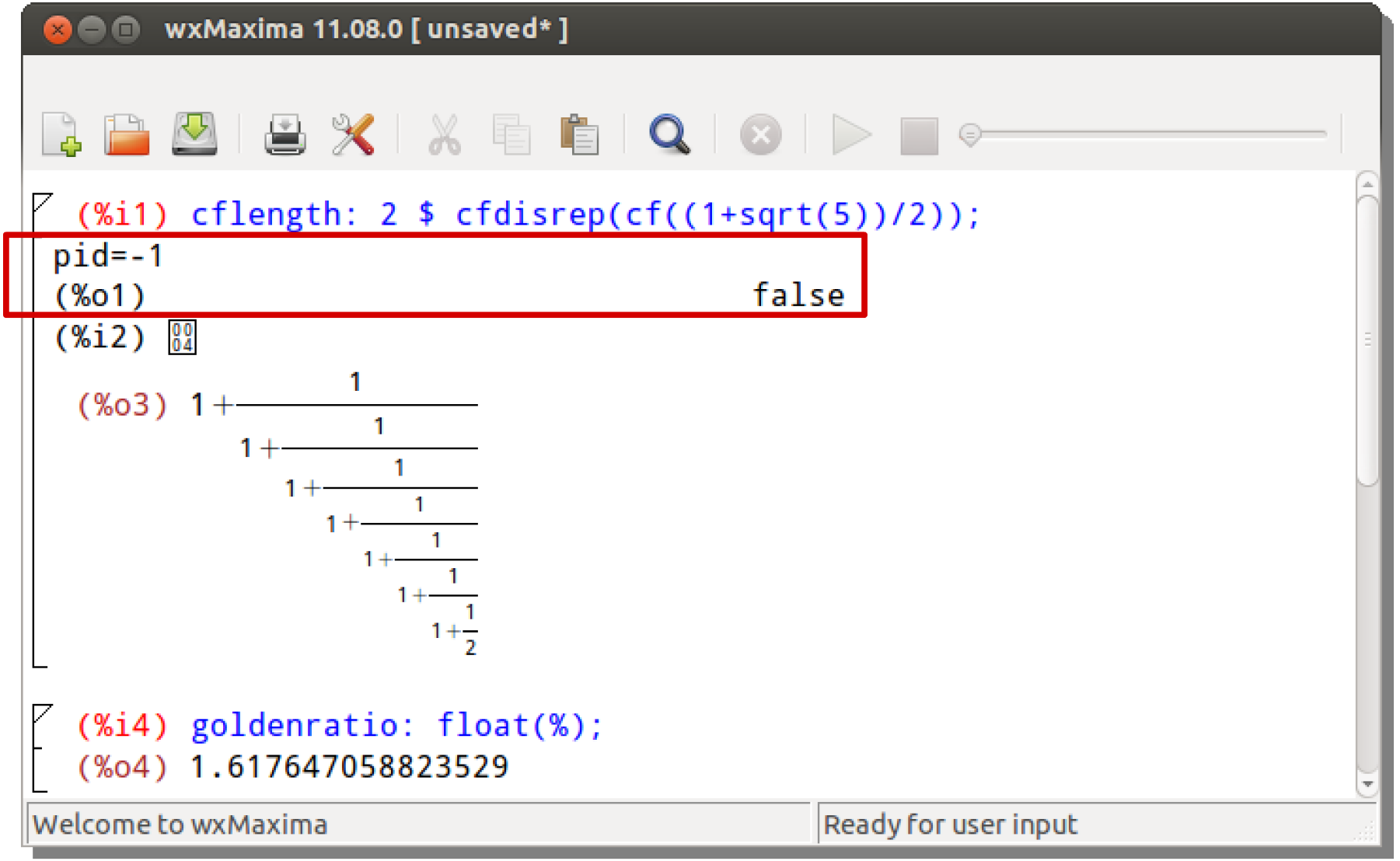
Type "`wxmconnect(4010);`" from Maxima Bridge
(change the port number 4010 if it is not that used by wxMaxima)



# Appendix E – Hacking wxMaxima to exploit its GUI

## Step 6

wxMaxima has been successfully hacked: now start your working session

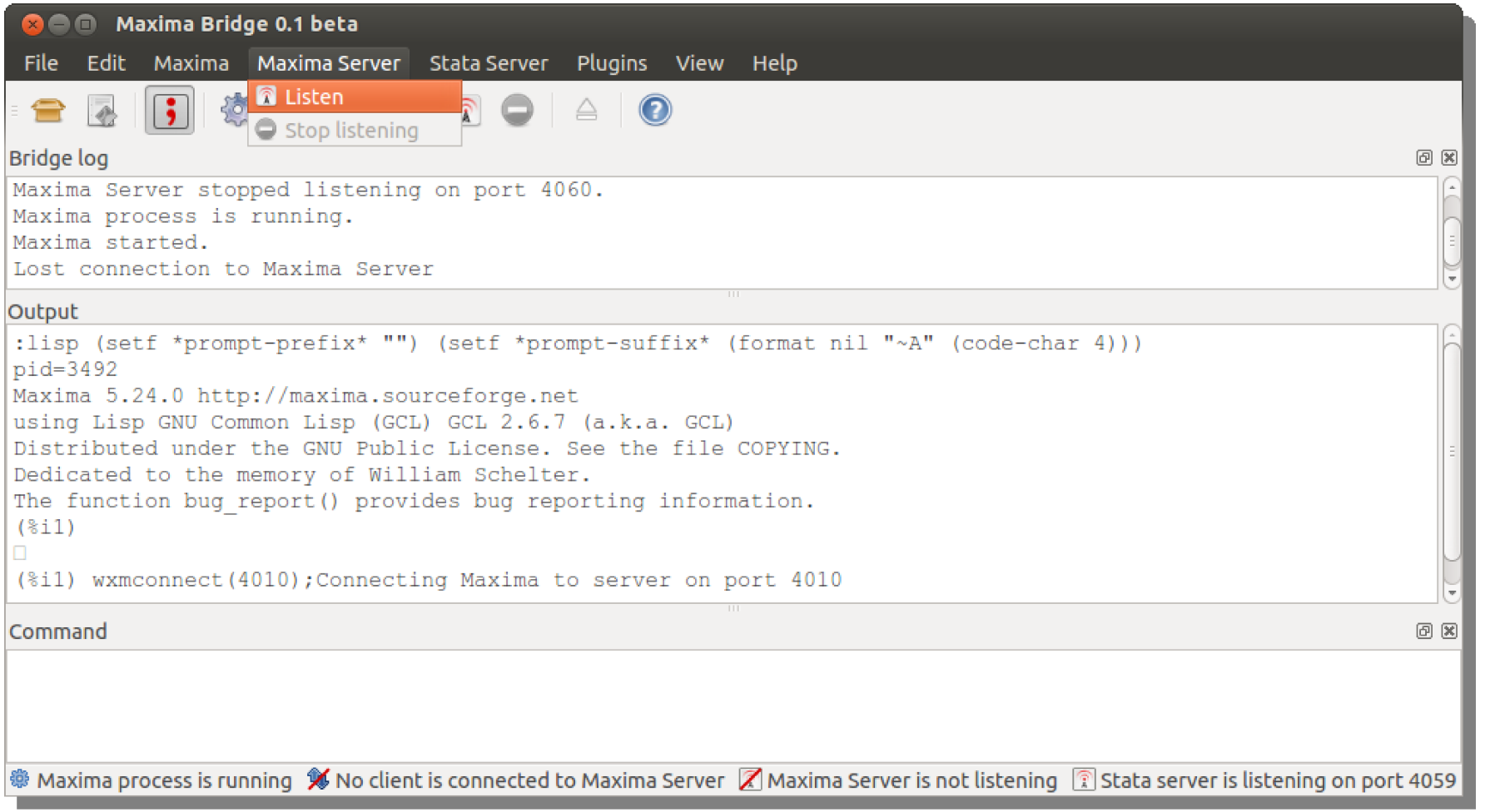



* Do not be alarmed about this *strange* message!

# Appendix E – Hacking wxMaxima to exploit its GUI

## Step 7

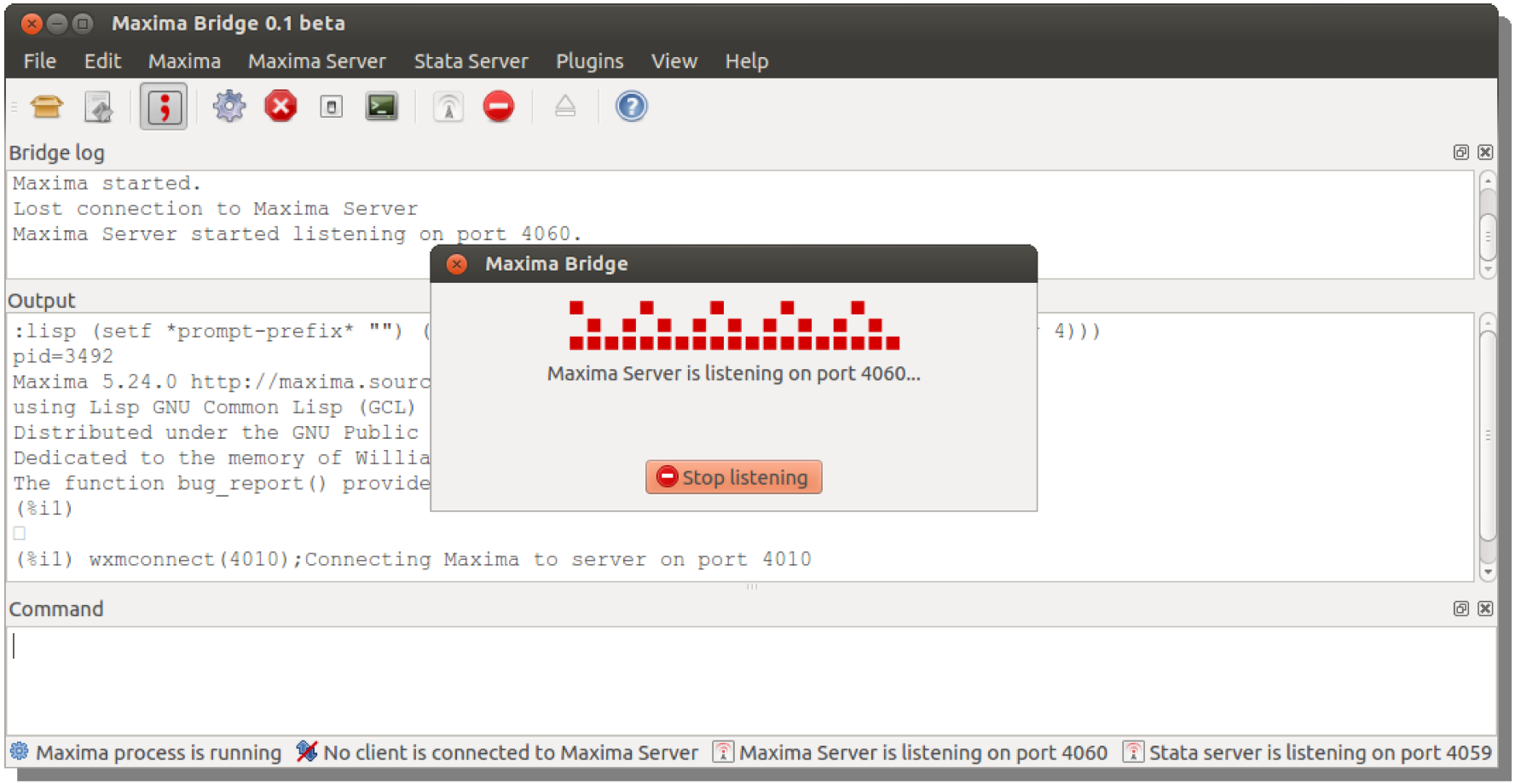
From Maxima Bridge: Maxima Server → Listen



# Appendix E – Hacking wxMaxima to exploit its GUI

## Step 8

Identify the port on which Maxima Server is listening for

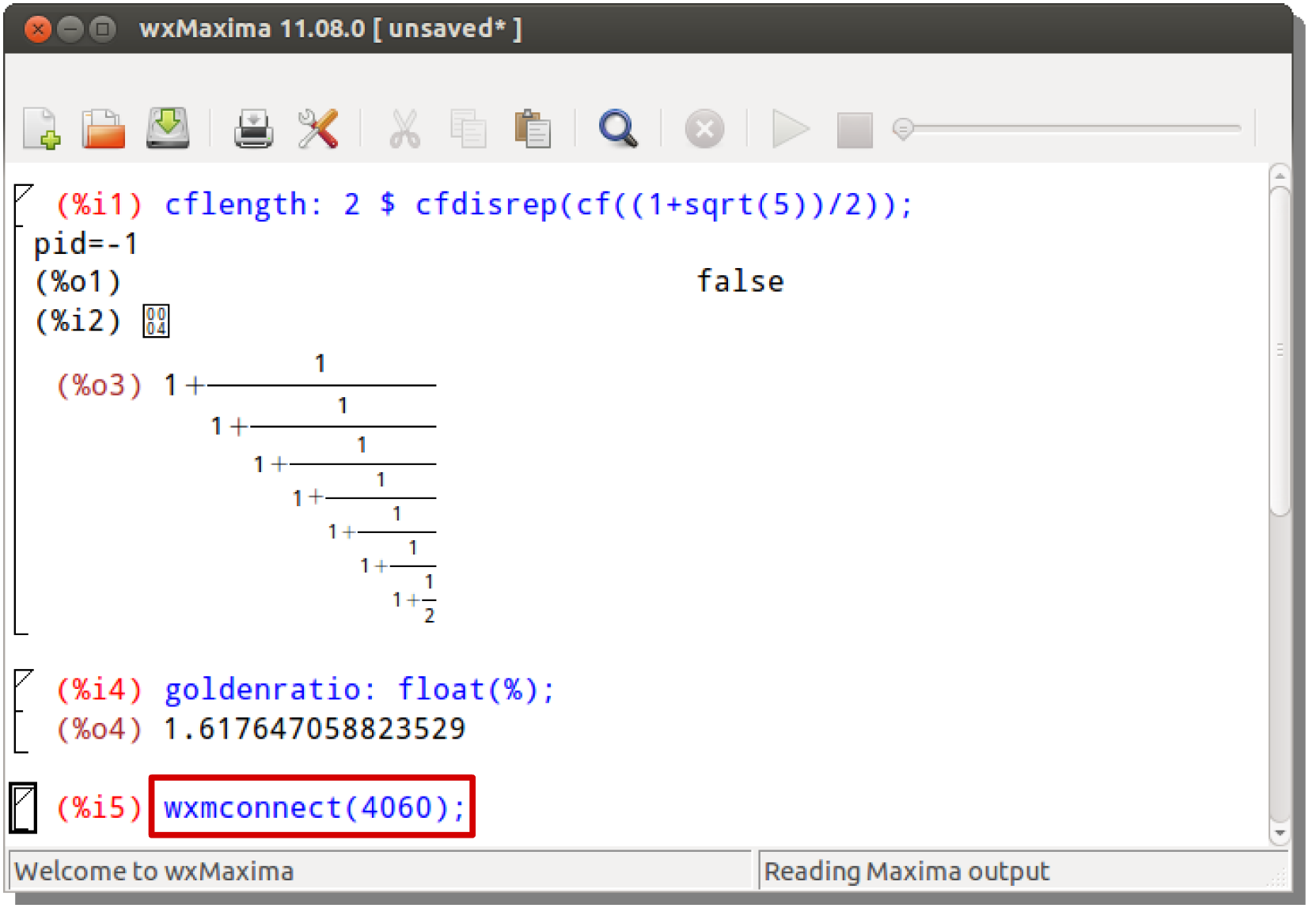



Default port is 4060

# Appendix E – Hacking wxMaxima to exploit its GUI

## Step 9

From wxMaxima: type "`wxmconnect(4060);`"
(change the port number 4060 if it is not that used by Maxima Bridge)



# Appendix E – Hacking wxMaxima to exploit its GUI

## Step 10

From Stata: use what you created in wxMaxima

```
. maximaget goldenratio

(%o7)                          done
(%i8)

. scalar list goldenratio
goldenratio =  1.6176471
```

# References


Bollen K. A. and Bauldry S. 2010a. A Note on Algebraic Solutions to Identification. *Journal of Mathematical Sociology* 34(2), 136-145.

Bollen K. A. and Bauldry S. 2010b. Model Identification and Computer Algebra. *Sociological Methods & Research* 39(2), 127-156.

Bradu D. and Mundlak Y. 1970. Estimation in Lognormal Linear Models. *Journal of the American Statistical Association* 65(329), 198-211.

Hilbert D. (1894). Ein Beitrag zur Theorie des Legendre'schen Polynoms. *Acta Mathematica* 18(1), 155-159.

Ji Y. And Lee C. 2010. Data Envelopment Analysis in Stata. *Stata Journal* 10(2), 267-280.

Shen H. and Zhu Z. 2008. Efficient Mean Estimation in Log-normal Linear Models. *Journal of Statistical Planning and Inference* 138(3), 552-567.

# Sometimes, someone imagines the future

*“Many persons who are not conversant with mathematical studies imagine that because the business of the engine [Babbage’s Analytical Engine] is to give its results in numerical notation, the nature of its processes must consequently be arithmetical and numerical rather than algebraical and analytical. This is an error. The engine can arrange and combine its numerical quantities exactly as if they were letters or any other general symbols; and in fact it might bring out its results in algebraical notation were provisions made accordingly”*

*Augusta Ada Byron (1815-1852)*