AJD-16

THE TRIO REGRESSION ANALYST

THE TRIO SHORT REFERENCE, VERSION 4 Beta x

by

Marc Gaudry, Francine Dufort, Holger Priske, Liem Tran

October 1997, February 2001

TABLE OF CONTENTS

Chapter 1

1. OVERVIEW

In this chapter, we will present certain concepts and basic skills that are common to all classes of variables. But first of all, we will define some of the elements that make up a TRIO document: the variable, the variant, the database. We will also define the regression model types available in TRIO.
On a more pratical level, we will then present the tasks of calling TRIO and creating the first TRIO database. For users of earlier versions of TRIO, some comparisons and notes on backward compatibility will be presented. Finally, a tool common to all variable classes will be described in detail: the algebraic expressions tool which gives the user the possibility of creating variables from other variables and intrinsic functions (available) and from variant results (not yet available).

1.1 BASIC ELEMENTS OF TRIO

1.1.1 The TRIO database

The role of the database is the integration of the formulation, estimation and result analysis of the regression models defined below. It contains all the data necessary for defining and analysing results. The database consists of four elements: the variable, the variant, the tablex and the graph. These elements are listed in the right-hand window of the TRIO desktop when a TRIO document is opened for the first time: they represent the basic elements that are saved between TRIO sessions. Also saved in the database are the user's notes and comments regarding modifications either to the original data or to the specification of a variant. Special features insure the integrity of the database, avoiding for example the destruction of a data element needed in a variant.

1.1.2 The TRIO document

Whereas a TRIO database may be associated with a single project, a TRIO document consists of one or several databases. The different databases are listed in the left-hand window of the TRIO desktop: only the Level models are available in the beta version of TRIO. Later versions will allow a user to save in a single TRIO document several databases belonging to the three types of regression classes: Level, Share and Probability. As a general rule, it is not recommended to create more than a single database within each TRIO document because when opening the document, all databases are loaded at the same time and available memory will be reduced. This feature may however prove useful when saving several small databases, for example when regrouping the demonstration data, or when working on related projects. The databases are independent of one another: variables in one cannot be used in another. It is possible however, to Cut/Paste/Copy database elements from one database into another. Drag and Drop operations are not yet supported.

1.1.3 The variable

The variable represents the data used in the regression models The values of a variable are a set of real numbers corresponding to observations. They may be obtained from another Windows application through Cut&Paste, or defined and modified within TRIO. The variables may also be created as algebraic expressions of existing variables (available) or derived from variant results (not yet available). A description and a comment may be given to each variable, thus allowing the user to maintain a history of any update or modification applied to the variable. Each variable is unique within the database; any inadvertent attempt to create

a variable whose name already exists in the database will cause TRIO to add an underscore followed by a version number to the variable name.

Deleting or modifying a variable may destroy consistency of variant results. Worse, the variant definition itself may become invalid. For this reason, the deletion/modification of a variable may imply the deletion/modification of all the database elements containing the affected variable: any related variable and all variants, tablexes and graphs. This procedure insures the **integrity** of the database and guaranties the **consistency** of all results that remain **reproducible** at any time.

The TRIO variable belongs to one of the three following classes:

- ***Level variables*** (ordered) most models explain the level of a given variable, say total consumption, or the suicide rate, or transit demand, by sets of explanatory variables. The data on both dependent and independent variables may be ordered (naturally) by time. The class of level variables is designed with classical regression models in mind and may be used for cross sectional or time series data. The running index of observations can be used by estimation procedures in order to account for autocorrelation of the residuals.

- ***Share variables*** (not yet available). Many models explain the observed continuous proportions of the market belonging to each member of a set of alternatives, say travel modes, or classes of firms, or types of sicknesses. The class of share variables is designed with aggregate contemporaneous cross sections in mind. It is not required that all alternatives be available for each observation.

- ***Probability variables*** (not yet available). A significant number of models explain the mutually exclusive discrete occurrence of various choices, events or alternatives, such as chosen modes of transport, states of health and occupational choices. The class of probability variables is designed with disaggregate contemporaneous cross sections in mind. It is not required that all alternatives be available for each individual.

1.1.4 The variant

A variant corresponds to a set of the following: the **regression class** (level, share or probability), the **model type** (OLS, LIN-LOGIT, etc…), the **detailed specification** given by the definition of the dependent variable, the independent variables, the parameters and of all power transformations applied to the variables, and finally, of the **variant results**. In the case of a level variant, the results computed by TRIO are the regression coefficients, six elasticities, as many derivatives, and two t-statistics. These results are saved within the TRIO database and may be viewed on screen any time after the variant estimation. The variant name is unique within the database. During the creation of a variant, if the variant name already exists in the database, then TRIO will add an underscore and a version number to the name. This feature, originally designed to insure the integrity of the database, may however be used to associate a variant name with a sequence of experiments, each version being a specific trial. As with the variables, each variant has two fields where a description and a comment may be saved.

1.2 COMMENTS FOR USERS OF EARLIER VERSIONS OF TRIO

In previous versions, a database was made up of several files called trioxx files. In version 4.0, the TRIO document is instead constituted of one single file containing one or more databases. The database may be viewed as all information related to a specific project or data analysis. The database has been enlarged to save results that were previously lost upon exiting TRIO: the **tablexes** and the **graphs** have now become elements of the TRIO database and are saved between TRIO sessions. Another notable difference between this version and earlier versions is the **initialisation** of a variant which has now become transparent to the user. In previous versions, the initialisation and the estimation of a variant were performed in two distinct steps. These two tasks are now performed in a single operation during the estimation procedure.

Finally, for backward compatibility, the user may import into the TRIO database the old **ASCII files** typically called d*.in files. To import for example a d111.in variables file, expand the Level pull down menu by clicking on the + sign before the Level item. Left click on the Variables item to highlight it, then right click to view the pop up menu. Click on **Import** and a pop up window will appear, where you will enter the path and the filename of your variables file. Click on the Open button to activate

the reading of the variables file. Another pop up menu will appear, requesting that you choose the language to be associated with one or two descriptions if they are present in the variables file. Click on the OK button to return to the desktop. You will now see a list of the variable names in the right window. When importing an ASCII file, if the variable name (or the variant name) already exists in the database, the imported element will be renamed by extending the duplicate name with and underscore and a number. The groups file g111.in may be imported in the exact same manner.

In a similar fashion, the variants file d211.in may be imported: you will simply highlight the Variant item rather than the Variable item before proceeding with the Import operation. All variants thus imported will subsequently have to be estimated because importing a variant is now limited to a simple read operation, contrary to earlier versions of TRIO which reestimated a variant whose solution was not optimal. Both the **Import** and **Export** operations are available for those who wish to create ASCII files that may be read by earlier versions. Both the Export and Import items may also be found in the roll down menu under File. A word of caution: if the automatic rename was applied to the imported variables, then the variants imported refer to the existing variables rather than the imported variants! For this reason, it is recommended that the variables and variants files be imported into empty databases.

1.3 THE MODEL TYPES

The three classes of regression model types eventually available in TRIO are best explained by considering the following very general regression problem:

$$y_m = f_m(X_m; \pi_m, \mu_m), \qquad m = 1, \ldots, M$$

where, for the m^{th} equation of interest, $f_m(.)$ is an unspecified function relating the continuous dependent variable y_m to exogenous variables X_m; π_m denotes the parameters to be estimated and μ_m denotes the error terms.

With this general problem in mind, one can understand the formal definitions found in Table 1.1:

- ***classes*** are defined by *limitations on possible values of the* y_m (on their individual range and on whether they must sum to one) due to the behavior of the set of regression functions considered and the (continuous or discrete) *nature of observations* available on the y_m;

- ***families and model types*** are defined by *limitations on the range of observations* and by *constraints on the specific functional form of the* $f_m(.)$. A specific functional form includes both a systematic part (involving the regressors) and a stochastic part (or model of the residuals).

In the following outline, we will give an idea of the planned model types together with their options. The last column of Table 1.1 gives the availability of each type in the present version of TRIO. The term later in this column denotes items available in earlier versions but not yet available in version 4.0.

Table 1.1 Definition of Regression Class and Model Type

Class	Values of the dependent variable y_m				Form of the $f_m(.)$		Avail.
	in the regression		in the sample		in the systematic and stochastic parts		in TRIO
	$\sum_m y_m = 1$	range	nature	range	#	name	v. 4.0
LEVEL	NO	$-\infty < y_m < \infty$	cont.	$-\infty < y_m < \infty$	OLS	OLS (Root)	yes
	NO	$0 < y_m < \infty$	cont.	$0 < y_m < \infty$	L-1.4	|- BC-GAUHESEQ	yes
	NO	$-\infty < y_m < \infty$	cont.	$-\infty < y_m < \infty$	L-2.0	|- BC-DAUHESEQ	yes
SHARE	YES	$0 < y_m < 1$	cont.	$0 < y_m < 1$	S-1	LIN-LOGIT (Root)	later
	YES	$0 < y_m < 1$	cont.	$0 < y_m < 1$	S-1	|- BC-LOGIT	later
	YES	$0 < y_m < 1$	cont.	$0 < y_m < 1$	S-2	|- S-DOGIT	later
	YES	$0 < y_m < 1$	cont.	$0 < y_m < 1$	S-3	|- G-DOGIT	later
	YES	$0 < y_m < 1$	cont.	$0 < y_m < 1$	S-4	|- LIN-IPT-LOGIT	later
	YES	$0 < y_m < 1$	cont.	$0 < y_m < 1$	S-5	|- BT-IPT-LOGIT	later
PROB.	YES	$0 < y_m < 1$	discr.	$y_m = 0$ or 1	P-2	LIN-LOGIT (Root)	later
	YES	$0 < y_m < 1$	discr.	$y_m = 0$ or 1	P-2	|- BC-LOGIT	later
	YES	$0 < y_m < 1$	discr.	$y_m = 0$ or 1	P-3	|- S-DOGIT	later
	YES	$0 < y_m < 1$	discr.	$y_m = 0$ or 1	P-4	|- G-DOGIT	later
	YES	$0 < y_m < 1$	discr.	$y_m = 0$ or 1	P-5	|- LIN-IPT-LOGIT	later
	YES	$0 < y_m < 1$	discr.	$y_m = 0$ or 1	P-6	|- BT-IPT-LOGIT	later
	Defines CLASSES		Defines FAMILIES and MODEL TYPES				

Table 1.2 Class Root Models and their Nesting Extensions

Class	TYPE		EXTENSIONS				
	Root	Family or Type (name)	Functional form estimation			Corrections of residuals	
			on y_m		on X_k	Generalized Autocorrelation	Generalized Heteroskedasticity
			Asymmetry	Non-0 tails			
LEVEL	OLS	OLS				yes	yes
		BC-GAUHESEQ	yes		yes	yes	yes
		BC-DAUHESEQ	yes		yes	yes	yes
PROB. and SHARE	LIN-LOGIT	LIN-LOGIT					
		BC-LOGIT	yes		yes		
		S-DOGIT	yes	yes	yes		
		G-DOGIT	yes	yes	yes		
		LIN-IPT-LOGIT	yes	yes	yes		
		BT-IPT-LOGIT	yes	yes	yes		

1.3.1 The Level Class.

Classical regression models belong to the **level class**. Model types of interest include:

- **L-1.4:** this procedure (Liem, Dagenais and Gaudry, 1993) naturally allows the user to specify the simplest root procedure of the LEVEL class, or more complex ones, namely:

  OLS **O**rdinary **L**east **S**quares
  BC-GAUHESEQ **B**ox-**C**ox **G**eneralized **AU**toregressive **HE**teroskedastic **S**ingle **EQ**uation

  More precisely, this procedure for **single equation** model estimation consists of a Box-Cox regression with multiple order autocorrelation and a very general form of the heteroskedasticity which includes as nested special cases most forms in practice. Distinct Box-Cox transformations can be used on the dependent variable, on groups of explanatory variables and in the model heteroskedastic residuals. The procedure assumes that y_m is *a priori* a doubly censored variable but that the actual data used for the model estimation include no limit (mass point) observations; it verifies *a posteriori* the validity of this assumption by calculating the probability that each observation be at the limits.

- **L-2.0:** this procedure (Liem and Gaudry, 1994) replaces the serial correlation specification by a much more general or ***directed*** autocorrelation potentially linking any residual to any other, as occurs for instance in the special case of spatially correlated residuals. Two orders are allowed and each can exhibit a geometrically declining lagged effect in a manner that generalizes Koyck's distributed lag of time series.

1.3.2 The Share Class

All model types which belong to the **share class** (Liem *et al.*,) include the simple LIN-LOGIT, i.e. the classical LOGIT form with linear-in-parameters "representative utility" or exponent functions for which an alternative may not be available for some observations. In consequence, this lowest common denominator, or root, option can be compared to the more complex additional possibilities offered by each TRIO model type of the share class, namely:

- **S-1:** this algorithm allows the user to specify the simplest root procedure of the SHARE class, or more complex ones:

  LIN-LOGIT **LIN**ear exponent **LOGIT** model
  BC-LOGIT **B**ox-**C**ox **LOGIT** model

  The first, LIN-LOGIT, is the classical LOGIT form with linear-in-parameters "representative utility" or exponent functions. The supplementary dimensions of the second procedure arise through the fact that the BOX-COX-LOGIT (Gaudry and Wills, 1978) is defined by adding Box-Cox transformations on all variables of a LOGIT. This makes possible the use of identical sets of explanatory variables in all exponent functions, an option that can be called the EXTENDED, or GENERALIZED, formulation (Gaudry, 1978). The STANDARD option is defined by using the same sets of explanatory variables as those one would use in the LIN-LOGIT.

- **S-2:** the STANDARD DOGIT model (Gaudry and Dagenais, 1979) adds to the LOGIT formulation parameters that make it possible to represent compulsive consumption, or captivity to alternatives. TRIO allows the user to simultaneously use Box-Cox transformations on the explanatory variables, as in S-1.

- **S-3:** the GENERALIZED DOGIT model (Gaudry and Dagenais, 1978), transforms the single-alternative "captivity" parameters of the STANDARD DOGIT model into parameters pertaining to each combination of alternatives. The TRIO user must decide of the functional form of the explanatory variables before using this option.

- **S-4:** the INVERSE POWER TRANSFORMATION (IPT) (Gaudry, 1981) makes it possible to define a first family, the LINEAR-IPT-LOGIT, where additional parameters applied to the exponential functions of the logit model itself represent

captivity to alternatives and can make the shape of the response curve asymmetric. Moreover, they also allow the user to specify an EXTENDED formulation by the use of identical sets of explanatory variables in all exponent functions; a STANDARD option is defined by using the same sets of explanatory as those one would use in the LIN-LOGIT. The TRIO user is allowed to simultaneously use direct Box-Cox transformations on the explanatory variables.

- **S-5:** the second family, the BOX-TUKEY IPT LOGIT, offers the same possibilities as S-4, but through transformations applied to the representative utility functions of the logit model itself. The TRIO user is allowed to simultaneously use Box-Cox transformations on the explanatory variables.

1.3.3 The Probability Class

In both the probability class and the share class, the expected value of the dependent variable y_m can be viewed as the probability of a particular state. These classes differ, however, in the observed realizations of the state y_m. In one case, shares are observed. In the other, mutually exclusive choices are observed, typically in the guise of non ordered categorical data: given binary values, they are often called **disaggregate** data. TheP-2 to P-6 families are the exact analogs of the S-1 to S-5 cases found in the SHARE class.

1.4 CALLING TRIO

To call TRIO, click on the TRIO icon or from within the Windows Explorer, click on the TRIO application normally located in the directory C:\Program Files\Trio\bin. This will bring to screen the first TRIO desktop made up of the menu bar, the icons bar and the TRIO windows. The left window, called the model window, lists one single model type "Level" (the "Share" and "Probability" model types are not yet available). In the right window, small coloured icons appear beside the items: "Variables", "Variants", "Tablexes" and "Graphs". For an existing document, one may click on File-> Open -> and enter the name of a TRIO document, or click on the name of a recently opened TRIO document listed at the bottom of the File pull down menu. Finally, one may click on the name of a TRIO document from within the Windows Explorer to open that particular document.

1.5 CREATING THE DATABASE

By definition, the TRIO database consists of four elements: the variable, the variant, the tablex and the graph. The basic element being of course the variable, we will now see how to initially enter raw data into the database.

The first task at hand is to define the **number and type of observations** for all variables in this new database. Expand the pull down menu by clicking on the + sign before one of the model types listed in the left window. Left click on the Variables item to highlight it, then right click to view the pop up window. This window is specific to each regression class and basically allows the user to define the number of observations and the index of the first observation. This window also allows the user to provide labels that will replace the running index of the observations in the graphic routines. Return to the desktop by clicking the OK button.

The next task is to **enter the variable names**. Again right click the highlighted Variables item. In the pop up menu seen previously, now click on New -> Variable and enter the name of your first variable. A description and a comment may also be entered for this variable. Terminate the variable creation by clicking the OK button. Repeat these steps for each variable: the names entered should now be listed in the right-hand part of the TRIO window.

When all variable names have been created, the final task is to **enter the values or raw data**. Two methods are available to the user: Copy&Paste raw data from a Windows application or, for small numbers of observations, entering the values directly into the TRIO database.

1.5.1 Copy&Paste values from a Windows application

Again right click the highlighted Variables item and then click on **Data Listing**. This will bring to screen a matrix of cells similar to that seen in an Excel worksheet with all cells being empty. Raw data may now be copied into the cells: open the Windows application where are stored the values, copy the values to the clipboard and then paste from the clipboard into the TRIO Data Listing matrix.

1.5.2 Enter values directly into the the Data Listing

Right click the highlighted Variables item and then click on **Data Listing**. Click on any cell, press Enter, enter a value and again press Enter. Move between cells with the arrow keys or the numeric pad keys.

1.6 ALGEBRAIC EXPRESSIONS

TRIO provides the user with a powerful variable creation tool: a built-in algebraic expressions calculator. An algebraic expression may contain: an existing variable name, keywords, real or integer constants, algebraic, relational or logical operators, and intrinsic functions. An algebraic expression containing a variant result is not yet supported. The following tables list the operators, constants, keywords and functions available in TRIO.

Table 1.3: Unary Operators

Operator	Argument type	Result Type	Comment
+	float or integer	same as argument	identity sign
-	float or integer	same as argument	negative sign
not	logical	logical	reverses the logical value of the argument

Table 1.4: Algebraic constants

Name	Type
pi	float
TRUE	logical
FALSE	logical
YES	logical
NO	logical

Table 1.5: Algebraic keywords

Keyword	Interpretation	Variable Class
#t	Index for the periods	Level
#o	Index for the observations	Share
#i	Index for the individuals	Probability

In an algebraic expression, the keywords represent the running index of the observations, one for each variable class.

In evaluating an algebraic expression, the following **order of precedence** is applied:

1. parentheses, functions, constants, not, numerical signs +-
2. ^
3. * / mod
4. + -
5. <=, >=, ==. !=, <>, >, <, =
6. AND
7. OR, XOR

The algebraic expression is evaluated for each index of the variable. If one of the variables from which is derived the algebraic expression has a missing value for a particular index, then the resulting variable will list the message "Error in Reference" in the corresponding Data Listing cell. The following table lists the binary operators where x1 and x2 represent variable names, keywords, constants or algebraic expressions and v is a variable name.

Table 1.6: Binary Operators

Operator	Example	Argument Type	Result Type	Comment
+	x1 + x2	float or integer	same as argument	addition
-	x1 - x2	float or integer	same as argument	subtraction
*	x1 * x2	float or integer	same as argument	multiplication
/	x1 / x2	float or integer	float	division
^	x1 ^ x2	float or integer	float	x^a: raise x to the power a
mod	x1.mod.x2	integer	integer	remainder after division
and	x1.and.x2	logical	logical	true if both operands are true
or	x1.or.x2	logical	logical	true if at least one operand is true
xor	x1.xor.x2	logical	logical	true if exactly one operand is true
>=	x1>=x2	float or integer	logical	greater than or equal to
<=	x1<=x2	float or integer	logical	less than or equal to
!= <>	x1!=x2	float or integer	logical	not equal to
= ==	x1==x2	float or integer	logical	equal to
<	x1<x2	float or integer	logical	less than
>	x1>x2	float or integer	logical	greater than
These operators are for backward compatibility only and should no longer be used				
.ge.	x1.ge.x2	float or integer	int(0:1)	greater than or equal to
.le.	x1.le.x2	float or integer	int(0:1)	less than or equal to
.ne.	x1.ne.x2	float or integer	int(0:1)	not equal to
.eq.	x1.eq.x2	float or integer	int(0:1)	equal to
.lt.	x1.lt.x2	float or integer	int(0:1)	less than
.gt.	x1.gt.x2	float or integer	int(0:1)	greater than
.min	x1.min.x2	float or integer	same as arguments	the smaller value of two values
.max.	x1.max.x2	float or integer	same as arguments	the greater value of two values
.case	v.case.n	n integer	type of v	v.case.n returns the n^{th} value of v

The following table lists the available intrinsic functions. If a function has more than one argument, they are separated by the semicolon (;).The argument of an intrinsic function may be a variable name, a constant, an algebraic expression or a another function except for the lag and lead functions. For these two functions, the first argument must be a variable name, the second optional argument is of course the number of periods shifted. The value of the shift may be negative. For example, the lag function with a negative valued shift will behave like the lead function.

Furthermore, any variable may be treated as a function. Passing the integer value n to a variable will return the n^{th} value for that variable. For example, X(8) will return the 8^{th} value of variable X.

Table 1.7: Intrinsic Functions

Name	Argument Type	Result Type	Comment
abs()	float or integer	same as argument	absolute value
bc(var;l)	var - variable, l- float	float	the Box-Cox transform of variable v is $v^{(l)} = \frac{v^l - 1}{l}$
cos()	radians	float	cosine function
erf()	float	float	normal error function $erf(x) = \left(\frac{2}{\sqrt{\pi}}\right)\int_0^x e^{-t^2} dt$
exp()	float	float	exponential function
frac()	float	float	non-integer part of a floating point value
if(c;a;b)	c - logical a and b same type	type of a and b	ic c (the condition) is true, returns value of a, otherwise returns the value of b
int()	float	integer	truncation returns integer part of a floating point value
lag(var)	var- a variable name	the type of var	returns the value of the previous period of a variable
lag(var;shift)	var - a variable name shift- integer	the type of var	returns the value of var(t-shift) where t is the period index and shift is the backward shift
ld()	float > 0	float	binary logarithm
lead(var)	var - a variable name	type of var	returns the value of the next period of a variable
lead(var;shift)	var - a variable name shift - integer	type of var	returns the value of var(t+shift) where t is the period index and shift is the forward shift
lg()	float > 0	float	decimal logarithm
lgam()	float ≥ 1	float	natural logarithm of gamma function
ln()	float > 0	float	natural logarithm
min(a;b;…)	all float or integer; at least two arguments	same as arguments	returns the minimum of all values
max(a;b;…)	all float or integer; at two arguments	same as arguments	returns the maximum of all values
sin()	radians	float	sine function
sqrt()	float or integer ≥ 0	same as argument	square root
tan()	radians	float	tangent function