

TOL USER MANUAL

www.tol-project.org

Bayes  Forecast

Escuela Bayes de Estadística y Modelación, S.L.

Gran Vía, 39, 5ª planta

28013 Madrid (España)

Tel. (34) 915327440

Fax. (34) 915322636

www.bayesforecast.com

www.tol-project.org

Contents

I. **Introduction**

1. Approach of this manual 6
2. How to unload the language?
3. Surroundings of work
4. Types of TOL files
 - 4.1 How to write a TOL temporal series?
 - 4.2 How to generate a TOL table?

II. **Main TOL operations**

1. Types of variables available in TOL
 - 1.1 Text
 - 1.2 Real
 - 1.3 Date
 - 1.4 Complex
 - 1.5 Matrix
 - 1.6 Set
 - 1.7 Serie
 - 1.8 TimeSet
 - 1.9 Polyn
 - 1.10 Ratio
 - 1.11 Code
 - 1.12 Anything
2. Access in data bases
3. Reading / Writing files
 - 3.1 Reading
 - 3.2 Writing on screen

III. **Programming in TOL**

1. Why the characteristics of TOL?
2. Functions
 - 2.1 Create a function with TOL
 - 2.2 Compile a function with TOL
 - 2.3 Logical Functions
3. Sentences of prog
 - 3.1 EvalSet()
 - 3.2 For()
 - 3.3 If()
 - 3.4 Case()

IV. **Set**

1. Definition of sets
 - 1.1 Simple sets
 - 1.2 Sets of sets
 - 1.3 Structured sets
 - 1.4 Other definitions of sets
2. Operations with sets

V. **TimeSet**

1. Daily representation
2. Algebraic operations
3. Functions
 - 3.1 Successor
 - 3.2 Range
 - 3.3 Periodic intervals

VI. **Series**

1. Operations, functions and polynomials

- 1.1 Operations between series
- 1.2 Series Functions
- 1.3 Polynomials
- 2. Deterministic series in TOL
- 3. Introduction in modelling
 - 3.1 Reading data
 - 3.2 Identification phase. Visualisation
 - 3.3 Estimation Phase
- 4. Validation Phase
- 5. Prediction Phase

VII. Other types of variables

- 1. Functions and operations with: Polyn
- 2. Functions and operations with: Ratio
- 3. Functions and operations with: Text

VIII. General recommendations in programming

- 1. Organization of a TOL archive
- 2. Structure of a function
 - 2.1 Header
 - 2.2 Body
 - 2.3 Special case: Clause IF
 - 2.4 Description

APPENDIX

Generation of files BDT with Excel

Cycles with While

Copyright	2008, Escuela Bayes de Estadística y Modelación, S.L.
Título	TOL User Manual
Edición	13/08/2008 13:51:00
Claves	TOL, manual, training

I. Introduction

TOL (Time Oriented Language) is a language oriented to the operation of the temporary information and the construction of systems of dynamic attention of demand, through an algebraic representation of the time. Among its objectives are:

- a) To organize the data according to a temporary structure in order to give them temporary meaning.
- b) To analyze the temporary data in order to study behaviours passed through the estimation and to foretell future behaviours, through the forecast.

It is oriented to the analysis and operation of temporary data, in order to model and to solve real problems in the scope of dynamic information. Moreover a user surrounding is added, Tolbase, to facilitate the analysis of data and the development of models of temporary series, tabular matrices, graphical representations, etc. Among the existing characteristics of TOL, it is a language that:

- a) Interprets, which facilitates the learning and use for the analysis of data, since it allows the easy modelling of temporary series.
- b) Declares and in addition it incorporates a type of Anything variables that allows to work with different types of variables.
- c) Auto evaluates, that is to say, it is possible to evaluate the TOL code that is constructed in the execution time, which is very useful when a code for great amounts of data with different structures is used.
- d) Slowed down evaluation (Lazy). This is specially adapted when the information is infinite.

Some of its technical aspects are:

- a) It is developed in C++, which is one of the most powerful and versatile languages that exist and its direction to objects.
- b) It is developed with multiform vocation and the code has been compiled under Windows and UNIX.

1. Approach of this manual 6

The objective of this manual it is to orient those people who begin programming in TOL. The manual begins with a description of the language, how to obtain the software and the surroundings of work. Both of the following chapters are focused on programming with TOL language. The remaining chapters describe the types of TOL objects for the utilization of the temporary information.

Throughout this manual examples of programming are included; reason why the reader is expected has a personal computer with TOL and is executing these examples. This manual describes the use of TOL under Windows XP.

2. How to unload the language?

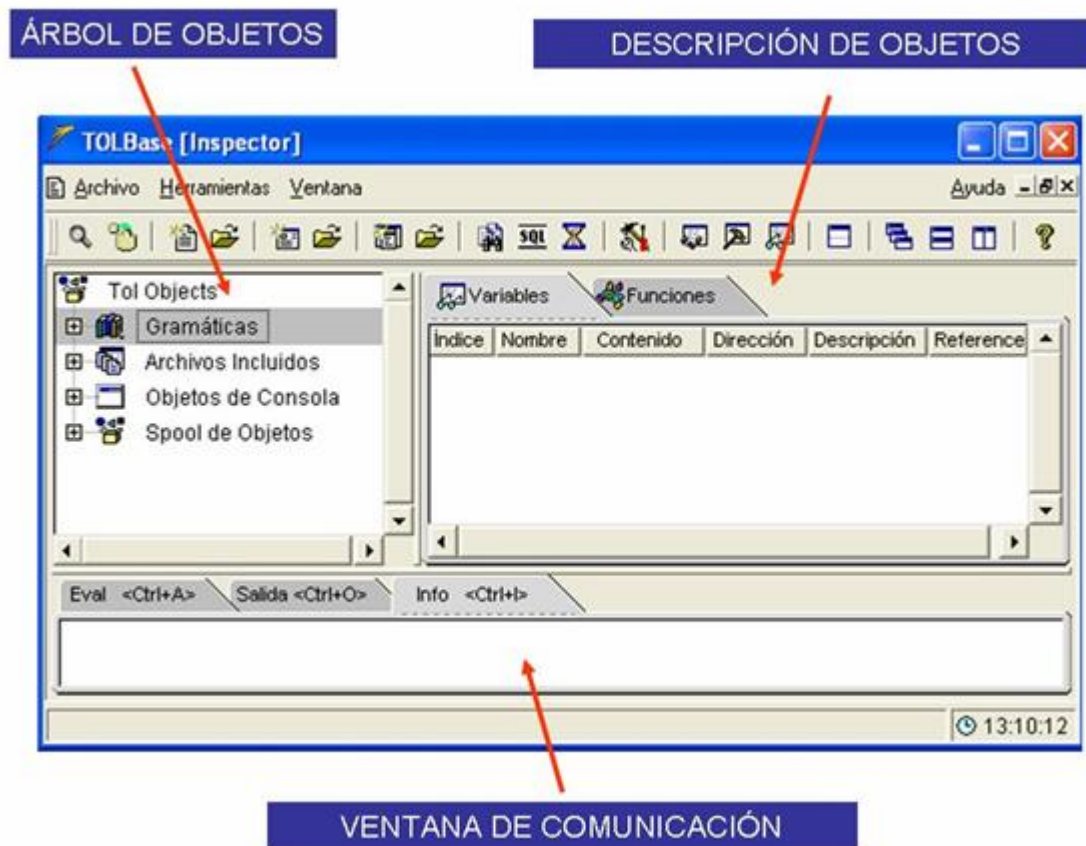
The following Internet address <http://www.tol-project.org> allows accessing the development community of TOL language. From this page it is possible to access the forums of discussion for the development of the program, announcements, chats, report of failures, improvements, etc. and the gratuitous unloading of software for Windows, Linux and surroundings of programming (Tolbase) developed in C++.

Depending on the TOL version it might need greater or smaller memory. In this manual we will follow the version for Windows.

The page also offers an online service to respond to the most frequent questions, as well as to solve possible installation errors.

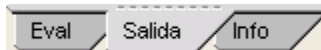
3. Surroundings of work

TOL can be considered as any other application of Windows, pressing the right mouse button on the next icon, located in the folder bin within the TolBase*.* folder in which the program *C:\Archivos de Programa\Bayes\TolBase*.** is unloaded. When starting TOL appears the next window. This initial window requires a more detailed description.



The window of description of objects, shows to the variables and functions created during the compilation process. Different files can be compiled and decompiled of independent different form. This is very useful when we have, for example, a set of functions defined in a file A and we want to use them for another set of sentences in a different file B. Therefore, the functions defined in the file A will be available for all the following files as long as we do not decompile the file A. And we will be able in the meantime to work with file B, compiling and decompiling as many times as it is necessary having the functions of the file always available.

The window of communication consists of three sub windows:




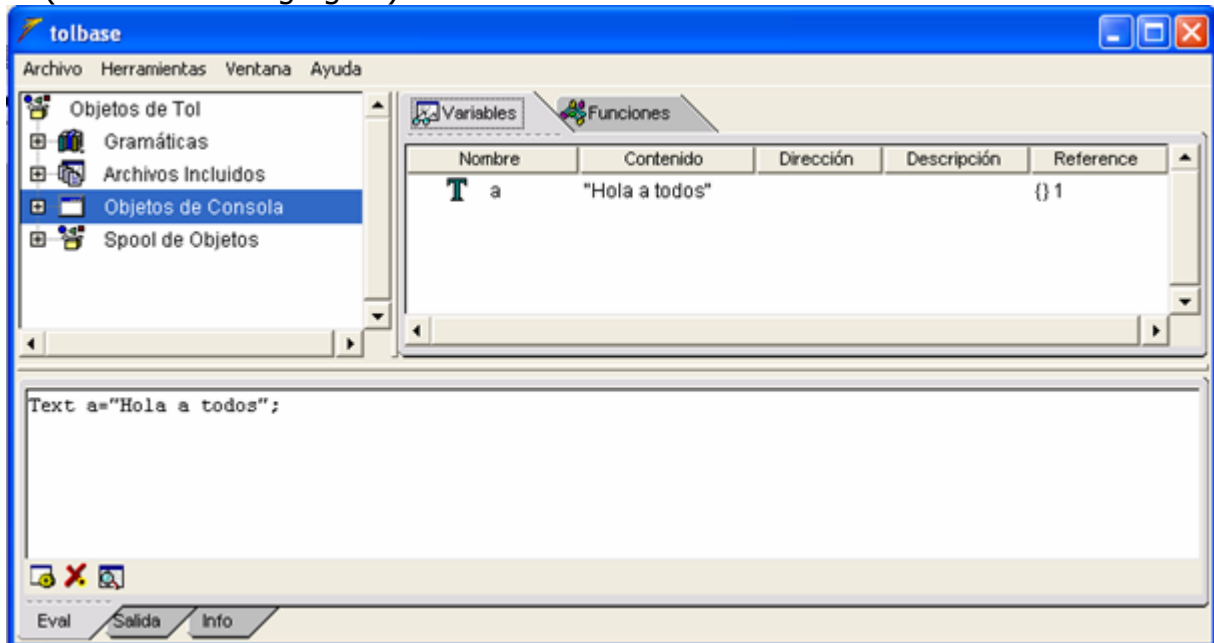
- **Eval:** this window allows us to write sentences in TOL code. Each sentence must be separated of the following one by semicolon (;). In order to execute the sentences, we press compile. If we have compiled previously, it is necessary to decompile first (to destroy the variables that have been created when compiling).
- **Output:** It shows the errors of compilation, the information of the files including and the messages that emit the programs
- **Info:** It shows to the information relative to the variables or functions created during the compilation process.

Example

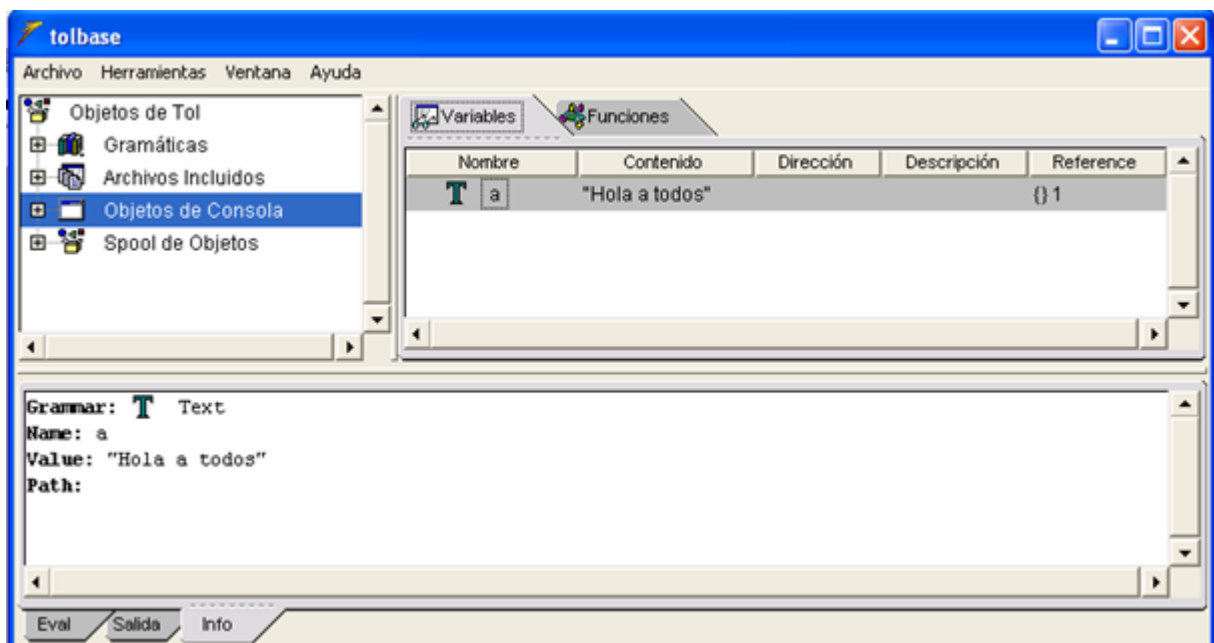
We write in the Eval window the following sentence:

Text a="Hola a todos";





We press compile . In the variables window the new created variable is obtained to (see the following figure).

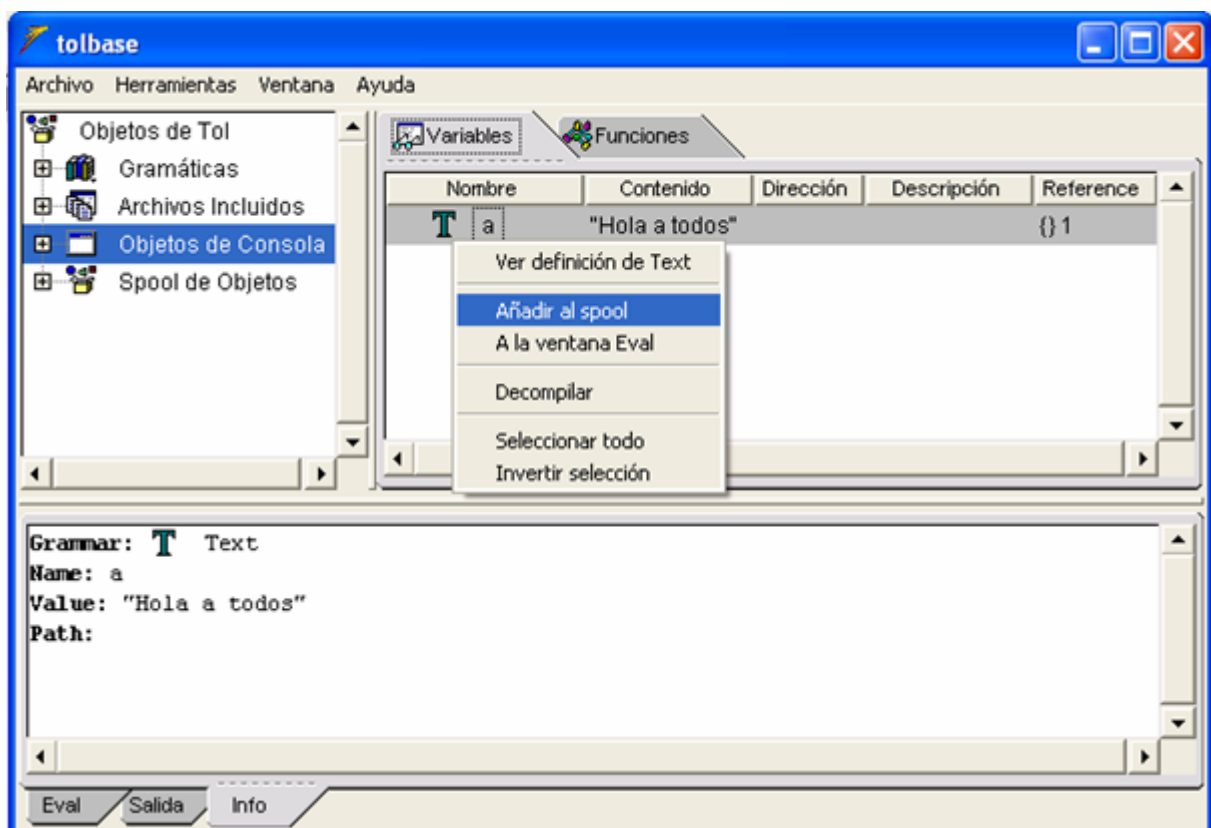



If we select the Info tab, and it is pressed on the created variable, the information relative to the mentioned variable is presented (see the following figure): Type of object (**Grammar**), name (**Name**), value (**Value**) and file in which the variable is defined (**Path**) (In this case, since we have written the code directly in Eval window field is left empty).



The tree of objects shows the types of objects available:

- **Grammars** : Types of variables and functions available in TOL to write sentences organized by data types.
- **Included Files** : It shows the archives included in the compilation process from a TOL file. When initiating TOL, the set of files *_inittol.tol* is included.
- **Console objects** : It shows the variables, functions and archives included in the compilation process from the Eval window
- **Objects' Spool** : It serves to take objects from the three previous options and to be able to visualize them or to work with them. For example, if one is interested in painting the graph of a series created from the **Eval** window along with another series created from a TOL file. In this case, both series are added in **Objects' Spool** by selecting each series with the right button of the mouse and pressing the left button. In the context menu that unfolds, we chose the option **Add spool**.



Every time we decompile , we eliminate the Objects' Spool.

Finally, the top toolbar allows the following options:

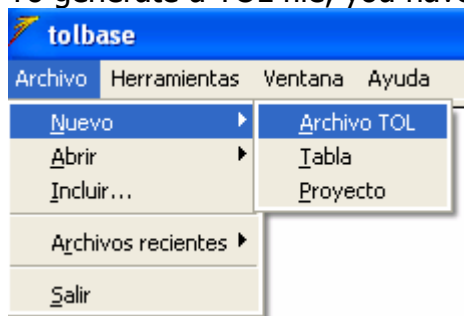
- **File**: It allows to create, to open, to include TOL's recent files.
- **Tools**: It allows to edit files, tables and TOL projects, to look for functions and variables between all the files included in the processes of compilation, to select the

types of calendars, manipulation of data bases through SQL, data connections and options of format of numbers and calendars.

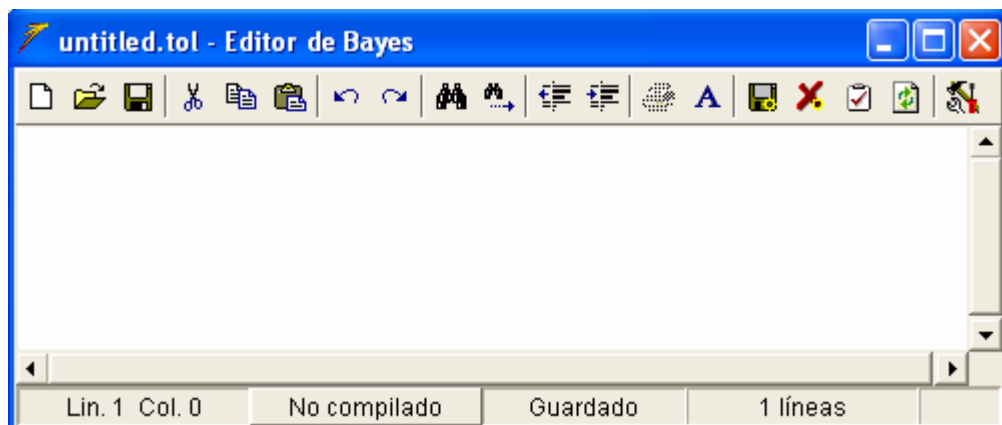
- **Window:** options of visualization of the TOL windows
- **Help:** it describes the TOL version.

4. Types of TOL files

The most important type of in TOL, are the files of code. In them, we can write sentences to execute them later and to save the work to be made in future sessions To generate a TOL file, you have to go to File - New - TOL File:



We have the following code editor:



The editor allows you to open, to save, to copy, to stick, to cut, etc among other options of editing and compile and decompile the generated sentences. In addition, it allows checking the syntax and options of syntax editing .

In the following sessions, we will use this type of files to save the examples. Therefore, we save this file (for example: in the folder *C:\Bayes*) with an identificating name (for example: *Ejemplo.tol*).

In addition, TOL possesses its own methods of storage and recovery of the information, in the form of files **BDT (Bayes Data Table)** for the definition of temporary series or **BST (Bayes Struct Table)** for the definition of tables.

The entry information or the results obtained with TOL can be exchanged by systems of management of databases or by systems of office automation like spreadsheets or processors of text.

TOL also possesses its own methods of presentation of results in the form of graphs and tables.

4.1. How to write a TOL temporal series?

The format that TOL waits is the following one:

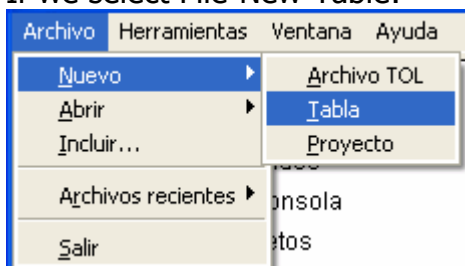
```
FechadoSerie;
NombreSerie1;
NombreSerie2;
...NombreSerieN
Fecha 1; dato S11; dato S21;...;dato SN1;
Fecha 2; dato S12; dato S22;...;dato SN2;
...
Fecha k; dato S1k; dato S2k;...;dato SNk;
```

The format of the dates is year / month / day (aaaa/mm/dd). If we have yearly series, where it is not necessary to indicate either the month, or the day, it is possible to fix the month in January (mm=01) and the day on the 1st (dd=01). This format corresponds with files **BDT (Bayes Data Table)**.

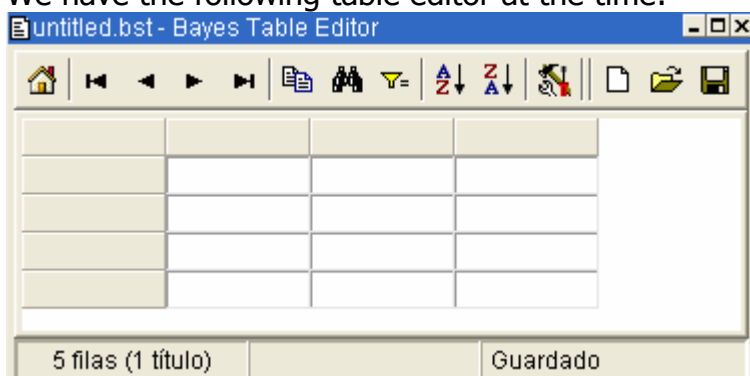
Example: We want to generate a table of information that contains the following fields:

Monthly	Sales	Purchases
January 2005	3.250.000	200.400
February 2005	4.130.000	315.800
March 2005	4.250.000	285.760
April 2005	3.400.000	320.150

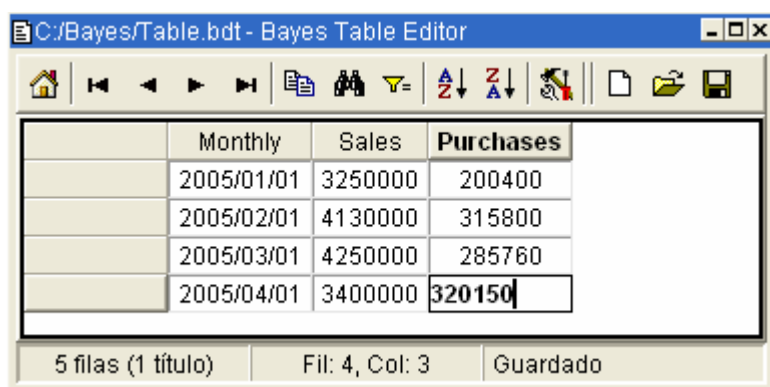
If we select File-New-Table:



We have the following table editor at the time:



The first row must contain the dates and the names of the series. We introduce the data as it continues and we select to save this file (for example: in the folder **C:\Bayes**) as **Bayes Data Table** with an identifying name (for example: *Table.bdt*)



To work with this file, which has format of TOL table (format **BDT**), we write in a TOL file the following judgment. We click on Save and compile  (specifying a name to identify later the file, for example *Exampleincludedate.tol*)

Set Table = IncludeBDT(Text "C:\Bayes\Table.bdt");

This sentence indicates that our file has format of table BDT, the separation of the different columns is indicated by semicolon (;) and the data that begin in January of 1999 (y1999m01d01) and finalize in April of 1999 (y1999m04d01).

4.2. How to generate a TOL table?

The format that TOL waits for is the following:

Nombreestructura; Variable1; Variable2;...; VariableN;
Numeracióncampo1; dato S11 ; dato S21 ;...; dato SN1;
Numeracióncampo2; dato S12 ; dato S22 ;...; dato SN2;
...
NumeraciónCampoN; dato S1k ; dato S2k ;...; dato SNk;

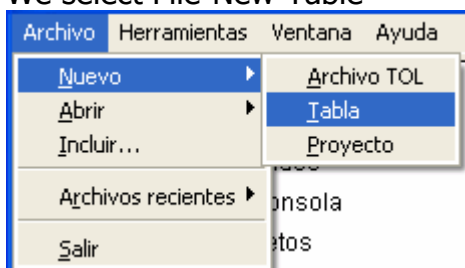
This format corresponds with files **BST (Bayes Struct Table)**.

The numeration of the fields is not obligatory, actually, you may start the file only with ;.

Example: We want to generate a table of information that contains the following constructed fields:

Name	Surname	Age
Alberto	Ruiz	14
Carlos	Álvarez	43
Maria	Gómez	32
Ana	González	67

We assign a name to the structure of the table of information: for example *Personaldata*, which contains a field of text (*Name, Surname*) and a Real field (*Age*). We select File-New-Table




We obtain the following table editor:



The first row must contain the name of the structure and the names of the fields. The fields of the variables of text must go within quotation marks (" "). We introduce the information as following and we select to save this file (for example: in the folder

C:\Bayes) as **Bayes Struct Table** with an identifying name (for example: *Nametable.bst*)



PersonalData	Name	Surname	Age
	"Alberto"	"Ruiz"	14
	"Carlos"	"Alvarez"	43
	"Maria"	"Gomez"	32
	"Ana"	"Gonzalez"	67

To work with this file, which has the format of table TOL (format BST), write in a TOL file the following sentences. We click on Save and compile (specifying a name to identify later the file, for example *Examplecreatetable.tol*): *// Create the structure with personal data.*

```
Struct Personaldata
(
```

```
Text Name,
Text Surname,
Real Age
```

```
);
```

// Open the table that contains the data with Personaldata structure.

```
Set Table = IncludeBST(
```

```
Text "C:\Bayes\Ejemplos_Manual\Cap01_Introduccion\Nametable.bst");
```

This judgment indicates that our file has format of table BST.

II. Main TOL operations

In this chapter we briefly describe the types of variables that are available in TOL and the main TOL operations.

1. Types of variables available in TOL

TOL is a typical language oriented for operations with series and temporary sets although they have other many data types of texts, real, etc. The types of objects available are the following ones:

1.1. Text



Text an object of text type can contain any chain of characters ASCII. The form to create a variable of type text is to put the text in inverted commas. **TOL** allows diverse treatments of the text variables (search of chains, transformations, etc.)

Example *Text Myname = "Ana";*

1.2. Real



Numerical variables, that are defined as real of double precision. Also the unknown value is included (?), infinitely or indeterminately. The separating decimal is the point. **TOL** incorporates multiple functions of manipulation (sum, product, etc.) and transformation of real (transformation trigonometric, statistical, etc.).

Example

Real Fract = 2/3;
Real Integer = 4;
Real Unknown = ?;
Real Infinite = 1/0;

1.3. Date



Dates. The format of the dates is y2004m01d23 (aaaa/mm/dd). All the fields are necessary, being able to eliminate character 0. If we have a yearly series, that it does not need to indicate nor the month, nor the day, is possible to fix the month to January (mm=01) and day 1 (dd=01).

Example *Date Meeting = y2005m05d03;*

1.4. Complex

Complex numbers. The imaginary part of a complex number is indicated by the variable *i*.

Example *Complex Aurea = SqRt(5)*i;*

1.5. Matrix



Matrices of real numbers. TOL incorporates multiple manipulation functions (sum, product, etc.), transformation of matrices (transformation of Choleski, etc.) and resolution of linear systems.

Example

Matrix a = SetMat([[[2,3,4]], [[1,2,1]]]]);
Matrix b = Cos(a);

1.6. Set



Sets. They are collections of objects of any type including other sets, which allows constructing structures like vectors, tables, trees, etc. TOL allows the treatment and manipulation of the sets focused on the work with temporary series.

Example *Set Data = SetOfSet(SetOfText("Hello", "Welcome"), SetOfReal(1,2,3));*

1.7. Serie



Temporary series. TOL allows the treatment and the algebraic manipulation of temporary series, to conduct operations, to create graphs, to estimate models and to predict data.

Example *Serie PulseY2005 = Pulse(y2005m01d01, Yearly);*

1.8. TimeSet



Temporary sets, which are subgroups, finite or infinite, of the set of all the dates from the beginning until the end of the time. A temporary infinite set could be, for example, all Mondays that fall in first day of a month, all Fridays and thirteen, all Sundays of Easter, etc. TOL allows operating with the temporary sets to construct new sets, something that is very useful to handle temporary series.

Example *TimeSet Friday13 = WD(5)*D(13);*

1.9. Polyn



Polynomials. The main utility is that they allow to the temporary displacement of the series through retardation operators or advance. TOL allows operations between polynomials such as the product, sum, powers, etc.

Example

Polyn Stepwise = F+B^{2+F}3+B^{4+F}5+B^{6+F}7;

Serie StepStepwise = Stepwise:Step(y2004m3d01, Monthly);

1.10. Ratio



Rational fractions of polynomials. A rational function of retardations is defined as a quotient of retardation polynomials. Its main utility is to solve equations in differences of the type $P(B)Z_t = Q(B)A_t$. TOL allows operations between rational fractions such as the product, sum, powers, etc.

Example

Ratio Fraction = F/(1-B⁵);

1.11. Code



TOL code. The functions available visualize in the variables of the window of the inspector of objects.

1.12. Anything

?

It admits any type. This is very useful to handle the functions associated of type *Case()*, *Eval()*, *Field()*, *Find()*, *If()*, *While()*, etc.

2. Access in data bases

In order to import and to manipulate a data base, **TOL** incorporates the connection to a remote origin of data through **ODBC**.

In order to execute consultations to data bases, first they must be opened. For this, we used the sentence *Real DBOpen(Text alias, Text usuario, Text clave)*

An alias to the given **ODBC** data base must exist of discharge in **DSN** of Windows/Unix. This function returns **1** (true) in case of success and **0** (false) in case of error. Among the diverse options of handling data bases, they are:

a) *DBExecute(Text consulta)*, that allows to update and to modify the data base through SQL sentences.

b) *DBSeries(Text consulta, TimeSet fechado, Set nombres)*: It includes a set that contains the series with the names (*Set nombres*) and in the indicated dated one (*TimeSet fechado*) whose data come dices by a consultation to an open data base (*Text consulta*).

c) *DBTable(Text consulta [, Text NombreEstructura])*: It includes a set that contains a table as a result of the consultation made to the open data base (*Text consulta*). The second element equips with the structure to each one of the registries of the table. This structure must be defined previously (*Text NombreEstructura*).

d) *DBSpool(Text consulta, Text fichero)* allows to save in a file (*Text fichero*) the consultation made (*Text consulta*) in the base of data.

In order to end the consultations to the data bases, these must be closed. For it, we used the command *Real DBClose(Text alias)*

This function returns **1** (true) in case of success and **0** (false) in case of error.

3. Reading / Writing files

3.1. Reading

In order to include files of data (series, matrix, table, etc.) generated from **Tolbase** we use the command `Include` which generates a set (🍌 set). Between the most useful forms of inclusion, they are:

- a. *IncludeBDT*: It includes the set of temporary series with the same date and between the same dates defined with format **BDT (Bayes Data Table)**.
- b. *IncludeBMT*: It includes the set of a matrix with format **BMT (Bayes Matrix Table)**.
- c. *IncludeBST*: It includes the set of a structure with format **BST (Bayes Structured Table)**.
- d. *IncludeTOL*: It includes the set of objects of a file **TOL**. The tabulators, breaks of line and consecutive characters of space are equivalent to an only character of space.
- e. *BDTFile*(Set cto [, Text NombreFichero , Text cabecera]): it generates a file of the form **BDT (Bayes Data Table)**, , that is importable and exportable from system and spreadsheets of management of data bases.
- f. *BSTFile*(Set cto, Text NombreFichero): it generates a file with structure **BST (Bayes Structured Table)**, for the storage of structures of data of any type in table form.

For the inclusion of **code files**, it is sufficient to indicate the access route through the *Include()*. function. The object that is generated is of type *Set*.

Example

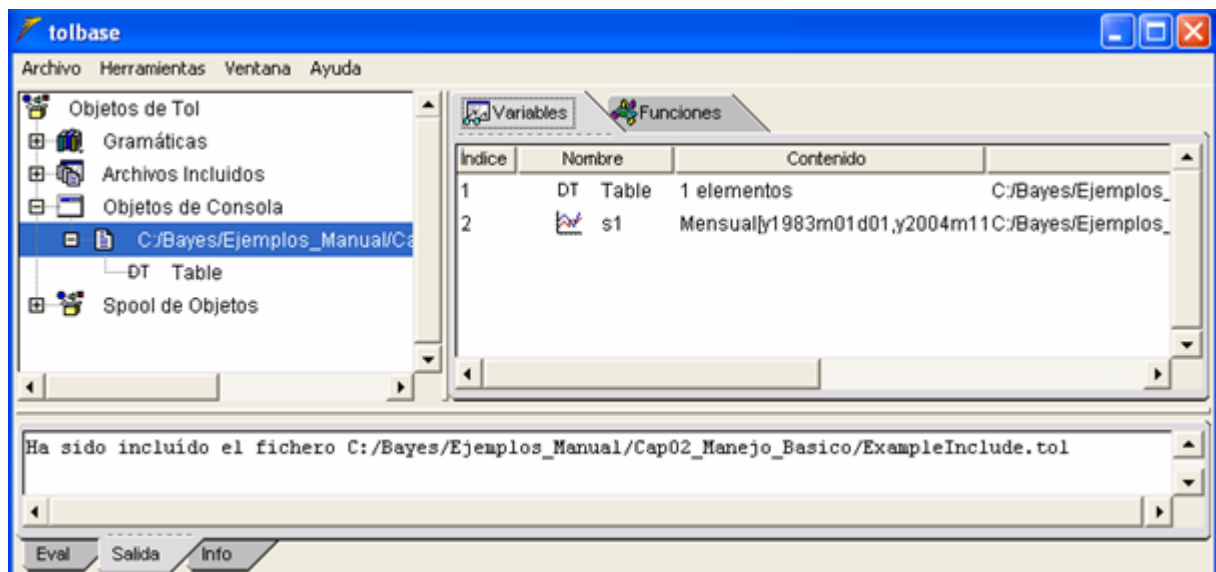
To include the file *ExampleInclude.tol*, we verified that it is located in

C:\Bayes\Ejemplos_Manual\Cap02_Manejo_Basico.tol

In order to add the file, write in the Eval window: *Set*

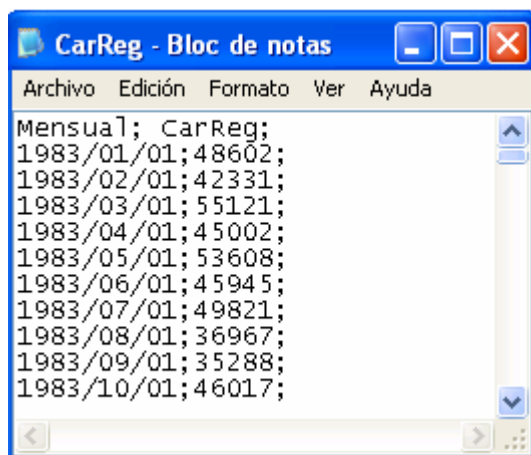
Include("C:\Bayes\Ejemplos_Manual\Cap02_Manejo_Basico\ExampleInclude.tol");

The result is a file that is added in Included Files, as the following figure shows:



Example

We are interested in the reading and including the data of a monthly series in the following file



We verified that the file *carreg.bdt* is located in
C:\Bayes\Ejemplos_Manual\Cap02_Manejo_Basico.tol

In order to add the file, write in the Eval window:

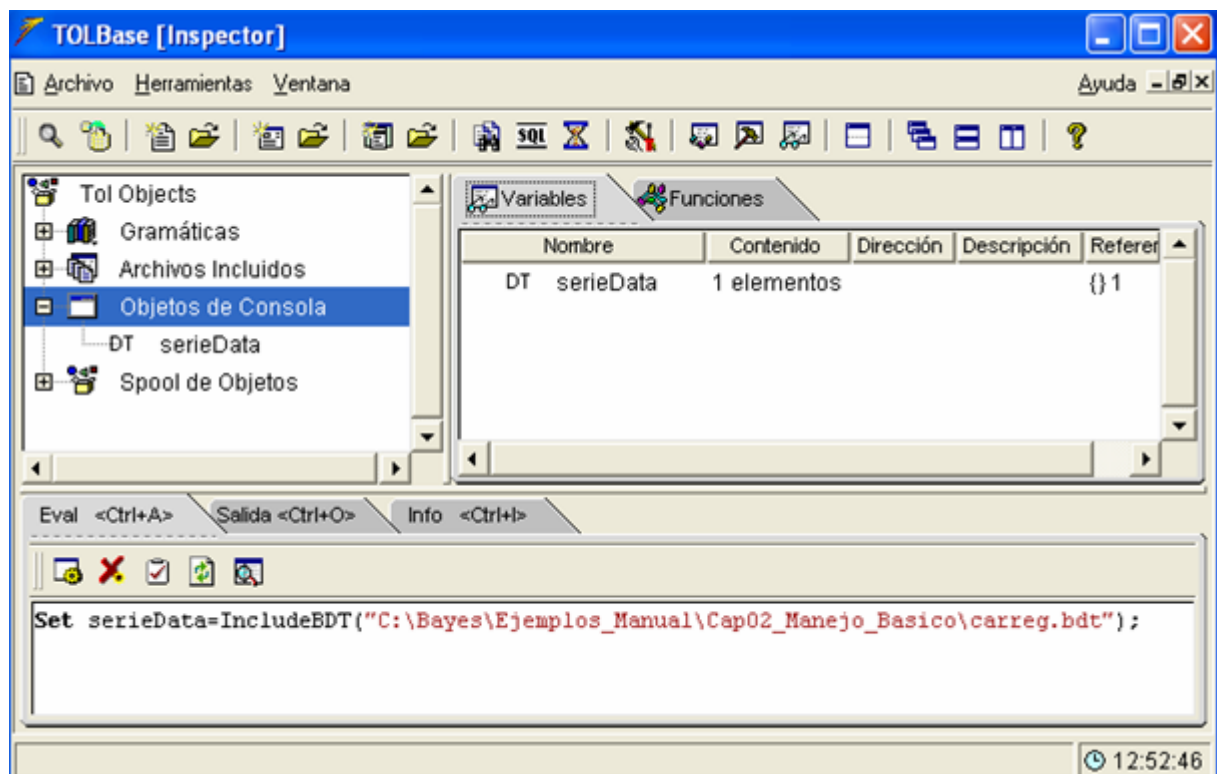
Set

```
serieData=IncludeBDT("C:\Bayes\Ejemplos_Manual\Cap02_Manejo_Basico\carreg.bdt");
```

We observe that:

a. In the file *CarReg.bdt* a series *CarReg* is monthly defined *Monthly (Mensual)*. This is specified by means of the head of the file (*Monthly; CarReg;*).

b. After loading this BDT file, the series *CarReg* is available for its manipulation and use. Once the data is loaded, the content of the file *carreg.bdt* can be consulted through the **inspector of objects** of the **Tolbase** surroundings. After the execution of the sentence *IncludeBDT()* a set is created (*Set*) under the name *serieData* that contains the series *CarReg*. In the **inspector of objects**, on the right of the series, it is possible to consult the temporary set for which the series are defined (*Monthly*) and the date of the beginning and the aim of the data (from day 1 of January of 1,983 to day 1 of November of 2004).



3.2. Writing on screen

For the writing in the terminal or output window, we can use:

- Function *Write()* that writes in the terminal the given text.
- Function *WriteLn()* that writes in the terminal the given text and jumps a line.

Example

When writing the following set of sentences in the Eval window:

```
WriteLn("");
Write("Hello ");
Write("world this is message 1");
WriteLn("");
WriteLn("Hello ");
WriteLn("world this is message 2");
```

We obtain the following messages in the output window

Hello world this is message 1
Hello
world this is message 2

For the writing in files in free format, we can use:

- *WriteFile()* that supwrites a file with a text, if it has been created previously or it creates it if it does not exist.
- *AppendFile()* that adds a text given at the end of a file, if it has been created previously or it creates it if it does not exist.

When writing the following set of sentences in the Eval window:

```
Text WriteFile("C:\Bayes\Ejemplos_Manual\Cap02_Manejo_Basico\file1.txt", "Hello  
world this is message 1\n");  
// \n : NewLine  
Text WriteFile("C:\Bayes\Ejemplos_Manual\Cap02_Manejo_Basico\file1.txt", "Hello  
world this is message 2\n");  
Text WriteFile("C:\Bayes\Ejemplos_Manual\Cap02_Manejo_Basico\file2.txt", "Hello  
world this is message 3\n");  
Text AppendFile("C:\Bayes\Ejemplos_Manual\Cap02_Manejo_Basico\file2.txt", "Hello  
world this is message 4\n");  
Text AppendFile("C:\Bayes\Ejemplos_Manual\Cap02_Manejo_Basico\file3.txt", "Hello  
world this is message 5\n");
```

We obtain three new files in *C:\Bayes\Ejemplos_Manual\Cap02_Manejo_Basico.tol*

III. Programming in TOL

In this section it is assumed that the reader has some knowledge of programming as we will focus more in the details of the language.

1. Why the characteristics of TOL?

One of the TOL characteristics is the slowed down evaluation (lazy). This is necessary for the handling of defined temporary series in infinite temporary sets and therefore it is required that the evaluation of the operations is the other possible use, when the evaluation concerns a finite subgroup.

For example, if we defined a temporary series that begins at the beginning of the calendar (*TheBegin*) and ends when it finishes (*TheEnd*):

Serie PulseElec = Pulse(y2004m03, Monthly);

TOL does not evaluate the expression, but it saves in memory the definition. If we wished to create a new series that is the double of the previous one:

Serie PulseElecDouble = 2 PulseElec;*

Again TOL waits to that the definition of both series is defined in a finite temporary interval. For example, we can limit the previous series in the period of time between the year 2000 and year 2005:

Serie PulseElec2000 = SubSer(PulseElec,y2000,y2005);

Serie PulseElecDouble2000 =SubSer(PulseElecDouble,y2000,y2005);

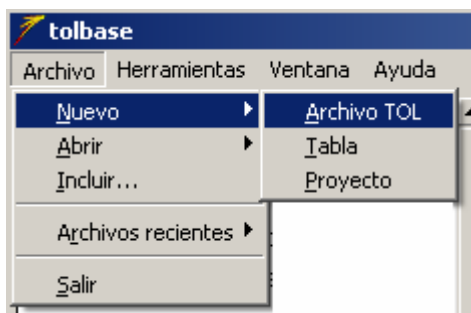
Now TOL evaluates the last expression in case of being necessary. In case of trying to evaluate an infinite series (*Serie PulseElec*) for example to create the graph of it, TOL internally limits the evaluation in a finite temporary set (*DefaultDates()*).

2. Functions

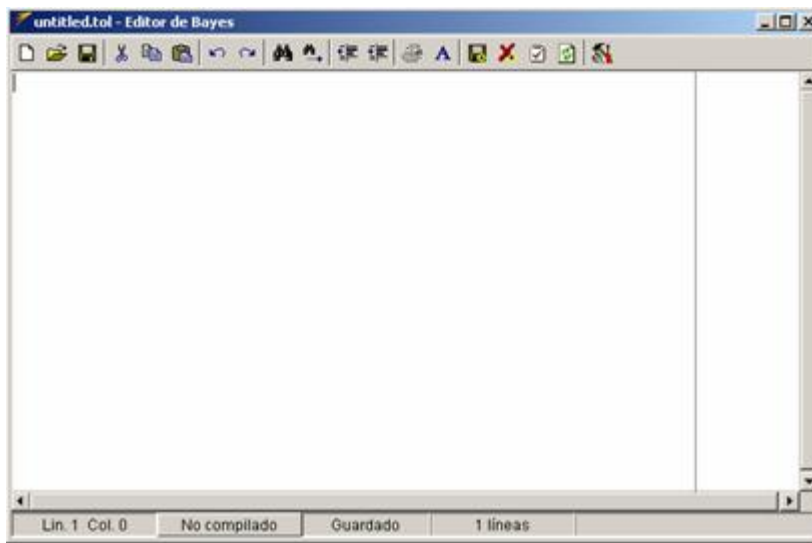
An essential part of the proper design of a computer program is its modularity; this is its division in smaller parts of very concrete purpose, **functions**.

2.1. Create a function with TOL


Next we are going to create a new function in TOL. For it, open a new file by clicking File - New - File TOL:

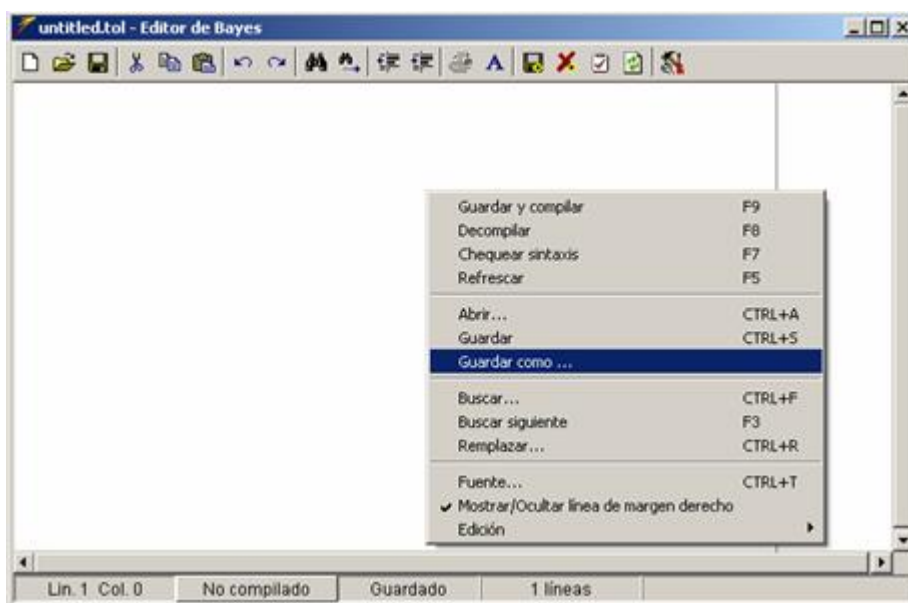


which from automatic form, receives the name *untitled.tol*:

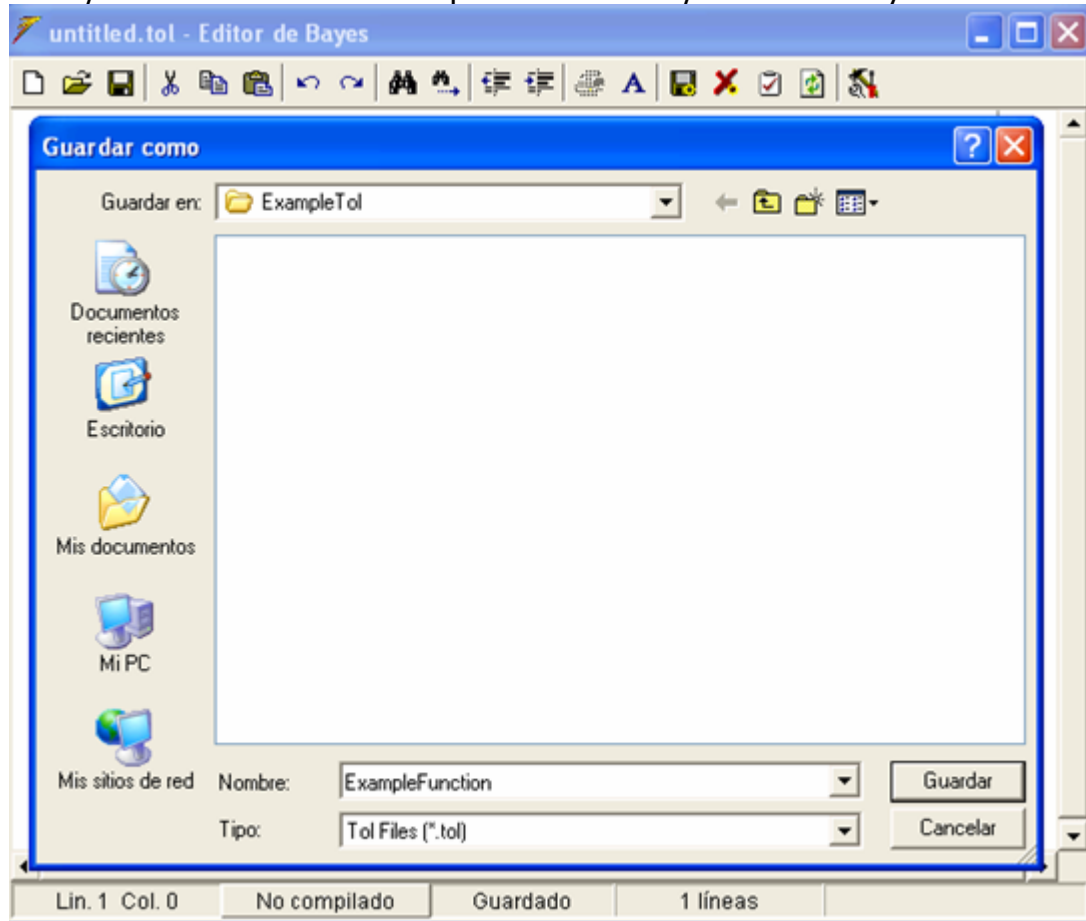


It is advised to give a name to this new file that allows us to identify it easily more quickly and to save it in the desired working directory before starting to write the new function. For that reason, create in the hard disk of the computer, a folder with the name *ExampleTol*. In order to save this file, it is possible to

- Use the save icon  from the main menu of the new file, or
- Use the option **Save As...** from the menu that appears when pressing the right button of the mouse on the editor of code of the new file:



A dialogue is displayed in order to save file TOL with the name we wish, in this case, *ExampleFunction.tol* and we kept it in the *ExampleTol* directory:



Observation: The **editor** of **TOL** code provided by the **Tolbase** surroundings is not the only text editor that allows us to create functions, projects and modules in TOL. It is also possible to use some other text editor available in the operating system in which one is working, for example in Windows is possible to use:

- The editor edit of DOS,
- The Notepad,
- The Wordpad, always saving as text or
- The processor of text, Word, always saving as text as in the previous case.

We can observe that one of the advantages that it provides to use the editor of code TOL it is that it provides a syntax heightened with the purpose of facilitating the reading of the code by the user:

- Differentiating the texts that go in inverted commas, "*text one*",
- Emphasizing in **Bold** the different types of objects (*Serie, Set, Text, TimeSet, Matrix, Ratio, Polyn, Real, Date y Code*), etc.

- The commentaries marked with the characters `//` or the characters `/* y */`, for example:

```

////////////////////////////////////
// Serie, time series,
////////////////////////////////////
The format to write a function is:
Tipo NombreFuncion(Tipo Argumento1,...,Tipo ArgumentoN)
{
  Sentencia 1;
  ...
  Sentencia K-1;
  Sentencia K
}

```

We must consider that TOL always returns the last sentence. It is more, for any set of sentences grouped between keys `{Sentencia 1;...; Sentencia K}`, TOL always returns the result of evaluating the last sentence. For that reason, the **last sentence does not finish in ; like the rest.**

This characteristic is the one that the operator **return** makes unnecessary in TOL like in other programming languages.

Example We want to create a function that returns the factorial of a whole number

```

////////////////////////////////////
Real FactInteger(Real x)
////////////////////////////////////
// PURPOSE : Function that computes the factorial of a given integer number
////////////////////////////////////
{
  If(x<=1,1,x*FactInteger(x-1))
};
PutDescription("This function returns the factorial number of a given
integer",FactInteger);
// We compute the factorial of 5
Real FactFive = FactInteger(5);

```

It is not necessary to assign a name to all the functions. Sometimes, we are interested in creating a function to assign a value solely to a variable.

Example We want to create a set that given the following set of real numbers `{3,6,2,-8,4,-5}`, takes the positive elements and in case of negative elements it returns the unknown value.


```

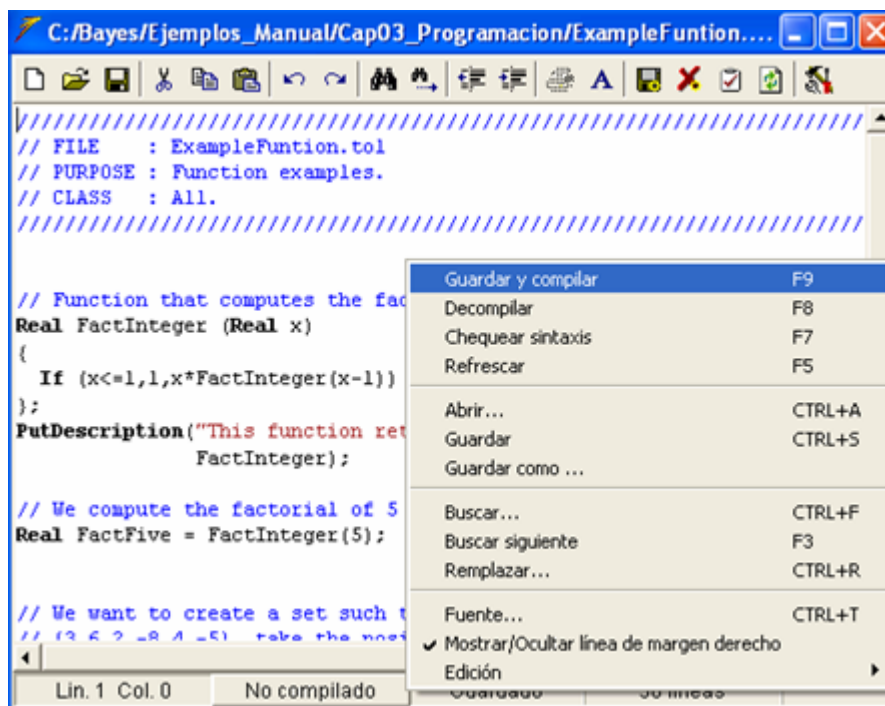
Set Numbers = SetOfReal{3,6,2,-8,4,-5};
Real maxi = 0;
Set MaxNumbers = EvalSet(Numbers, Real(Real x){If(x>=maxi,x,?)});

```

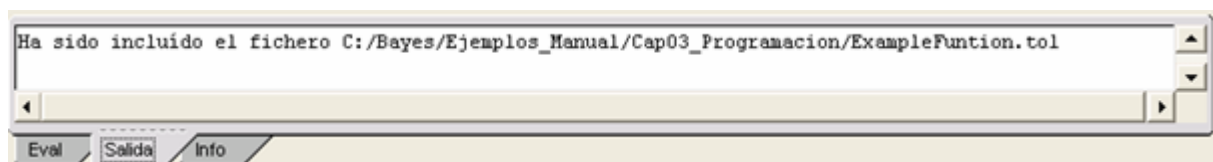
2.2. Compile a function with TOL

Once introduced the code in the file in order to compile it and to execute it is possible to:

- Use the icon, save and compile  of the main menu of Tolbase ,or as well
- Use the option Save and compile of the menu that appears when pressing the right button of the mouse on TOL editor.



In the output window, we can then see if we have committed syntax errors in the code, as well as the confirmation of which the file has been compiled and executed correctly,



Through the **inspector of objects** of the **Tolbase** surroundings the different objects created as a result of the compilation and execution of this file TOL can be consulted. In order to understand the operation of the inspector of objects we can make a parallelism between this and an inspector of archives of Windows:

- The inspector of objects shows the content of sets TOL as the inspector of archives shows the content of directories.

- For each object within a set are their properties as in the inspector of archives are the properties of the files.

Each TOL file is presented like a set that contains all the objects declared within it. In the following figure are the objects declared within the file *ExampleFunction.tol*:

Through the inspector of objects it can be observed, for example, that the denominated variable *Numbers* represents an object of type Set (*Set*) that contains six real numbers (3,6,2,-8,4,-5). The content of this set can be explored through the inspector of objects, as in the following figure:

Using the inspector of objects it is possible to create graphs, tables, to see the definition, among other options of the created variables. For example, a table with the content of a set can be constructed (*Numbers*). For it, we press the right button of the mouse on this object and obtain a menu with the applicable methods:

In the example, the set *Numbers* is made up of six real numbers, the result of its visualization in table form is in the following figure.

2.3. Logical Functions

The Logical Functions return true (it brings the value 1), false (false gives back 0) or not unknown (?). Given the importance that have the logical functions in the construction of programming structures, we remembered here the most common:

- the negation (operator ! and function *Not()*)
- the logical and (operator & and function *And()*)
- the logical or (operator / and function *Or()*)
- the functions of comparison like the equal (operator == and function *Eq()*)
- the different (operator != and function *NE()*)
- the minor than (operator < and function *LT()*)
- the minor or equal to (operator <= and function *LE()*)
- the greater than (operator > and function *GT()*).
- *GT GT {Serie GT(Serie S1 [, Serie S2, ...])} {} {Devuelve cierto si cada argumento es mayor que el siguiente.}*
- the greater or equal to (operator >= and function *GE()*).

In TOL it is considered that any *Real* variable whose value is different from zero has the certain logical value (true). Not all the logical values have sense for all the types of TOL variables (to know that logical values are available, they are possible to be visualized in the existing functions for the type of variable at issue).

Example Application of logic functions to real numbers:

Real $x = 0;$

Real $y = 1;$

```

Real z = 2;
Real
logic01 = !x; logic02 = Not(y);
logic03 = x&y; logic04 = And(x,y,0);
logic05 = y|x; logic06 = Or(x,y,0);
logic07 = x<y; logic08 = LT(x,x,y);
logic09 = x<=y; logic10 = LE(x,x,y);
logic11 = x!=y; logic12 = NE(x,z,y,x);
logic13 = x==y; logic14 = Eq(x,x,x);
logic15 = x>y; logic16 = GT(z,y,x);
logic17 = x>=y; logic18 = GE(y,y,x);

```

The results of this example can be consulted through the inspector of objects:

3. Sentences of prog

The programming sentences allow to make a repeated process several times and to make decisions. They are the denominated **sequences of expression**, **sentences of selection** and **sentences of iteration**.

A **sentence of iteration** is used to make a process repeated times, that will be executed while certain conditions are fulfilled. The simplest ones are constructed using the functions *For()* and *EvalSet()*.

The **sentences of selection** allow executing one out of several actions based on the value of a logical or relational expression. They are very important structures since they are the ones in charge to control the flow of execution of a program. They are constructed using the *If()* function and the *Case()* function.

3.1. EvalSet()

EvalSet() is the most useful type of loop in **TOL** language. Its general form is the following one:

```
EvalSet (NombreConjunto, code hacer )
```

Explanation: For the set *NombreConjunto* evaluates element by element the set of sentences *code hacer*. The sentence *Eval()* can be used also to evaluate an expression, polynomials and fractions.

Example Next we display an example using an *EvalSet()* loop. The function is defined *EvalSum01()*, that uses a *EvalSet()* and accumulates the sum in a *sum* variable using the operator *:=* of reassignment.

```

////////////////////////////////////
Real EvalRandGen(Real seed, Real n)
////////////////////////////////////
// PURPOSE : Generates n random numbers using the seed.
// Function implemented using a EvalSet() schema.

```

```
// The numbers are written in a file, this functions always returns
// TRUE.
//
////////////////////////////////////
{
  Real memory:=seed;
  Real randFun(Real k)
  {
    If((k%20)==1, { WriteLn("Step: "+k); TRUE }, { Write("."); TRUE });
    Real (memory := RandGen(memory));
    Text AppendFile("eval.txt", "rnd = " + memory + NL);
    memory
  };
  Set EvalSet(Range(1,n,1),randFun);
  TRUE
};
```

3.2. For()

For() is perhaps the type of loop used more in most of the programming languages. Its general form is the following one:

For (inicial, final, Real (Real n) {iteracción en n })

Explanation: For all whole *n* from the *inicial* value to the *final* value one by one it gives back the set results *interactions* in *n*.

Where the third argument of the *For()* function is a function without name (*Code* type) of *Real* in *Real* which it receives *n* and it gives back the results of the evaluation of the function of *Code* type in *n*.

Example In this example the *For()* cycle appears nested:

```
Set For (0,5, Real (Real n)
{
  WriteLn("Iteration number " + FormatReal(n));
  Set For (0, n, Real (Real m)
  {
    WriteLn(" Subiteration number " + FormatReal(m));
    m
  }
  );
  n
}
);
```

Plan at the **window of messages:**

Iteration number 1

Subiteration number 1

Iteration number 2

Subiteration number 1
Subiteration number 2
Iteration number 3
Subiteration number 1
Subiteration number 2
Subiteration number 3
Iteration number 4
Subiteration number 1
Subiteration number 2
Subiteration number 3
Subiteration number 4
Iteration number 5
Subiteration number 1
Subiteration number 2
Subiteration number 3
Subiteration number 4
Subiteration number 5

3.3. If()

The *If()* function control consists of three operands (ternary) and has the following general form:

If(Expresión, Expresión_1, Expresión_2)

Explanation: *Expression* is evaluated and

- If the result is true it executes *Expresion_1*
- If the result is false it executes *Expresion_2*

Remember also that *Expression* can be simple or composed (block {...}).

Example In this example we present the use of the function of simple *If()* control that returns the second argument if the first is true and in any other case, returns the third:

Real x = 1;

Real y = 0;

Real c1 = If(x>y,x,y);

Real c2 = If(x<y,x,y);

Example

In this example we present the nested use of the control function of *If()*

c3 = If (x > y, If (y != 0.0, x/y, 2) , 3);

In the inspector of objects, we can observe the results of both of the previous examples:

3.4. Case()

The *Case()* control function consists of three or more operands and has the following general form:

Case(Condicion1, Expresion1, [Condicion2, Expresion2,...])

Explanation: *Condicion1* is evaluated and

- If the result is true it executes *Expresion1*
- If the result is false it evaluates *Condicion2*
- If the result is true it executes *Expresion2*
- If the result is false it evaluates *Condicion3...*

It is necessary to remember that *Expression* can be simple or be composed as (block {...}).

Example In this example we present the use of *Case()* control function in order to compare two real numbers:

Real x = 1;

Real y = 0;

Real c1 = Case(GT(x,y),x,EQ(x,y),3,LT(x,y),y);

IV. Set

In TOL it is possible to construct sets (*Set*) like collections of elements of the same or of different types.

1. Definition of sets

Four types of sets exist, simple sets, sets of sets, structured sets and set of structured sets (tables).

1.1. Simple sets

Next, as an example, some definitions of sets can be seen.

Examples

- A set of real numbers, defined by means of the function *SetOfReal()*,
Set RealSet1 = SetOfReal(5, 1.4, .23, 456, 83.85);
- A set of dates, defined by means of the function *SetOfDate()*,
Set DateSet = SetOfDate(y1995m12d3, y1996m11d15, y1994m1d4, y1994m12d04);
- A text set, defined by means of the function *SetOfText()*,
Set TextSet = SetOfText("alfa01", "beta01", "afa02", "beta03", "afa04");
- A set of polynomials, defined by means of function *SetOfPolyn()*,
*Set PolynSet = SetOfPolyn(3*B⁷, 87*B³+32+4*B²);*

- A set of polynomial fractions, defined by means of function *SetOfRatio()*,

$$\text{Set RationSet} = \text{SetOfRatio}((3*B^7)/(87*B^3+32+4*B^2), 1/(1-B));$$
- A set of temporary sets, defined by means of the function *SetOfTimeSet()*,

$$\text{Set WeekSet} = \text{SetOfTimeSet}(\text{Daily}, \text{Weekly});$$
- A set of series, defined by means of the function *SetOfSerie()*,

$$\begin{aligned} \text{Serie S1} &= \text{Pulse}(y2001, \text{Daily}); \\ \text{Serie S2} &= \text{Pulse}(y2002, \text{Daily}); \\ \text{Set SeriesSet} &= \text{SetOfSerie}(S1, S2); \end{aligned}$$

1.2. Sets of sets

TOL also allows to construct a set of sets, using the *SetOfSet()*, function, for example, with some of the previously defined sets,

$$\text{Set SetofSets} = \text{SetOfSet}(\text{RealSet}, \text{WeekSet}, \text{TextSet}, \text{SeriesSet});$$

In the following figure we can observe, through the inspector of objects, the created sets:

1.3. Structured sets

In TOL it is possible to declare structures,

Example

The following command declares a structure of sets formed by a temporary series (*Serie*), a date (*Date*), two real numbers (*Real*) and a set (*Set*).

$$\text{Struct Structure}\{\text{Serie Ser}, \text{Date Dates}, \text{Real Real1}, \text{Real Real2}, \text{Set Sets}\};$$

From this structure we could define a set on it, using

$$\text{Set SetStructure} = \text{Structure}(S1, y2001, 200, 1, \text{TextSet});$$

In the following figure we can observe, through the **objects inspector**, this structure:

1.4. Other definitions of sets

In addition sets can be created by means of other functions, for example

- The *Eva/Set()* function, that allows to generate a set that results to evaluate a set and to apply a series of sentences on
- The *For()* function, that allows to generate a set that results to evaluate a function of real parameter between two given numbers
- The *Range()* function, that returns a set of numbers between two dices following a certain interval

- The *Select()* function, that selects a subgroup of another set with the elements that fulfil certain logical condition.

Examples

Set Rang = Range(1,4,1); // Returns the set of numbers one by one, from 1 to 4.

The following sentence defines a selection function that allows to select the maximum real numbers

Real Selection (Real x){ GT(x, 2)};

By using this function of selection it is possible to extract a set previously defined RealSet those greater than two like

Set Choice = Select(RealSet, Selection);

Example

Let 's define the Double function that returns the double of a number,

*Real Double(Real x){ 2*x };*

The following function returns the set of numbers that are the double of the numbers between -1 and the 2.

Set IntervDouble = For(-1, 2, Double);

In the following figure we can observe, through the objects inspector, the content of this !IntervDouble set:

Cartesian products can be constructed, for example, the expression

Set CartesProdSet = Rang^3;

Generates the result of multiplying the set 'Rang' by itself twice and the expression

Set CartesProdSet2 = CartProd(RealSet, DateSet, TextSet);

Is the cartesian product of the three sets specified as parameters.

In addition, in TOL sets can be ordered given a certain criterion of arrangement by means of 'Sort()' function. The criterion is specified as a function that compares two parameters or arguments and which:

- returns *-1* when the first argument is minor than the second
- returns *0* when both elements are located at the same level according to this criterion and
- returns *1* when the first argument is greater than the second.

Example The following functions allow to order numbers of minor to greater and greater to minor, respectively.

Real Order1(Real x, Real y) { Sign(x-y) };

Real Order2(Real x, Real y) { Sign(y-x) };

Next it is displayed how to order sets of real numbers both by using 'Sort()' and the annotated criteria of arrangement:

Set Increase = Sort(!SetOfReal(6,2,-1,-6,0,3,-2,-3,4,-4,5,-5), Order1);

Set Decrease = Sort(!SetOfReal(6,2,-1,-6,0,3,-2,-3,4,-4,5,-5), Order2);

Another form to make this example by means of functions without name:

2. Operations with sets

Many other operations on sets exist in TOL. We must have special care of the use of the operations, since the sets in TOL have one double function: they are possible to be used as sets or as vectors.

Combined operations when they are used as sets:

- the union of two sets using **operator** +
Set Summ = RealSet + DateSet;
- the difference between two sets using **operator** –
Set Subtract = RealSet - SetOfReal(1.5, .23, 83.85);
- the intersection of sets using the **operator** *
*Set Intersection = RealSet * Range(2,5,1);*
- the extraction of the non repeated elements using the *Unique()* function
Set Elements = Unique(Choice);

Combined operations when they are used as vectors:

- the concatenation of sets using the **operator** <<
Set Appendd = RealSet << SetOfReal(1.5,.23,83.85);

V. TimeSet

It is a type of data with basic dates ($Y(x) = Year$, $M(x) = Month$, $D(x) = Day$, $H(x) = Hour$, $Mi(x) = Minute$, $S(x) = Second$), *basic operations* (*Unión, Intersección y Diferencia*), that allow to generate new temporary sets, like *WD (WeekDay)* and others. TOL language is based on an infinite representation of time. It is a fact that a representation of time has important implications, since:

- it is not only possible to manipulate in TOL temporary series with daily, weekly, decennial, monthly, semi-monthly, annual regularities habitual, etc.
- but it is also possible to describe irregular regularities, for example one or certain days of the week, certain months or certain days of the month, specific combinations of specific days of the week per months of the year, as much loose days of the year

by intervals, periods of Easter, etc. that can be defined in ad-hoc form for each problem in specific.

The time is a continuous variable and in order to work with it is necessary to divide it in intervals. This partition of time in intervals is known in **Bayes** with the name **date**.

With the *TimeSet* type we find two types of objects, the temporary sets (finite) on the one hand and the dated ones (infinite) on the other hand.

In order to create a temporary series in TOL one is due to define on date. The necessity to have dates occurs given the study:

- Each how long the same basic conditions for the occurrence of the data are repeated. Example: for the daily series Sales: every Monday we have the same conditions, space (in days) between Monday and the following one: 7 days that correspond with the period of the series
- If the collection is daily we will apply the Daily, if it is every week the Weekly, the etc. but if we have a particular case of irregular collection of data and want to create an ARIMA model that gathers the behaviour of those sales we have to create particular date, to count on an essential requirement of this class of models that is the period.

When initiating TOL we counted on a series of predefined date, like Daily, Weekly, Monthly, Easter (Sunday of Easter), etc. In order to see that dates are already defined we go to Grammars - TimeSet as it shows the figure:

For example, the temporary set of all Mondays (*Semanal*, $WD(1)$) is date, but the temporary set of Mondays of year 2000 ($WD(1)*Y(2000)$) is not dated.

We can characterize dated by the following way:

- For a date pertinent to a dated one we can calculate its successor within that dated, something that does not happen in the temporary set
 - The dated one is a temporary infinite set.
- Within the algebra of the time we distinguish between: the temporary sets and the operators to treat them. Within the temporary sets it is possible to distinguish between:

- Primitive elements, predefined according to the daily representation.

Example

// Set of all days on the year that belongs to January
TimeSet January = M(1);

- Temporary sets derived from primitive elements and the application of operators on them.

Example

// Set of all days on the year that belongs to January and February
TimeSet JanuaryFebruary=*M(1)+M(2)*;

- And with respect to the operators and functions:
- Operators: union (+), difference (-) and intersection (*).
- Functions: the successive function (*Succ()*) and its derivatives (*Range()*, *Periodic()*).

1. Daily representation

We define the sets C and W that represent the universal set (therefore, all the days) and the dummy set respectively. The following families of temporary sets are also consider:

- *D(i)*, $i=1\dots, 31$, where *D(i)* represents the set of every i-th day of any month.

Example: *D(i)* is the temporary set of every day One of every Month

- *WD(i)*, $i=1\dots, 7$, where *WD(i)* is the set of days that belong to the i-th day of the week.

Example: *WD(1)* is the temporary set of all Mondays

- *M(i)*, $i=1\dots, 12$, where *M(i)* is the set of days that belong to the i-th month of the year.

Example: *M(1)* is the temporary set of all days of January

- *Y(i)*, that represents the i-th year in the present Gregorian calendar.

Example: *Y(2004)* is the temporary set of all days of year 2004.

- The **closed intervals**, that contain all the days between two extremes. When the left end is TheBegin indicates lack of landmark by the left, whereas when the right end is TheEnd indicates lack of landmark by the right. Thus, $In(TheBegin, TheEnd) = C$ and if the inferior end is greater than the superior end the resulting set is W.

Example:

// Set of all days on the year between Day 1 of January 1999 and 15 of
 // January 1999
!TimeSet Fifteenth99 = *In(y1999m01d01,y1999m01d15)*;

Within this representation it is possible to use smaller units of time, the minimum unit of time represented in TOL is the second, and also there are the minute and the hour whose representation is:

- $S(i)$, $i=0\dots, 59$, where $S(i)$ represents the temporary set of all the dates in the i -th second.

Example: $S(0)$ is the temporary set of all the first seconds of a minute

- $H(i)$, $i=0\dots, 59$, where $H(i)$ represents the temporary set of all the dates in the i -th minute.

Example: $M(0)$ is the temporary set of all the first minutes of the hour

- $H(i)$, $i=0\dots, 23$, where $H(i)$ represents the temporary set of all the dates in the i -th hour.

Example: $H(0)$ is the temporary set of the first hour of the day.

2. Algebraic operations

The basic algebraic operations for the construction of new temporary sets are:

- **Union (operator +):** If A and B they are two temporary sets, then $A + B$ is defined as: $\{d/d \text{ pertains to A or d pertains to B}\}$.

Example

TimeSet MonTues = WD(1) + WD(2); // TimeSet of Mondays and Tuesdays.

- **Intersection (operator *):** If A and B are temporary sets, then $A * B$ is defined as: $\{d/d \text{ pertains to A and d pertains to B}\}$.

Example

*TimeSet FirstFeb = D(1)*M(2); // TimeSet of first day of February.*

- **Difference (operator -):** If A and B are temporary sets, then $A - B$ is defined as: $\{d/d \text{ pertains to A and d does not pertain to B}\}$.

Example

TimeSet WeekWitOutSund = C - WD(7); // TimeSet of all week days except of Sundays

3. Functions

In TOL there are functions predefined temporarily. The temporary sets constitute an algebra and they can obtain new sets through the displacement operators.

3.1. Successor

We define the $\text{Succ}(Ct, n, CtUnidades)$ like the obtained set calculating the n th successor to each element of Ct in the given units.

Example *TimeSet SeconDayMonth = Succ(D(1),1,Daily); // TimeSet of second day of each month = D(1) + 1 el primer día seria D(1), 0 .*

3.2. Range

We define the function *Range(Ct, n, m, CtUnidades)* like:

Example // TimeSet of first, second, third and fourth day of each month.
TimeSet FirstToFourthDayMonth = Range(D(1),0,3 (es mas 1),Daily);

3.3. Periodic intervals

We define the *Periodic(fecha, n, CtUnidades)* like the set made up of a date and its displacements to a number of dates multiple of n within the dated given data.

Example

//TimeSet of the ten successors of the first day of January 1999.
!TimeSet TenDaysAfter = Periodic(y1999m01d01, 10,Daily);

Example

The following example shows the temporary set of the national celebrations and when some of these celebrations falls on a Sunday, it is transferred to Monday. The following ones are included:

TimeSet NatHol=

{

*NewYear = D(1) * M(1),*

SaintFriday = Range(Easter,-2,-2),

*WorkDay = D(1) * M(5),*

AugustVirgin = D(15) M(8),*

PilarDay = D(12) M(10),*

*AllSaints = D(1) * M(11),*

*Constitution = D(6) * M(12),*

*Immaculate = D(8) * M(12),*

Christmas = D(25) M(12),*

*NewYear + SaintFriday + WorkDay + AugustVirgin + PilarDay +
 AllSaints + Constitution + Immaculate + Christmas*

};

TimeSet NatHolNoSun = NatHol - WD(7);

*TimeSet NatHolTrans = Succ(NatHol * WD(7), 1, Diario);*

TimeSet CtNatHol = NatHolNoSun + NatHolTrans;

The result of this example for the year 2004 is shown in the following figure:

Example Temporary sets from Monday through Friday and Workable.

TimeSet CtMonFri = C - WD(6) - WD(7); // TimeSet Mondays to Fridays

TimeSet CtLabour = CtMonFri - CtNatHol; // TimeSet labour days

Example Months that contain 5 Sundays. In this case we indicate days One of these months:

*TimeSet CtFifthSunMonth = (D(29)+D(30)+D(31)) * WD(7);*

TimeSet CtFirstDayMonth5Sun = Succ(CtFifthSunMonth, -1, D(1));

The result of this example for year 2004 is in the following figure:

In order to obtain the temporary set corresponding to all the days of the month, of the months of 5 Sundays, we could define the following sets:

TimeSet CtFirstDayMonth5Sun = Succ(CtFifthSunMonth, 1, D(1));

TimeSet CtLastDayMonth5Sun = Succ(CtFirstDayMonth5Sun, -1, C);

TimeSet CtMonth5SunCom = Range(CtFirstDayMonth5Sun, 0, 30)-

(Succ(CtLastDayMonth5Sun, 1, C) + Succ(CtLastDayMonth5Sun, 2, C));

The result of this example for year 2004 is in the following figure:

VI. Series

TOL allows the treatment and the algebraic manipulation of temporary series, to conduct operations, graphs, estimations and forecast of data. It must be considered that the statistics available in TOL for temporary series require the specification of the moment of time to which each specific value talks about. That moment can come represented by a TimeSet variable.

1. Operations, functions and polynomials

TOL has all basic arithmetical functions like the sum, subtraction, multiplication, division, concatenations of series, statistical functions (*AvrS()*, *StDsS()*, *MaxS()*), generation of random series (*Rand()* and *Gaussian()*); trigonometric and logarithmic functions, logical functions and of comparison, another type of transformations (*Exp()*, *Pow()*, *Sqrt()*,...). All of them, with their description and arguments, can be located by the following way:

The following table summarizes some of the most excellent statistical functions of temporary series in the space of the real numbers:

1.1. Operations between series

In order to see the operation of some of the functions, we are going to construct a pair of series randomly:

Serie s1 = Gaussian(1,0.1,Daily);

Serie s2 = Gaussian(-1,0.5,Daily);

Observations:

- In order to conduct operations between two series these must be dated in the same way but they do not have because to be defined between the same dates. The operations will be carried out in the single dates that they have in common. The series generated by means of *Gaussian()* and *Rand()* will be defined between the dates that **TOL** has for defect. These can modify from **Tools Options** and selecting the dates that interest us puncturing in the calendars that appear next to **Initial Date** and **Final Date**, as it is next
- The defined series as it finishes describing are not calculated until they are not delimited between two dates. For it, we used the *SubSer()* command, to limit it between two dates.

Example

```

Serie s11 = SubSer(s1,y2003m01,y2003m12); // We bound the first serie on year
2003
Serie s21 = SubSer(s2,y2003m01,y2003m10); // We bound the second one between

// January 2003 till October 2003.

Serie add = s11 + s21;
Serie dif = s11 - s21;
Serie prod = s11 * s21;

```

The following table shows that the operations are made piece by piece and only between those dates, in which both series are defined, is to say:

- The series add begins in maximum {January 2003, January 2003} and finalizes in the minimum {December 2003, October 2003}. Analogous it happens to the series dif and prod.

Between some of the possible functions that can be used between series they are also some to emphasize:

Sum of a series of series:

SummSeries

Serie SummSeries = Sum(s1,s2);

Product of a series of series:

Serie ProdSeries = Prod(s1,s2,s1+s2);

Power of series by a number:

*Serie PowerSeries = s1**3;*

Power of series by series

*Serie PowerSerSer = s1**s2;*

*Serie PowerSerSer2 = s1^{s2}; // ** is equivalent to*

Serie PowerSerSer3 = Pow(s1,s2); // Pow is equivalent to previous ones

Maximum of a series

Serie MaxSerie = Max(s1,s2);

Minimum of a series

Serie MinSerie = Min(s1,s2);

Rounding

Serie Rounded = Round(s1);

Truncation

Serie Trunc = Floor(s2);

1.2. Series Functions

TOL provides functions of change of dated form, statistical concatenation of series, operations, trigonometric transformations, etc.

The functions of series contain in their arguments dates or temporary sets in addition to series and the result of applying a function to them is a new series. We will define the following functions:

SubSer(Serie, Date, Date) allows to restrict the acquaintance data of a series to the known data included in the series, between two given dates. The first argument is the series that is desired to restrict, the second and third arguments are the initial dates and the final of the resulting series.

Example

// We define two infinite series:

Serie StepInf = Step(y2005m05,Monthly);

Serie PulseInf = Step(y2005m08,Monthly);

// We bound the first serie between January 2004 to September 2005

Serie StepMay = SubSer(StepInf,y2004m01,y2005m09);

// We bound the second serie between January 2004 to december 2005

Serie PulseAugust = SubSer(PulseInf,y2005m01,y2005m12);

Concat(Serie, Serie, Date) concatenates the two series of the arguments with respect to the given date. The resulting series takes the values from the first series of the arguments if their dates are smaller or equal than the given date, and the values of the second series from the rest of the dates.

Example

// We append both series from June 2005

Serie Appendd = Concat(StepMay,PulseAugust,y2005m06);

<<(Serie, Serie) gives back a series whose values are values of the second series by every greater date or just as the last date of the dated one of the first series, or values of the first series in opposite case.

Example

// We append both series:

Serie Appendd2 = StepMay<<PulseAugust;

>>(Serie, Serie) gives back a series whose values are values of the first series by every smaller date or just as the first date of the dated one of the second series, or values of the second series in opposite case.

Example

// We append both series:

Serie Appendd3 = StepMay>>PulseAugust;

DatCh(Serie, TimeSet, Code) give back a series with the indicated dated one. The transformation of the original values is indicated either by a statistical one or by a set of sentences. It has as a restriction that the new dated form must be harmonic with the original dated form of the series. This means that every dated date of the new one must contain a whole number of date of the old one, this number at least does not have to be constant.

Example

S is a daily dated series, *DatCh(S, Semanal, AvrS)* gives back a weekly dated series whose values are the average of the values of every week.

1.3. Polynomials

TOL allows temporary displacements through the **Backward** operator *B* of retardation, and of the **Forward** operator *F* of progress. The operator *:* allows to apply polynomials to temporary series returning a series that comes determined by means of the polynomial and the original series.

- The **operator** *B* of retardation (**Backward**) on a temporary series is a new displaced temporary series backwards in time one temporary unit

Example: $[B: z(t)] = z(t-1).$

- The **operator** *F* of advance (**Forward**) on a temporary series is a new displaced temporary series forwards in time one temporary unit

Example: $[F: z(t)] = z(t+1).$

These two operators handle themselves like polynomials, so many times are applied as it is required and recognise operations with real numbers.

We can apply these operators recursively (B^m is possible to be applied in order to construct to a new $z'(t)=z(t-m)$ or $z'(t)=z(t+m)$, where m being any whole number). In addition, they recognise operations with real ($z'(t)=(0.3*B):z(t)$). The most well known applications of these operators are the moving averages, series of increases, etc. The following example illustrates these concepts:

Example

```
// We generate a random time serie with normal distribution with
// average 1 and standard deviation 0.1
Serie alfa = SubSer(Gaussian(1,0.1,Monthly),y2005m01,y2005m12);
// We apply the backwards operator
Serie AlfaBackwards = (B):alfa;
// We translate the serie two units on time
Serie AlfaFoward = (F**2):alfa;
// We apply different foward / backward operators
Serie AlfaPolin1 = ((1-B)*(1+B)):alfa;
Serie AlfaPolin2 = (1+2*B+3*B**2):alfa;
Serie AlfaPolin3 = (1+2*B+3*F):alfa;
Serie AlfaPolin4 = (1+F+2*B**2):alfa;
Serie AlfaPolin5 = ((1+2.5*B)+(F2+3.5*F4)):alfa;
// We construct a serie stepwise like through an step variable:
Polyn Stepwise = F+B2+F3+B4+F5+B6+F7;
Serie Step Stepwise = Stepwise:Step(y2005m3d01,Monthly);
```

2. Deterministic series in TOL

A deterministic series, also denominated as an artificial series, is an infinite list of known values, which represents temporary contents or qualitative events. The determinist series are used in the intervention analysis to represent precise behaviours, sporadic facts or simply calendars effects in the observed series. The most common determinist series are:

- The pulse that takes value 1 for a date and value zero in all the others. It has as an objective to determine the measurement of the influence that a specific event has had on the object variable of the analysis, when it is expected that that influence is pronounced of precise form. A pulse is generated with the function *Pulse(Date, TimeSet)*.

Example

```
Serie PulseG = Pulse(y2000, Monthly);
```

- The compensation that takes values 1 and -1 in correlative dates and value zero in all the others. A compensation is the difference between two consecutive pulses, which provides a measurement of the influence of the represented event when the initial effect can be seen total or partially rectified by an effect of opposite sign in immediately later date. A compensation is generated with the function *Compens(Date, TimeSet)*.

Example

```
Serie CompensG = Compens(y2000, Monthly);
```

- The step that takes value 1 as of a capable date, and value zero in all the precedents. A step will be the representation of a constant event on the object

variable of analysis as of a moment of time. A step is generated with the function *Step(Date, TimeSet)*.

Example

Serie StepG = Step(y2000, Monthly);

- The trend with value zero until a capable dates in which it takes value 1, is growing arithmetically with ratio 1 in posterior dates. A trend is generated with the function *Trend(Date, TimeSet)*.

Example

Serie TrendG = Trend(y2000, Monthly);

Another group of artificial variables is the calendar variables. They have as an objective to represent phenomena related to the almanac and labor agendas that exert a systematic influence on the variable object of analysis. The calendar variables are generated by means of the function of series *CalVar(TimeSet, TimeSet)*, and the function '*CalInd(...,...)*' among others.

Let's see an example with declaration in TOL: *[[BR]] Serie IndicatorG = CalInd(Y(2000), Monthly);*

And in order to see them in a graph we limit them between two dates:

Serie pulse = SubSer(8 + Pulse(y2005, Monthly), y2004m01, y2006m01);
Serie compensate = SubSer(6 + Compens(y2005, Monthly), y2004m01, y2006m01);
Serie step = SubSer(4 + Step(y2005, Monthly), y2004m01, y2006m01);
Serie trend = SubSer(2 + Trend(y2005, Monthly), y2004m01, y2006m01);
Serie indicator = CalInd(M(1) + M(2) + M(3), Monthly);

Observations:

- When differentiating a pulse we obtain a compensation:

Serie PulseDif = (1-B):PulseG;

- When differentiating a step we obtain a pulse:

Serie StepDif = (1-B):StepG;

- When differentiating a trend we obtain a step:

Serie TrendDif = (1-B):TrendG;

- Cumulative tendency with the *DifEq* function:

Serie Trend02 = SubSer(DifEq(1/(1-B), TrendG), y2004m01, y2006m01);

3. Introduction in modelling

In this section, we will briefly introduce how to model a temporary series with TOL and how to use the models that better adjust the forecast of data. In a modelling process, the following steps are due to follow:

1. Analysis and exposition of the problem.
2. Taking of data.

3. Identification phase.
4. Estimation of the model.
5. Validation.
6. Forecast.

3.1. Reading data

As we said previously TOL stores the data in files with format BDT that is the basic format that will be used for the storage of temporary series. It is based on a flat file ASCII, which consists of a column with the dated form (daily, monthly, annual...) and the remaining columns correspond to values of series in relative dated form. The different columns are separated from each other with semicolon.

Let's remember that in order to read the file we used the *IncludeBDT()* function or other similar (*Include()*, *IncludeBDC()*), whose description and parameters we can consult in the following way:

It is important to indicate that for the reading of the file it is necessary to detail the route of directories where the same one is.

```
Set Table = IncludeBDT("C:\Bayes\Ejemplos_Manual\Cap06_Series\CarReg.bdt");
```

We are going to work with the CarReg file that contains data on the matriculation of automobiles. One is a monthly series from January of 1983 to November of 2004.

Most of the temporary series are generated by non-stationary models where transformations being necessary to turn them to stationary processes, in order to use the advantages that stationarity offers for their modelling. In a first stage of the elaboration of an ARIMA model they fundamentally require like basic instruments of identification the function of simple and partial autocorrelation considered. Once identified the model values considered for the parameters are obtained, and the phase of validation is carried out that goes directed to establish if it takes place or not that adjustment between data and model. Finally, in the phase of prediction, prognoses in probabilísticos terms of future values of the variable are made.

3.2. Identification phase. Visualisation

Once including the file, it appears in console Objects under the name that we have given the set, in our case **Table**. Within the set we select the series that we want to visualize and by pressing the right button of the mouse they appear among other options the ones of creating graphs and tables of the series and the functions of autocorrelation, as well as the one to see some of its most important statistics.

Next we show the graph of the series, within the graphical interface we also have options that allow changes of category, change of status, selection of parts of the series, change of the number of marks, etc.

In the following graph we show the functions of simple and partial autocorrelation of the series.

By observing the graphs of FAS and FAP of the series it is decided to apply a regular difference of order 1, an annual seasonal difference and to establish the model that seems more appropriate to us. In order to make this type of transformations certain types of polynomials must be applied to the series.

Serie s1 = B:CarReg; // s1 represents the original time serie

//with a backwards.

Serie s2 = (1-B^12):CarReg; // s2 represents the original time serie

// transformed by a seasonal regular

// differentiation.

Serie a1 = (1-B):CarReg; // a1 represents the original time serie

// transformed by a regular order 1

// differentiation.

Serie a2 = (1-B^12):a1; // a2 represents the serie that is attained

// dafter applying a regular order 1 seasonal

// differentiation.

Another form to define these transformations, instead of doing it sequentially, would be to use products of polynomials in TOL:

Polyn diff = (1-B)(1-B^12);*

Serie s3 = diff:CarReg;

Next in the following figure we show the series s3 free from seasonality and trend. It is a stationary process where a series of atypical data is observed that would be necessary to take part through artificial variables.

We define next the following variables that we are going to introduce in our model and whose objective will be to obtain all these effects.

Serie Pulse9607 = Pulse(y1996m07,Monthly); Serie Pulseo9301 = Pulse(y1993m01, Monthly); Serie Pulse8601 = Pulse(y1986m01, Monthly);

In the differentiated series, seasonal and regular, we observed again its FAS and FAP to decide the possible models to consider.

After observing these graphs we propose to fit to the series a SARIMA(2,1,0)x(0,1,1)₁₂ process.

3.3. Estimation Phase

After identifying the model that follows the series, we come to estimate the parameters. For this, TOL has the function

Set Estimate(Set modelDef [, Date from, Date to], Set parametersAPriori)

This function requires:

- Like the first argument a set of the following type

Set ModelDef(Serie Output, Real FstTransfor, Real SndTransfor, Real Period, Real Constant, Polyn Dif, Set AR, Set MA, Set Input, Set NonLinInput)

where the parameters represent:

Serie Output is the series that we try to fit.

Real FstTransfor is a real number that represents the first Box-Cox transformation.

Real SndTransfor is a real number that represents the second Box-Cox transformation.

Real Period represents the estacionalidad that presents/displays the series.

Real Constant is used if we want to fit the model with a constant.

Polyn Dif will be the product of the polynomials regular difference and the seasonal difference that we needed to apply to the series to turn it to stationary.

Set AR is a set of polynomials with two elements. The first it is polynomial AR of the regular part and the second is the polynomial AR of the seasonal part.

Set MA is a set of polynomials with two elements. First it is polynomial MA of the regular part and the second is the polynomial MA of the seasonal part.

Set Input is the set of explicative variables (for example, interventions).

- It is important to clarify that this set is constructed by using the function:

Set InputDef(Polyn Omega, Serie X) where Omega is the polynomial that is applied as input that we are going to introduce in the model (that in general is going to be a constant), whose coefficients must be estimated with the model. *Set InputNonLin* represents the set of nonlinear inputs.

- As second and third arguments (which are optional) come the initial dates and the final ones that we want to take under consideration to estimate (in general all the data available are used).

- As a last argument (also optional) *Set parametersAPriori* is used.

For our case:

Set Model = ModelDef(CarReg, // output

1, // First transformation: exponent

1, // Second transformation: translation

12, // Period

0, // Constant

diff, // Regular and seasonal difference

SetOfPolyn(1-0.1*B-0.1*B^{2,1}), // AR[[BR]] SetOfPolyn(1,1-0.1*B¹²), // MA

```
SetOfSet(InputDef(0.1, Pulse8601),  
  
InputDef(0.1, Pulse9607),  
InputDef(0.1, Pulse9301)), // Input set  
  
Copy(Empty) ); // Non linear input set
```

Once compiled we obtain the following output:

4. Validation Phase

The objective when elaborating a SARIMA model is to find a model that must be the most suitable possible to represent the behaviour of the series examined. Thus, an ideal model would be the one that would fulfil the following requirements:

- The residuals of the estimated model approximate the behaviour of a white noise.
- The estimated model is stationary and invertible.
- The coefficients are statistically significant, and are a little correlated between them.
- The coefficients of the model are sufficient to represent the series.
- The level of adjustment is elevated in comparison to the one of other alternative models.

The purpose of the validation phase consists indeed of analyzing the adjustment between the model and the data. For this we build an analysis of the residuals that consists of verifying if the residuals behave like a white noise.

It is important to clarify some of the different series that the output of this program shows:

CarRegRes treats the residuals of the series obtained like they turn out after estimating the original series with all its effects.

CarRegNoise treats the original series but without the effects of the artificial variables.

CarRegDifNoise treats the series *CarRegNoise* differentiated in regular form.

For the validation phase we use the residuals series, *CarRegRes*. If we build its FAS and FAP we will observe that they correspond to those of a white noise.

5. Prediction Phase

The three first stages of the elaboration of a model ARIMA constitutes an iterative process whose final result is the obtaining of an estimated model that is compatible with the structure of the data. Once this result is obtained, the following phase consists of using this estimated model in the prediction of future values of the variable examined. For example, we want to predict the following year at the end of the data of the series. For it, we implement the function

```
CalcForecasting(model, Initial Series, Final Series, Number of forecasts, alfa); where
alfa is the adjustment of the confidence limits. DateIniSer = First(CarReg); // We
define the date where the time series starts Date FinSer = Last(CarReg); // We
define the date where the time series ends Set ModeloPrev =
CalcForecasting(ModeloEstim,IniSer,FinSer, 12,0.05);
```

Let's see a graph with the forecasts of the next year with the data of the two last years and the confidence levels of 95%:

It is important also to emphasize that before the appearance of the Box-Jenkins methodology, the classic analysis of temporary series was based on decomposition methods. The movements that present the evolution of a temporary series, are considered as the result of the composition of four effects or denominated forces: trend, cycle, seasonality and irregular component. The four forces or components are no observable and it assumes that its integration determines the evolution of the temporary series. After introducing this, it is possible to indicate that in the evaluation of trend components the so called **methods of average moving bodies** were used among others. These methods consist of dividing equally each value of the series with some of the observations that precede this value and some that they follow this value. Of this form some accidental variations are eliminated and the resulting series is identified with the trend component. An example of how to introduce this function in TOL is:

We create a useful function that gives back the average moving part of order n. *Serie*
MovAverN(Serie s, Real n)

```
{
```

```
(Quotient((1-B^n)/(1-B)):s)/n
```

```
};
```

```
PutDescription("Returns the order n moving average of a given serie",MedMovN);
```

// We apply the order 3, 4 and 12 average moving function

```
Serie MedMov3 = MovAverN(CarReg,3);
Serie MedMov4 = MovAverN(CarReg,4);
Serie MedMov12 = MovAverN(CarReg,12);
```

Let's see the result of the approach of the series *MedMov3*, *MedMov4* and *MedMov12* for the *CarReg* series:

- *MovAverBack6()* is a function user defined that generates a moving average part with the present value and the values of 6 periods, its definition and use is the following one:

```
Polyn PolyBack6 = 1+B+B^2+B^3+B^4+B^5+B^6; // Definition of Polinomio
Serie MovAverBack6(Serie Ser){ // Definition of the función
```

```
(PolyBack6:Ser)/7
};
```

```
Serie CarRegMediaMovil = MovAverBack6(CarReg); // Apply the av. moving part
```

Let's see the result of the approximation of the series *CarRegMediaMovil* by the series *CarReg*.

VII. Other types of variables

In the preceding chapters the more important types of data of language TOL will be described. In this chapter, we will detail some of the operations and more frequent functions for variables of type *Polyn*, *Ratio* and *Text*.

1. Funciones y operaciones con:Polyn

We have previously described the polynomials of retardation and development. We will detail next the polynomials Delta and unknown:

- The operating *Delta* of regular difference on a temporary series is a new temporary series that represents the increase by a unit of time of the original series.

Expression:

- The operator *-?* unknown on a temporary series is a new temporary series with omitted values.

These operators are applied so many times as one requires and recognise operations with real numbers.

Between the most important functions of manipulation of polynomials, they are:

- The function $+$ (sum), $*$ (product), $-$ (subtraction), $/$ (division) and $^$ (power) between different polynomials.

Example

// We consider the following two polynomials:

// a) The delta polynomial

Polyn Increase = Delta;

// b) The polynomial that creates a new serie like the original one

// minus the original one five times translated on time

Polyn Polinom1 = (1-B⁵);
 // The polynomial summ:
 Polyn summ = Increase + Polinom1;
 // The polynomial product:
 Polyn product = Increase * Polinom1;
 // The polynomial substrate:
 Polyn substract = Increase - Polinom1;
 // The polynomial increase over six:
 Polyn division = Increase / 6;
 // The polynomial increase to the power of three:
 Polyn Power3 = Increase³;

- The function *Derivate* (derived from a polynomial), *Expand* (expansion of degree n of a reason of polynomials), *Integrate* (integral of a polynomial), *SetProd* (product of all the polynomials of a set), *SetSum* (sum of all the polynomials of a set), *Sub* (polynomial formed by the monomials of a polynomial of degrees between the given limits).

Example: We construct different polynomials

// We consider the two following polynomials:

Polyn TrimBackward = 1-B³6-B⁹12;

Polyn AnualBackward = 1-B¹²;

// We construct the derivate of the first polynomial:

Polyn TrimDerivate = Derivate(TrimBackward);

// We construct the expansion of the ratio of both polynomials to degree 4:

Polyn Expantion = Expand (Ratio TrimBackward / AnualBackward, 4);

// We construct the integrate of the derivate of the first polynomial:

Polyn TrimIntegrateDerivate = Integrate (TrimDerivate);

// We construct a set that contains both polynomials:

Set Polynomials = (TrimBackward, AnualBackward);

// We construct the polinomial product of the elements of the previous set:

Polyn Prod Polynomials = SetProd(Polynomials);

// We construct the polinomial sum of the elements of the previous set:

Polyn SumPolynomials s = SetSum(Polynomials);

// We construct the polinomial composed by the monodes of degree less than 6 of TrimBackward.

Polyn MonoLessSix = Sub(TrimBackward,1,6);

2. Functions and operations with: Ratio

These are rational fractions of polynomials. The main utility is to solve equations in differences of the type $P(B)Z_t = Q(B)A_t$. The basic operators are:

- The operator *CompensOut* on a temporary series, is a new temporary series that represents the difference between the present value and the immediately previous value of the original series.

Expression:

- The operator *IDelta* (inverse of the operator delta) on a temporary series is the inverse operator of the previous one. The *StepOut* operator on a temporary series, is equivalent to the *IDelta* operator.

Expression:

- The operator *TrendOut()* on a temporary series, is a new temporary series that represents the original series between the result of the present value less twice the value of the immediately previous original series plus the previous value to the previous one.

Expression:

- The operator *Unknown* on a temporary series is a new temporary series with omitted values. In order to apply these operators to a temporary series, *DifEq* operator is used.

Example

We apply the previous operators to the temporary series *!arReg* that is located in *C:\Bayes\Ejemplos_Manual\Cap07_Otros*. In order to add the file, write in the Edit window:

```
// We include the time serie:
Set SerieData=Include("C:\Bayes\Ejemplos_Manual\Cap07_Otros\CarReg.csv");
// We apply the operator CompensOut to the time serie CarReg:[[BR]] Serie CarReg1
= DifEq(CompensOut,CarReg);
// We apply the operator IDelta to the time serie CarReg:[[BR]] Serie CarReg2 =
DifEq(IDelta,CarReg);
// We apply the operator StepOut to the time serie CarReg:[[BR]] Serie CarReg3 =
DifEq(StepOut,CarReg);
// We apply the operator TrendOut to the time serie CarReg:[[BR]] Serie CarReg4 =
DifEq(TrendOut,CarReg);
```

Between the most important functions of manipulation of ratios, they are: Function $+$ (sum), $*$ (product), $-$ (it reduces), $/$ (division of polynomials), $./$ (division of ratios) and $^$ (power) between different fractions of polynomials.

Example

```
// We consider the following two ratios:
// a) Ratio CompensOut:[[BR]] Ratio compens = CompensOut;
// b) The ratio between the identity polynom and the polynom original serie
```

```
// minus the original serie five times translated in time backwards.  
Ratio fraction1 = 1/(1-B^5);  
// We construct the sum ration:  
Ratio sum = compens + fraction1;  
// We construct the product ration:  
Ratio product = compens * fraction1;  
// We construct the subtract ration:  
Ratio subtract = compens - fraction1;
```

3. Functions and operations with: Text

An object of type text can contain any chain of characters ASCII. The form to create a variable of type text is to put the text in inverted commas. TOL allows diverse treatments of the text variables (search of chains, transformations, etc.).

Among the basic operators of TOL for texts it is possible to emphasize the inclusion of format HTML for the generation of information.

We will describe solely basic functions of handling of texts, conversion of any type of objects to objects of text type

- The function `FormatReal` is an example of conversion of objects of real type to text type.

Example

```
Text FormatReal(3);
```

- The function `GetFileExstension` is an example of conversion of capture of extensions (ways, names of files, etc. can be captured).

Example

```
Text GetFileExtension("C:\Bayes\Ejemplos_Manual\Cap07_Otros\CarReg.cvs");
```

- The `Grammar` function returns the type of a variable.

Example

```
Text Grammar(3);
```

- The `ListOfDates` function returns the set of dates between two dates given in the indicated dated form.

Example

```
Text ListOfDates(Monthly,y2002,y2003);
```

- The `ToLower` function is an example of change to very small letters of a text.

Example

```
Text ToLower("Hello world");
```

VIII. General recommendations in programming

As much in this as in any other language a common form is necessary to program by multitude of motives

- The reutilisation of code on the part of other people
- To take the same reading of the easiest and most transparent
- Greater accessibility at the time of retaking the code after being programmed.

In addition the characterization to common forms to all the codes produces an enrichment of the same one.

1. Organization of a TOL archive

One can find several parts within an archive *.tol*:

- **Head:** It is the part in which reference to the file becomes *.tol* and in that one the content and objective is transformed.
- **File:** Name of the file in small letters the possible most significant and including the extension
- **Intention:** One declares the content and objective of the file. Is written everything in small letters respecting the score and the capital letters but without the use of accents
- **Body:** The different parts of the body are separated through two lines blankly. The labels of the headed ones of each part will go in separated capital letters and by inclined lines (slash). The sub labels like the previous ones must be separated by lines of slash and will go with the first letter in capital letters and the rest in small letters. They will make reference to sets of objects that have a same function or characteristic. The titles of the labels can go in English or Spanish indifferently but always in the same language.
- **Constants:** These must be declared with the first letter in capital letter. All the constants must describe through PutDescription leaving a line in target of separation among them.

To be simple its use in the rest of the code is advisable in the case of having great amount of constants to pack the same ones, like to use a mark (in development tag) on these to identify them, usually the three initial letters. For example: In the case of having constants of control we can have several types namely, of plan, error, warning, etc. With which the package could be CtrTace, CtrError, CtrWarn, etc.

- **Structures:** They are declared between keys or parenthesis, that is to say, {or () or}.
- **Inclusions:** Inclusion of archives
- **Functions:** They can be in the same file or in other files that are included in the part of inclusions. Its name is bound to tag. They are specified more ahead
- **Procedures:** They are code segments that are not functional and that include forms to do of other types of functions. They are specified more ahead. Evidently not all *.tol* files have necessarily all the parts of the body, it depends on the use one is going to give to the file.

```
//////////////////////////////////////////////////////////////////
// FILE : nombredelarchivo.tol
// PURPOSE : Descripcion del archivo.
//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
// CONSTANTES
//////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////
// Constantes de tipo 1
//////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////
// Constantes de tipo 2
//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
// ESTRUCTURAS
//////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////
// Estructuras de tipo 1
//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
// Estructuras de tipo 2
//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
// INCLUSIONES
//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
// Inclusión de archivos de tipo 1
//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
```

```
// Inclusión de archivos de tipo 2
////////////////////////////////////
////////////////////////////////////
// FUNCIONES
////////////////////////////////////
////////////////////////////////////
// Función 1: Descripción del procedimiento.
////////////////////////////////////

////////////////////////////////////
// Función 2: Descripción del procedimiento.
////////////////////////////////////
```

2. Structure of a function

2.1. Header

It is declared as the function between two lines of slash /. Next entering two spaces the data type of return of the function is written, later leaving a space to the name of the function with the first letter in capital letter and finally without intermediate space the arguments of the function in small letter are declared, writing the data type of each one and leaving a space behind the commas.

- **Grammar:** Type of data of the return of the function
- **Name:** The name of the function must be the shortest possible and using in the first place tag corresponding to the group of functions that it belongs (if it belongs to some). Next the structure infinitive verb in more object will be used to designate to the assignment the function, is to say: a function that turns any thing to text it we could be written with the name *CnvAny2Txt*.
- **Arguments:** The names of the arguments must be the most possible explicit, that is to say, if an argument is of Set type and is a set of series the name of this argument would have to be SetSer.

2.2. Body

It declares the code delimited between keys and indentado two spaces. The norms on definition of arguments within the function are identical to the previously mentioned. For clauses TOL like *EvalSet*, *For*, *While*, *Select*, *Sort*, etc. the structure is exactly equal to which it is described previously.

2.3. Special case: Clause IF

The indentation of code when using the If clause appears of different ways based on the complexity of the sentences:

Example:

```

/////////////////////////////////////////////////////////////////
// Example 1.
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
Text TxtParity(Real r)
/////////////////////////////////////////////////////////////////
{ If(EQ(r%2,0), "Is even", "Is odd ") };
/////////////////////////////////////////////////////////////////
PutDescription("Returns the parity of a given number",TxtParity);
/////////////////////////////////////////////////////////////////

```

Example:

```

/////////////////////////////////////////////////////////////////
// Example 2.
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
Set IncludeExtTree(Text pathDir, Text extension)
/////////////////////////////////////////////////////////////////
{

```

```

Set RecIncludeExtTree(Text pathDir)
{

```

```

Set dir = GetDir(pathDir);
Set setNameFiles = dir[1];
Set setPathFiles = If(EQ(Card(setNameFiles),0), Empty,
{

```

```

Set dirExtFiles = Select(setNameFiles, Real(Text nameFile)
{

```

```

Sub(Reverse(nameFile), 1, 3) == Reverse(extension)

```

```

});
Set pathDirExtFiles = EvalSet(dirExtFiles, Text(Text nameFile)
{ pathDir + "/" + nameFile });
pathDirExtFiles

```

```

});
Set setNameDir = dir[2];

```

```
Set setPathDir = If(EQ(Card(setNameDir),0), Empty,
{
Set dirExtFiles = EvalSet(setNameDir, Set(Text nameDir)
{
RecIncludeExtTree(pathDir + "/" + nameDir)

});
BinGroup("+", dirExtFiles)

});
setPathFiles << setPathDir

};
EvalSet(RecIncludeExtTree(pathDir), Set(Text pathExt) { Include(pathExt) })

};
PutDescription("Returns the inclusion of all files with the extension indicated below
the directory pathDir", IncludeExtTree);
```

2.4. Description

In this part of the function which refers to what the function makes, trying be brief and trying to explain the best possible of its arguments. The beginning of the description must explain what it the result of the function. It goes after the body and between two lines of slash. The structure of the PutDescription command is as it follows:

- **Description:** The description of the function is written taking care of the previous norms. When the text exceeds a line, it is forced to jump and it is aligned with the quotation marks of opening
- **Name of the function:** The name of the function is written to which the description aligned with the initial parenthesis makes reference. Next a function is explained which can serve as example for all the previously mentioned on functions.

APPENDIX

In this appendix, we will describe some relevant aspects concerning at the same time the generation of files of data in other publishers with excel and text editors.

Generation of files BDT with Excel

An option to generate this type of files is through excel, selecting to save as CSV comma delimited. In order to have at the end of each row semicolon it is needed to add a column with a space (that is to say, a column that in each one of the rows is a space solely).

Example We want to generate a table of data that contains the following fields:

We write the data in an excel file and we add a column that contains in each cell a blank space:

We select to save as CSV comma delimited (for example: in the *C:\Bayes* folder) with an identifying name (for example: *Table.csv*). If we open the resulting file with the notepad, we obtain:

In order to work with this file, that has format of table TOL (format BDT), write in a TOL file the following sentence. Klick on Save and compile (to specify a name to identify the file later on, for example *EjemploIncluirDatos.tol* Set *Tabla = IncludeBDT("C:\Bayes\Tabla.csv");*

This sentence indicates that our file has format of table BDT, the separation of the different columns comes indicated by semicolon (;) and the data begin in January of 1999 (y1999m01d01) and finalize in April of 1999 (y1999m04d01).

Cycles with While

This sentence allows executing repeatedly, while a certain condition, a sentence or block of sentences is fulfilled.

The general form is as it follows:

While(CondicionDeContinuacion, Iteracion)

Explanation: *CondicionDeContinuacion* is evaluated and

- If the result is false it does not evaluate Iteration and it continues with the execution of the program.
- If the result is true it executes Iteration and it comes to evaluate *CondicionDeContinuacion* (evidently some variable of these that take part in *CondicionDeContinuacion* will have to be modified, because if not. the loop would continue indefinitely).

In other words, Iteration is executed repeatedly while CondicionDeContinuacion is true, and it is let to be executed when CondicionDeContinuacion becomes false. Observe that in this case the control to decide if it leaves or not the loop is before Iteration, reason why it is possible that the Iteration is not to get execute nor a single time. The *While()* cycles always are associated to variables that are reassigned with the operator *:=*.

Example

The following example shows a simple cycle, where

- *n* defines the counter of the cycle
- *n*<10 defines the condition of shutdown
- and the Iteration consists of writing the number of iteration

```
Real n = 0;  
Real While  
( n<10,  
  
{  
  
  n := n+1;  
  
  WriteLn("Iteration number " + FormatReal(n));  
  n  
  
}  
  
);
```

Appearance of the window of messages:

```
Iteration number 1  
Iteration number 2  
Iteration number 3  
Iteration number 4  
Iteration number 5  
Iteration number 6  
Iteration number 7  
Iteration number 8  
Iteration number 9  
Iteration number 10
```

Example

This example shows a simple *While()* cycle where the condition of *LE(beta, alpha)* does not depend on a one by one increase of a variable, as in the previous case, but on the decrement of a variable *alpha*, that in each cycle of iteration is divided between two (*alpha := alpha/2*).

The counter variable, in this case, is only used with the object of visualizing the number of iterations to reach the condition of shutdown.

```
Real beta = 0.001;
```

```
Real alfa = 2;
```

```
Real counter = 0;
```

```
Real While
```

```
{
```

```
LE(beta,alfa),
```

```
{
```

```
counter := counter +1;
```

```
alfa := alfa/2;
```

```
WriteLn("Iteration number: " + FormatReal(counter));
```

```
WriteLn(" alfa value = " + FormatReal(alfa));
```

```
alfa
```

```
}
```

```
};
```

Appearance of the **window of messages**:

```
Iteration number: 1
```

```
alfa value = 1
```

```
Iteration number: 2
```

```
alfa value = 0.5
```

```
Iteration number: 3
```

```
alfa value = 0.25
```

```
Iteration number: 4
```

```
alfa value = 0.125
```

```
Iteration number: 5
```

```
alfa value = 0.0625
```

```
Iteration number: 6
```

```
alfa value = 0.03125
```

Iteration number: 7

alfa value = 0.015625

Iteration number: 8

alfa value = 0.0078125

Iteration number: 9

alfa value = 0.00390625

Iteration number: 10

alfa value = 0.00195312

Iteration number: 11

alfa value = 0.000976562

Example This example presents a nested *While()* cycle, where

- the main cycle is defined using accountant *n* and consists of the writing of the number of *WriteLn("Iteration iteration to number" + ormatReal(*n*))*.
- the secondary cycle comes defined by means of counter *m* and writes the number of subiteration ("Subiteration to number" + *FormatReal(m)*).

Real n = 0;

Real While

(

n < 5,

{

n := n + 1;

*WriteLn("Iteration number " + FormatReal(*n*));*

Real m = 0;

Real While

(

m < n,

{

m := m + 1;

*WriteLn("Subiteration number" + FormatReal(*m*));*

m


```
}
```

```
);  
n
```

```
}
```

```
);
```

Appearance of the window of messages:

Iteration number 1

Subiteration number 1

Iteration number 2

Subiteration number 1

Subiteration number 2

Iteration number 3

Subiteration number 1

Subiteration number 2

Subiteration number 3

Iteration number 4

Subiteration number 1

Subiteration number 2

Subiteration number 3

Subiteration number 4

Iteration number 5

Subiteration number 1

Subiteration number 2

Subiteration number 3

Subiteration number 4

Subiteration number 5