Time Oriented Language (TOL)

MANUAL BÁSICO DE PROGRAMACIÓN

Bayes Forecast

Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636

Contenido

Resumen

El objetivo de este manual es orientar a aquellas personas que comienzan a programar en **TOL**. El manual comienza con una descripción del lenguaje, cómo obtener el software y el entorno de trabajo. Los dos siguientes capítulos están enfocados a la programación con el lenguaje **TOL**. Los capítulos restantes describen los tipos de objetos propios de **TOL** para la explotación de la información temporal.

Se han incluido ejemplos de programación a lo largo de este manual, por lo que se espera el lector disponga de un computador personal con **TOL** y vaya ejecutando dichos ejemplos. Este manual describe la utilización de **TOL** bajo Windows XP.

Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636

1. Introducción

TOL (**Time Oriented Language**) es un lenguaje orientado a la explotación de la información temporal y la construcción de sistemas de atención dinámica a la demanda a través de una representación algebraica del tiempo. Entre sus objetivos están:

a. Organizar los datos de acuerdo a una estructura temporal para darles significado temporal.

b. Analizar los datos temporales para estudiar comportamientos pasados a través de la estimación y pronosticar comportamientos futuros, a través de la previsión.

Está orientado al análisis y explotación de datos temporales, para modelar y resolver problemas reales en el ámbito de la información dinámica. Además se añade un entorno de usuario, **Tolbase**, para facilitar el análisis de datos y el desarrollo de modelos de series temporales, tabular matrices, representaciones gráficas, etc.

Entre las características de **TOL** están, el ser un lenguaje:

a. Interpretado, lo que facilita el aprendizaje y uso para los analistas de datos, ya que permite fácilmente la modelización de series temporales.

b. Declarativo y además incorpora un tipo de variables `Anything` que permite trabajar con distintos tipos de variables.

c. Autoevaluable, es decir, se puede evaluar el código **TOL** que se está construyendo en tiempo de ejecución, lo cual es muy útil cuando se emplea un código para grandes cantidades de datos con diferentes estructuras.

d. De evaluación retardada (Lazy). Es especialmente adecuado cuando la información es infinita.

Entre sus aspectos técnicos están:

a. Está desarrollado en C++, por ser uno de los lenguajes más potentes y versátiles que existen y su orientación a objetos.

b. Está desarrollado con vocación multiforme y se ha compilado el código bajo Windows y UNIX.

1.1 Acerca de este manual

El objetivo de este manual es orientar a aquellas personas que comienzan a programar en **TOL**. El manual comienza con una descripción del lenguaje, cómo obtener el software y el entorno de trabajo. Los dos siguientes capítulos están enfocados a la programación con el lenguaje **TOL**. Los capítulos restantes describen los tipos de objetos propios de **TOL** para la explotación de la información temporal.

Se han incluido ejemplos de programación a lo largo de este manual, por lo que se espera el lector disponga de un computador personal con **TOL** y vaya ejecutando dichos ejemplos. Este manual describe la utilización de **TOL** bajo Windows XP.

1.2 ¿Cómo descargar el lenguaje?

La siguiente dirección de Internet **http://www.tol-project.org** permite acceder a la comunidad de desarrollo del lenguaje **TOL**. Desde esta página se puede acceder a los foros de discusión para el desarrollo de programas, anuncios, chats, reporte de fallos, mejoras, etc. y la descarga gratuita del software para Windows, Linux y un entorno de programación (**Tolbase**) desarrollado en C++. Dependiendo de la versión de **TOL** se precisa mayor o menor memoria. En este manual seguiremos la versión para Windows.

La página ofrece también un servicio on-line para responder a las preguntas más frecuentes, así como para solucionar posibles errores de instalación.

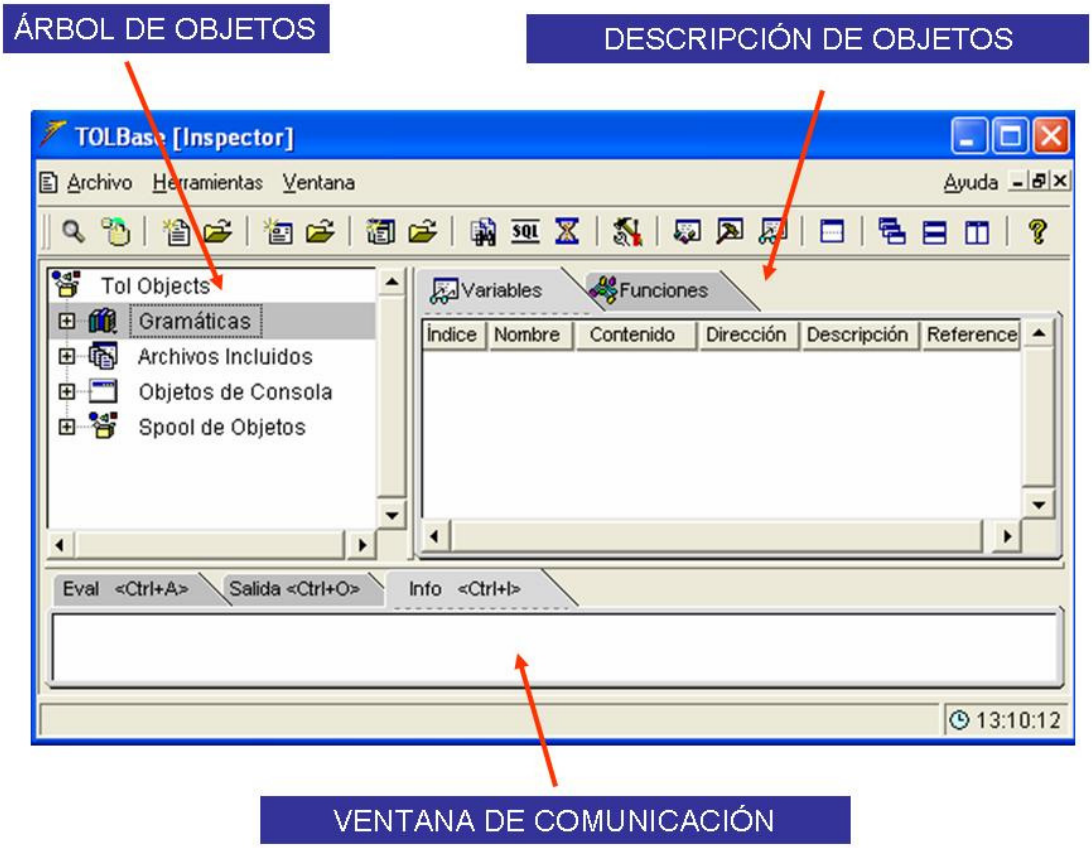
1.3 Entorno de trabajo

TOL se puede arrancar como cualquier otra aplicación de Windows, pulsando el botón derecho del ratón dos veces sobre el icono que se muestra a continuación, ubicado en la carpeta **bin** dentro de la carpeta `TolBase*.*` en la cual se ha descargado el programa (generalmente `C:\Archivos de Programa\Bayes\TolBase*.*`).


Tolbase.exe

Al arrancar **TOL** se obtiene la ventana que se muestra a continuación. Esta ventana inicial requiere una descripción más detallada.

Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636



La ventana de **descripción de objetos**, muestra las variables y funciones creadas durante el proceso de compilación. Se pueden compilar y decompilar de forma independiente diferentes ficheros. Esto es de gran utilidad cuando tenemos, por ejemplo, un conjunto de funciones definidas en un fichero A y las queremos emplear para otro conjunto de sentencias en un fichero distinto B. Así pues, las funciones definida en el fichero A estarán disponible para todos los ficheros siguientes siempre y cuando no decompilemos el fichero A. Y podremos entre tanto trabajar con el fichero B, compilando y decompilando tantas veces sea necesario con las funciones del fichero a siempre disponibles.

La **ventana de comunicación**, consta de tres subventanas:

- **Eval**: esta ventana nos permite escribir sentencias en código **TOL**. Cada sentencia debe ir separada de la siguiente por **punto y coma** (;). Para ejecutar las sentencias, presionamos en compilar  Si hemos compilado previamente, hay que decompilar primero  (para destruir las variables que se han creado al compilar).

- **Salida**: Muestra los errores de compilación, la información de los ficheros incluidos y los mensajes que emiten los programas.

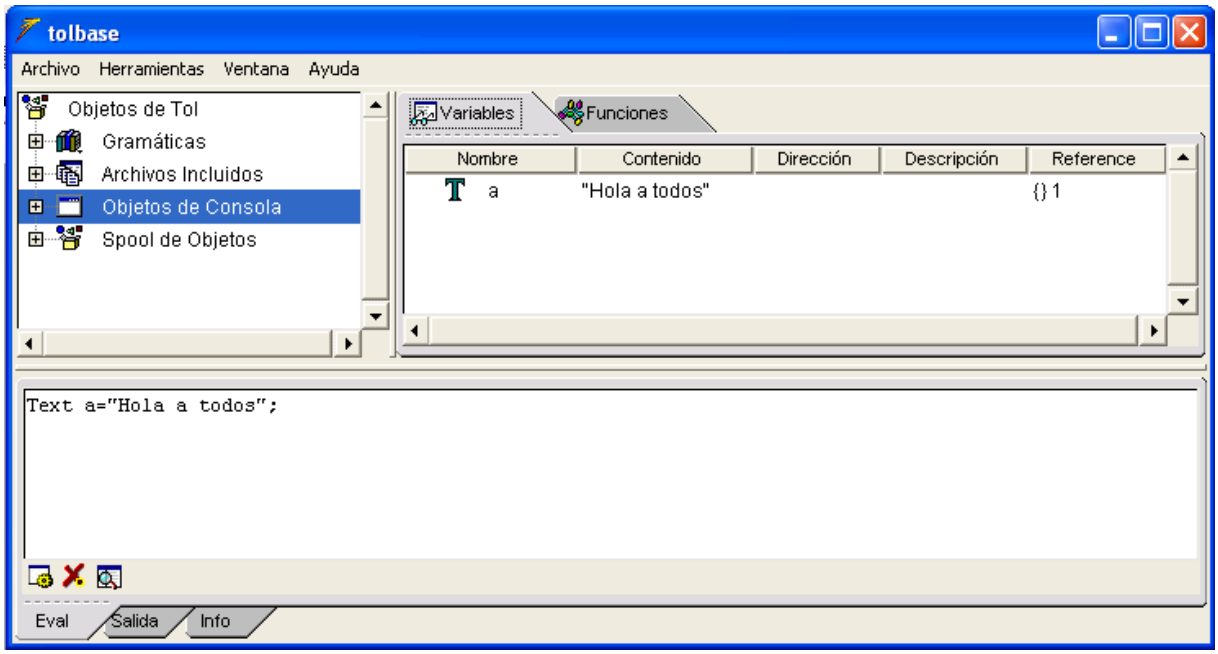
- **Info**: Muestra la información relativa a las variables o funciones creadas durante el proceso de compilación.

Ejemplo

Escribimos en la ventana Eval la siguiente sentencia:

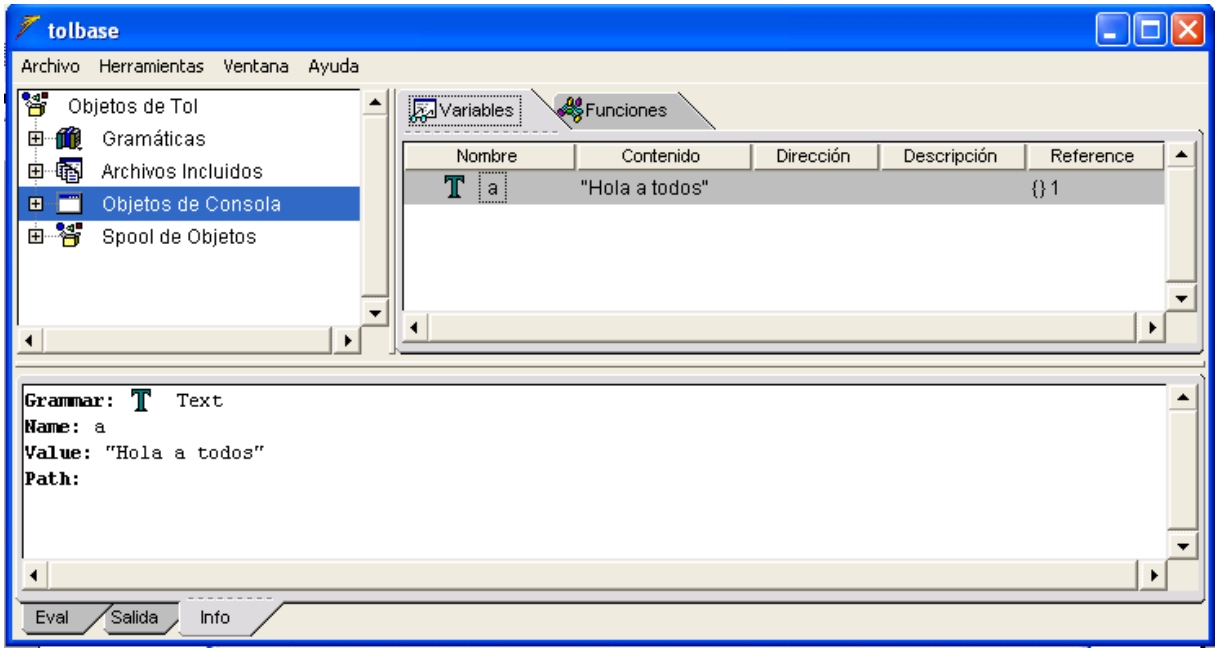
```
Text a="Hola a todos";
```

Presionamos en compilar  . En la ventana de variables se obtiene la nueva variable a creada (ver la siguiente figura).



Si seleccionamos la pestaña **Info**, y se pulsa sobre en la variable creada, se muestra la información relativa a dicha variable (ver la siguiente figura): Tipo de objeto (**Grammar**), nombre (**Name**), valor (**Value**) y fichero en el cual se encuentra definida la variable (**Path**) (En este caso, como hemos escrito el código directamente en la ventana Eval este campo queda vacío).

Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636



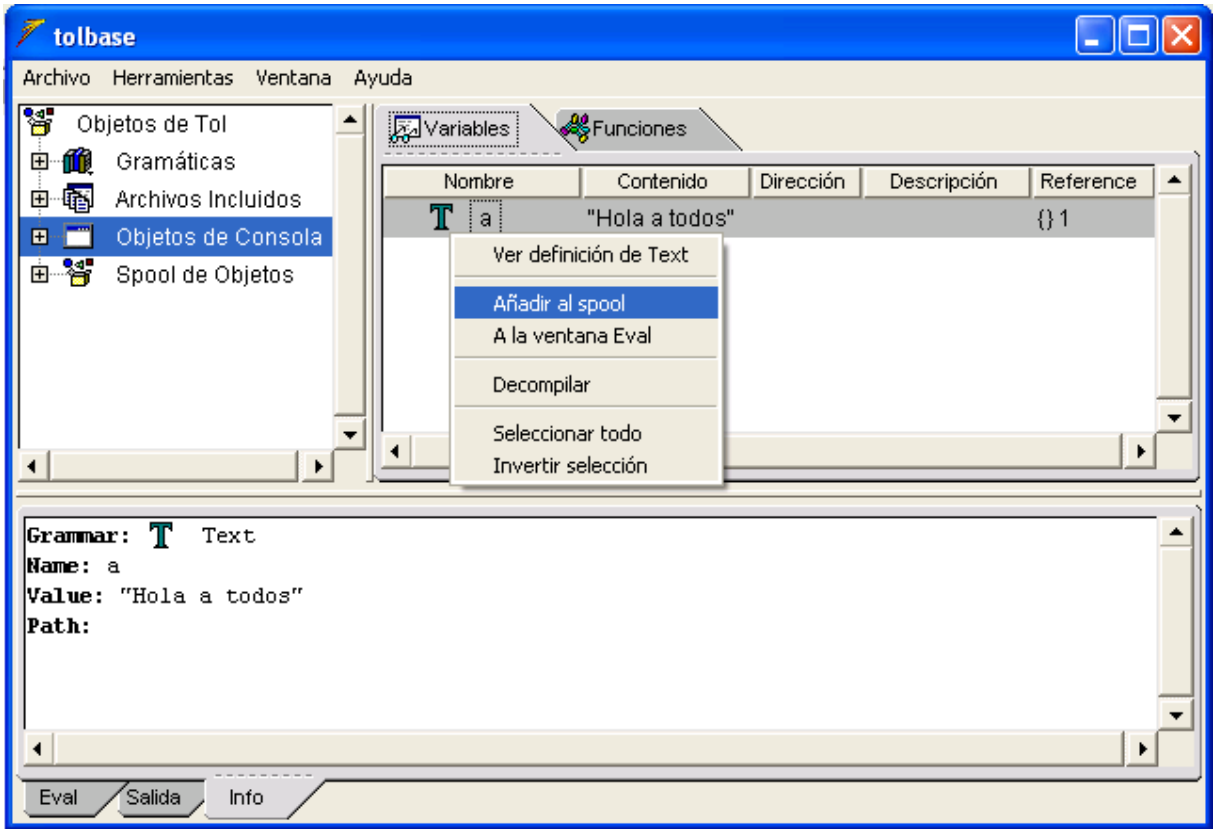
El **árbol de objetos**, muestra los tipos de objetos disponibles:

- **Gramáticas**: Tipos de variables y funciones disponibles en **TOL** para escribir sentencias organizadas por tipos de datos.

- **Archivos incluidos**: Muestra los archivos incluidos en el proceso de compilación desde un fichero **TOL**. Al iniciar **TOL**, se incluye el conjunto de ficheros `_inittol.tol`.

- **Objetos de consola**: Muestra las variables, funciones y archivos incluidos en el proceso de compilación desde la ventana **Eval**.

- **Spool de Objetos**: Sirve para llevar objetos desde las tres opciones anteriores y poder visualizarlas o trabajar con ellos. Por ejemplo, si se está interesado en pintar el gráfico de una serie creada desde la ventana **Eval** junto con otra serie creada a partir de un fichero **TOL**. En este caso, se añaden ambas series a **Spool de Objetos** seleccionando cada serie con el botón derecho del ratón y presionando el botón izquierdo. En el menú de contexto que se despliega, elegimos la opción **Añadir al spool**.



Cada vez que decompilemos , se eliminan los objetos del **Spool**.

Por último, la barra de herramientas superior permite las siguientes opciones:

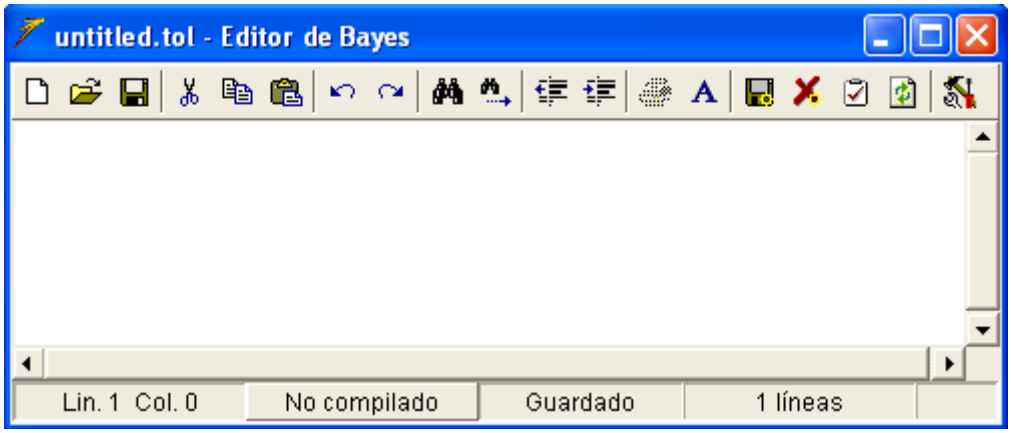
- **Archivo**: Permite crear, abrir, incluir archivos recientes de **TOL**.
- **Herramientas**: Permite editar ficheros, tablas y proyectos **TOL**, buscar funciones y variables entre todos los archivos incluidos en los procesos de compilación, seleccionar los tipos de calendarios, manipulación de bases de datos a través de **SQL**, conexiones a base de datos y opciones de formato de números y calendarios.
- **Ventana**: opciones de visualización de las ventanas de **TOL**.
- **Ayuda**: describe la versión de **TOL**.

1.4 Tipos de ficheros TOL

El tipo de ficheros más importante en **TOL**, son los ficheros de código. En ellos, podemos escribir sentencias para ejecutarlas posteriormente y guardar el trabajo realizado para futuras sesiones. Para generar un fichero **TOL**, basta con pinchar en Archivo – Nuevo - Archivo TOL:

Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636

Tenemos el siguiente editor de código:



El editor permite abrir, guardar, copiar, pegar, cortar, etc. entre otras opciones de edición y compilar y decompilar las sentencias generadas. Además, permite revisar la sintaxis y opciones de edición de sintaxis

En lo que sigue, emplearemos este tipo de archivos para guardar los ejemplos. Por tanto, guardamos este fichero (por ejemplo: en la carpeta `C:\Bayes`) con un nombre identificador (por ejemplo: `Ejemplo.tol`).

Además, **TOL** posee sus propios métodos de almacenamiento y recuperación de la información, en forma de ficheros **BDT** (***Bayes* Data Table**) para la definición de series temporales o **BST** (***Bayes* Struct Table**) para la definición de tablas.

Sus datos de entrada o los resultados obtenidos con **TOL** pueden ser intercambiados con sistemas de gestión de **bases de datos** o con sistemas de ofimática como hojas de cálculo o procesadores de texto.

TOL también posee sus propios métodos de presentación de resultados en forma de gráficos y de tablas.

1.4.1 ¿Cómo se escribe una serie temporal TOL?

El formato que **TOL** espera es el siguiente:

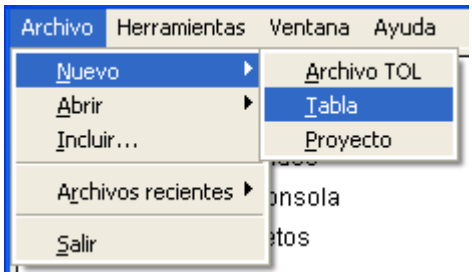
```
FechadoSerie; NombreSerie1; NombreSerie2;…. NombreSerieN;

Fecha 1; dato S11; dato S21;….;dato SN1;

Fecha 2; dato S12; dato S22;….;dato SN2;

…

Fecha k; dato S1k; dato S2k;….;dato SNk;
```

El formato de las fechas es año/mes/día (aaaa/mm/dd). Si tenemos una serie en fechado anual, que no precisa de indicar ni el mes, ni el día, se puede fijar el mes en Enero (mm=01) y el día 1 (dd=01). Este formato se corresponde con ficheros **BDT** (***Bayes*** **Data Table**).

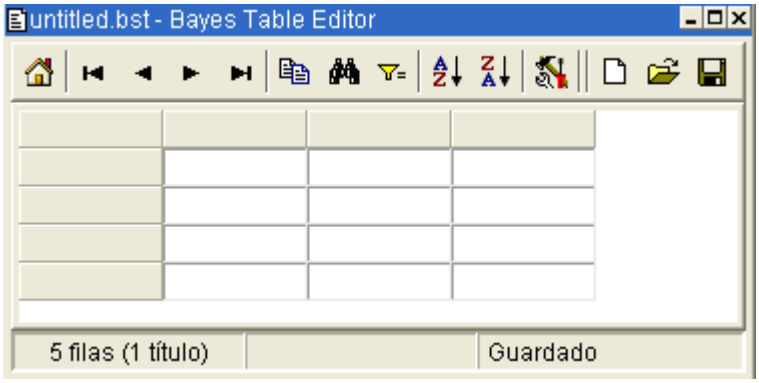
Ejemplo: Queremos generar una tabla de datos que contenga los siguientes campos:

Monthly	Sales	Purchases
Enero 2005	3.250.000	200.400
Febrero 2005	4.130.000	315.800
Marzo 2005	4.250.000	285.760
Abril 2005	3.400.000	320.150

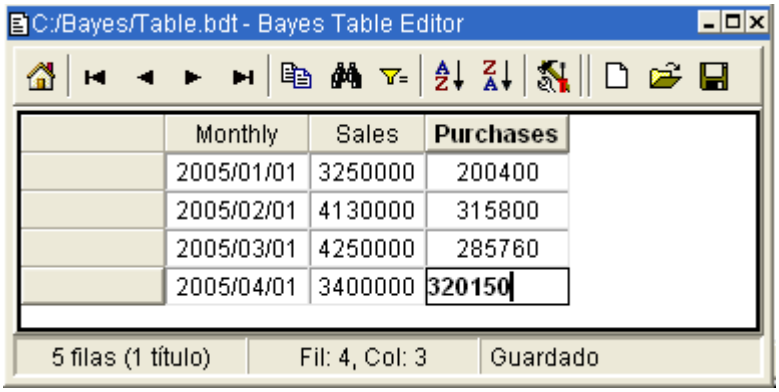
Seleccionamos Archivo – Nuevo - Tabla:



Tenemos entonces el siguiente editor de tablas:



La primera fila debe contener el fechado y los nombres de las series. Introducimos los datos como sigue y seleccionamos guardar este fichero (por ejemplo: en la carpeta `C:\Bayes`) como ***Bayes* Data Table** con un nombre identificador (por ejemplo: `Table.bdt`)



Para trabajar con este fichero, que tiene formato de tabla **TOL** (formato **BDT**), escribimos en un fichero **TOL** la siguiente sentencia. Pinchamos en Guardar y compilar (especificando un nombre para identificar posteriormente el archivo, por ejemplo `ExampleIncludeData.tol`

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

Esta sentencia indica que nuestro fichero tiene formato de tabla **BDT**, la separación de las distintas columnas viene indicada por un punto y coma (;) y los datos comienzan en Enero de 1999 (y1999m01d01) y finalizan en Abril de 1999 (y1999m04d01).

1.4.2 ¿Cómo se genera una tabla TOL?

El formato que **TOL** espera es el siguiente:

```
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  ;
```

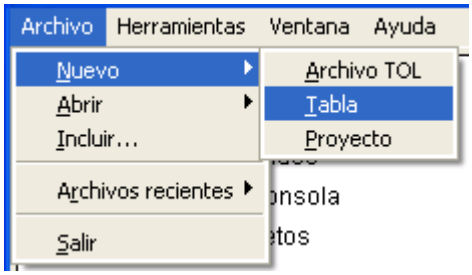
Este formato se corresponde con ficheros **BST** (***Bayes* Struct Table**). La numeración de los campos no es obligatoria, se puede dejar en blanco empezando el fichero con `;`.

Ejemplo Queremos generar una tabla de datos que contenga los siguientes campos estructurados:

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

Asignamos un nombre a la estructura de la tabla de datos: por ejemplo `PersonalData`, que contiene dos campos de texto (`Name`, `Surname`) y un campo Real (`Age`).

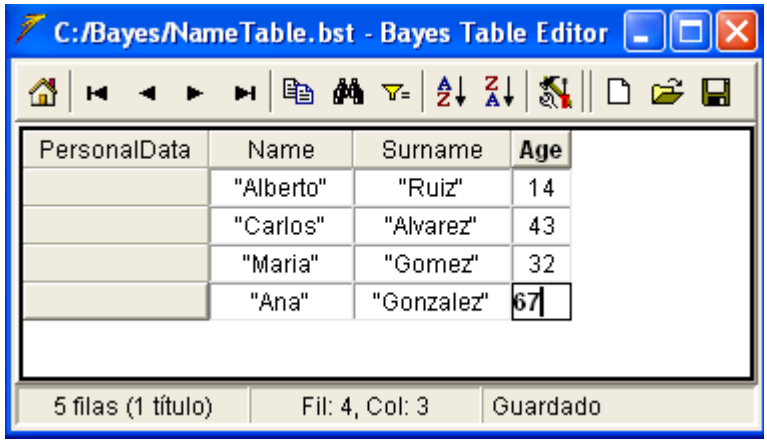
Seleccionamos Archivo – Nuevo - Tabla:



Tenemos entonces el siguiente editor de tablas:

Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636

La primera fila debe contener el nombre de la estructura y los nombres de los campos. Los campos de las variables de texto deben ir entre comillas (“ “) Introducimos los datos como sigue y seleccionamos guardar este fichero (por ejemplo: en la carpeta `C:\Bayes`) como ***Bayes* Struct Table** con un nombre identificador (por ejemplo: `NameTable.bst`)



Para trabajar con este fichero, que tiene formato de tabla **TOL** (formato **BST**), escribir en un fichero **TOL** las siguientes sentencias. Pinchamos en Guardar y compilar (especificando un nombre para identificar posteriormente el archivo, por ejemplo `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");
```

Esta sentencia indica que nuestro fichero tiene formato de tabla BST.

2. Manejo básico de TOL

En este capítulo describiremos brevemente los tipos de variables que se encuentran disponibles en **TOL** y el manejo básico de ficheros.

2.1 Tipos de variables disponibles en TOL

TOL es un lenguaje tipado orientado a operaciones con series y conjuntos temporales aunque disponen de otros muchos tipos de datos de textos, reales, etc.. Los tipos de objetos disponibles son los siguientes:

2.1.1 Text

Un objeto de tipo texto puede contener cualquier cadena de caracteres ASCII. La forma de crear una variable de tipo texto es poner el texto entre comillas. **TOL** permite diversos tratamientos de las variables de texto (búsqueda de cadenas, transformaciones, etc.)

Ejemplo

```
Text MyName = "Ana";
```

2.1.2 Real

Variables numéricas, que se definen como reales de doble precisión. También se incluye el valor desconocido (?), infinito o indeterminado. El separador decimal es el punto. **TOL** incorpora múltiples funciones de manipulación (suma, producto, etc.) y transformación de reales (transformación trigonométricas, estadísticas, etc).

Ejemplo

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

2.1.3 Date

Fechas. El formato de las fechas es y2004m01d23 (aaaa/mm/dd). Todos los campos son necesarios, pudiéndose eliminar el carácter 0. Si tenemos una serie en fechado anual, que no precisa de indicar ni el mes, ni el día, se puede fijar el mes en Enero (mm=01) y el día 1 (dd=01).

Ejemplo

```
Date Meeting = y2005m05d03;
```

2.1.4 Complex

Números complejos. La parte imaginaria de un número complejo se indica a través de la variable `i`.

Ejemplo

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

2.1.5 Matriz

Matrices de números reales. **TOL** incorpora múltiples funciones de manipulación (suma, producto, etc.), transformación de matrices (transformación de Choleski, etc.) y resolución de sistemas lineales.

Ejemplo

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

2.1.6 Set

Conjuntos. Son colecciones de objetos de cualquier tipo incluyendo otros conjuntos, lo cual permite construir estructuras como vectores, tablas, árboles, etc. **TOL** permite el tratamiento y manipulación de los conjuntos enfocado al trabajo con series temporales.

Ejemplo

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

2.1.7 Serie

Series temporales. **TOL** permite el tratamiento y la manipulación algebraica de series temporales, para realizar operaciones, graficar, estimar modelos y predecir datos.

Ejemplo

```
Serie PulseY2005 = Pulse(y2005m01d01,Yearly);
```

2.1.8 TimeSet

Conjuntos temporales, que son subconjuntos, finitos o infinitos, del conjunto de todas las fechas desde el comienzo hasta el final del tiempo. Un conjunto temporal infinito podría ser, por ejemplo, todos los lunes que caen en primero de mes, todos los viernes y trece, todos los domingos de Pascua, etc. **TOL** permite operar con los conjuntos temporales para construir nuevos conjuntos, lo cual es muy útil para manejar series temporales.

Ejemplo

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

2.1.9 P(x) Polyn

Polinomios. Su principal utilidad es que permiten el desplazamiento temporal de las series a través de operadores de retardo o adelanto. **TOL** permite operaciones entre polinomios tales como el producto, suma, potencias, etc.

Ejemplo

```
Polyn Stepwise = F+B^2+F^3+B^4+F^5+B^6+F^7;
Serie StepStepwise = Stepwise:Step(y2004m3d01,Monthly);
```

2.1.10 P(x)/Q(x) Ratio

Fracciones racionales de polinomios. Una función racional de retardos se define como un cociente de polinomios de retardo. Su principal utilidad es resolver ecuaciones en diferencias del tipo $P(B)Z_t = Q(B)A_t$. **TOL** permite operaciones entre fracciones racionales tales como el producto, suma, potencias, etc.

Ejemplo

```
Ratio Fraction = F/(1-B^5);
```

2.1.11 f() Code

Propio código **TOL**. Las funciones disponibles se visualizan en las variables de la ventana del inspector de objetos.

2.1.12 ? Anything

Admite cualquier tipo. Es de gran utilidad para manejar las funciones asociadas de tipo **Case(), Eval(),Field(), Find(), If(), While()**, etc.

Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636

2.2 Acceso a bases de datos

Para importar y manipular una base de datos, **TOL** incorpora la conexión a un origen de datos remoto vía **ODBC**.

Para ejecutar consultas a bases de datos, primero deben abrirse. Para ello, empleamos la sentencia

```
Real DBOpen(Text alias, Text usuario, Text clave)
```

Debe existir un alias a la base de datos **ODBC** dado de alta en **DSN** de Windows/Unix. Esta función retorna **1** (verdadero) en caso de éxito y **0** (falso) en caso de error.

Entre las diversas opciones de manejo de bases de datos, se encuentran:

1. `DBExecute(Text consulta)`, que permite actualizar y modificar la base de datos a través de las sentencias **SQL**.

2. `DBSeries(Text consulta, TimeSet fechado, Set nombres)`: Incluye un conjunto que contiene las series con los nombres (`Set nombres`) y en el fechado indicado (`TimeSet fechado`) cuyos datos vienen dados por una consulta a una base de datos abierta (`Text consulta`).

3. `DBTable(Text consulta [, Text NombreEstructura])`: Incluye un conjunto que contiene una tabla como resultado de la consulta realizada a la base de datos abierta (`Text consulta`). El segundo elemento dota de la estructura a cada uno de los registros de la tabla. Esta estructura debe estar definida previamente (`Text NombreEstructura`).

4. `DBSpool(Text consulta, Text fichero)` permite guardar en un fichero (`Text fichero`) la consulta realizada (`Text consulta`) en la base de datos.

Para finalizar las consultas a las bases de datos, éstas deben cerrarse. Para ello, empleamos el comando

```
Real DBClose(Text alias)
```

Esta función retorna **1** (verdadero) en caso de éxito y **0** (falso) en caso de error.

2.3 Lectura / Escritura de ficheros

2.3.1 Lectura

Para incluir ficheros de datos (serie, matriz, tabla, etc.) generada desde **Tolbase** se emplea el comando **`Include`** que genera un conjunto (set). Entre las formas de inclusión más útiles, se encuentran:

a. **`IncludeBDT`**: Incluye el conjunto de series temporales con el mismo fechado y entre las mismas fechas definidas con formato **BDT** (***Bayes* Data Table**).

b. **`IncludeBMT`**: Incluye el conjunto de una matriz con formato **BMT** (***Bayes* Matrix Table**).

c. **`IncludeBST`**: Incluye el conjunto de una estructura con formato **BST** (***Bayes* Structured Table**).

d. **`IncludeTOL`**: Incluye el conjunto de objetos de un fichero **TOL**. Los tabuladores, saltos de línea y caracteres de espacio consecutivos son equivalentes a un único carácter de espacio.

e. **`BDTFile(Set cto [, Text NombreFichero , Text cabecera])`**: genera un fichero con estructura **BDT** (***Bayes* Data Table**), que es importable y exportable desde hojas de cálculo y sistemas de gestión de bases de datos.

f. **`BSTFile(Set cto, Text NombreFichero)`**: genera un fichero con estructura **BST** (***Bayes* Structured Table**), para el almacenamiento de estructuras de datos de cualquier tipo en forma de tabla.

Para la inclusión de **ficheros de código**, basta con indicar la ruta de acceso a través de la función **`Include()`**. El objeto que se genera es de tipo **`Set`**.

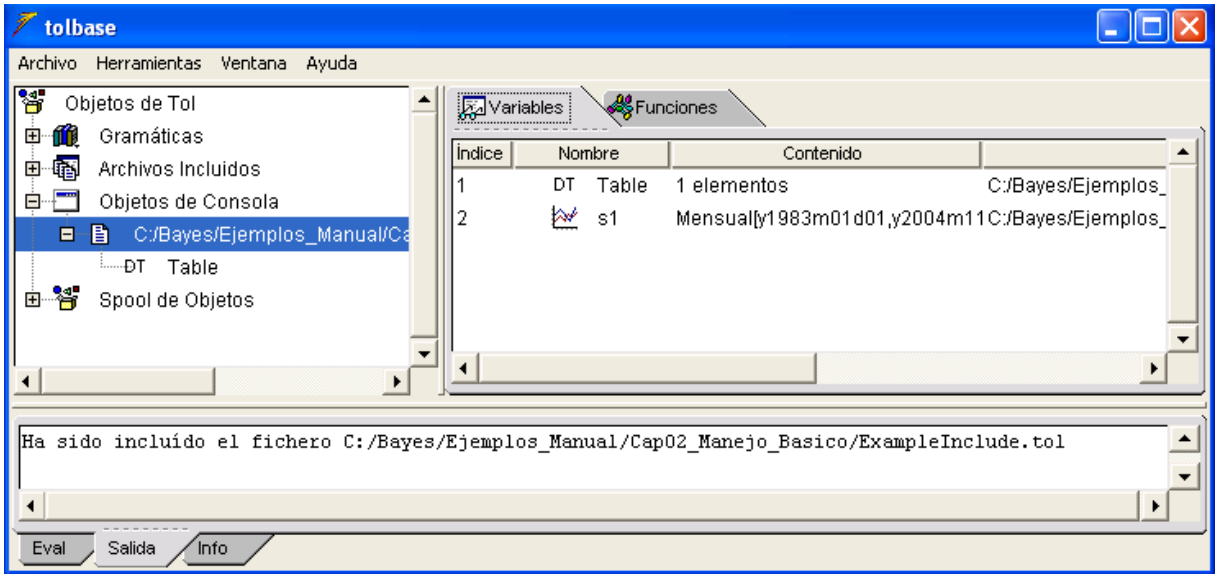
Ejemplo

Para incluir el fichero `ExampleInclude.tol`, comprobamos que se encuentra ubicado en `C:\Bayes\Ejemplos_Manual\Cap02_Manejo_Basico`.

Para añadir el fichero, escribir en la ventana Eval:

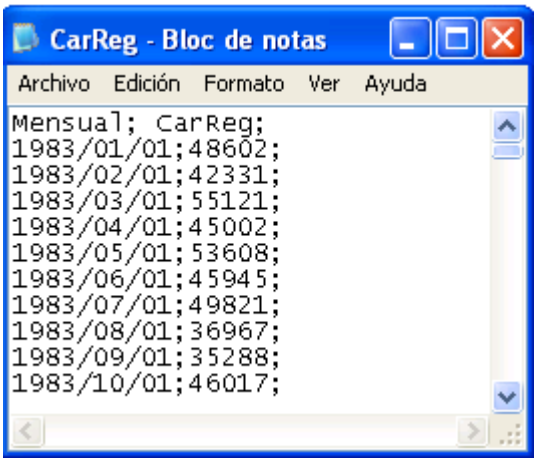
```
Set Include("C:\Bayes\Ejemplos_Manual\Cap02_Manejo_Basico\ExampleInclude.tol");
```

El resultado, es un fichero que se añade en Archivos incluidos, como muestra la siguiente figura:



Ejemplo

Estamos interesados en la lectura e inclusión de los datos de una serie en fechado mensual contenidos en el siguiente fichero



Comprobamos que el fichero `carreg.bdt` se encuentra ubicado en `C:\Bayes\Ejemplos_Manual\Cap02_Manejo_Basico`.

Para añadir el fichero, escribir en la ventana Eval:

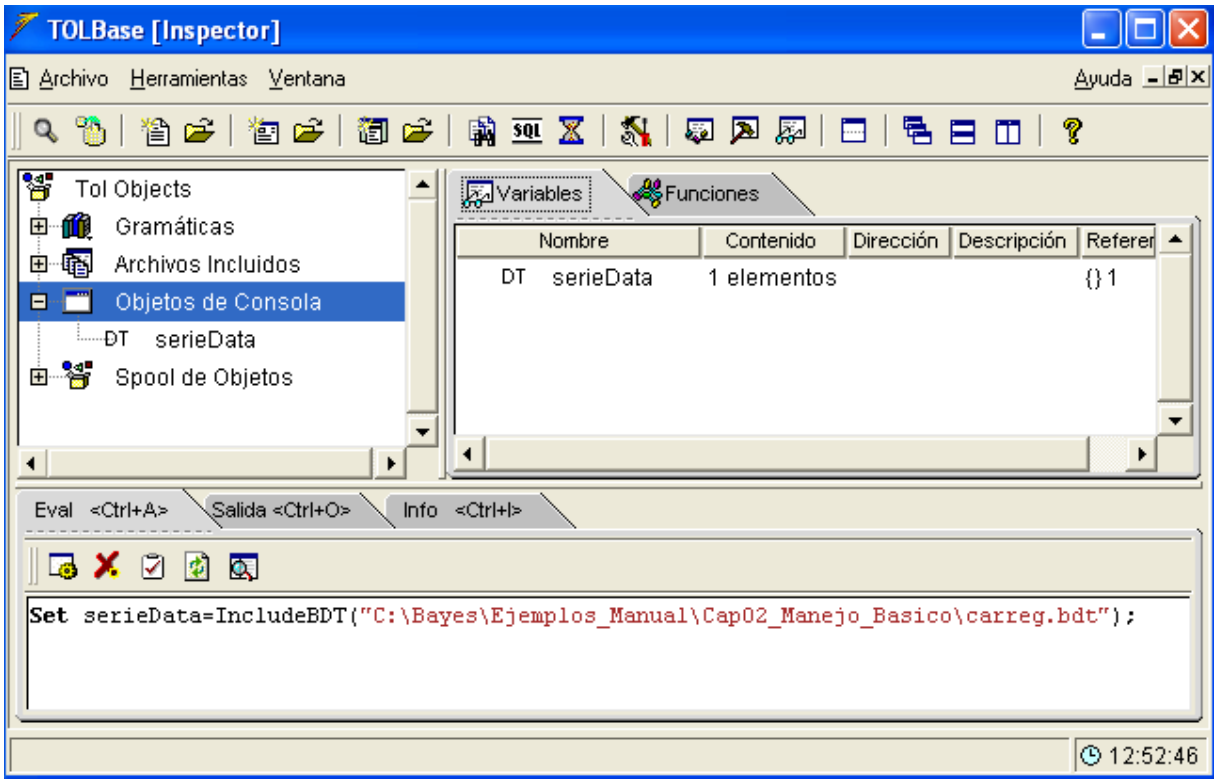
```
Set serieData=IncludeBDT("C:\Bayes\Ejemplos_Manual\Cap02_Manejo_Basico\carreg.bdt");
```

Observamos que:

a. Dentro del fichero `CarReg.bdt` se define una serie `CarReg` sobre el fechado **`Monthly (Mensual)`**. Esto se especifica mediante la cabecera del fichero (**`Monthly`**; `CarReg`;).

b. A partir de la carga de este fichero **BDT**, la serie `CarReg` esta disponible para su manipulación y uso.

Una vez cargados los datos, el contenido del fichero `carreg.bdt` puede consultarse a través del **inspector de objetos** del entorno **Tolbase**. A partir de la ejecución de la sentencia **`IncludeBDT()`** se crea un conjunto (**`Set`**) con el nombre `serieData` que contiene la serie `CarReg`. En el **inspector de objetos**, a la derecha de la serie, puede consultarse el conjunto temporal para el que las series están definidas (**`Monthly`**) y la fecha de comienzo y fin de datos (desde el día 1 de Enero de 1.983 hasta el día 1 de Noviembre de 2004).



2.3.2 Escritura en pantalla

Para la escritura en el **terminal o ventana de salida**, podemos utilizar:

- función **`Write()`** que escribe en el terminal el texto dado.
- función **`WriteLn()`** escribe en el terminal el texto dado y salta una línea.

Ejemplo

Al escribir el siguiente conjunto de sentencias en la ventana Eval:

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

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

Obtenemos los siguientes mensajes en la **ventana de salida**

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

Para la escritura en **ficheros** en formato libre, podemos utilizar:

- **`WriteFile()`** que sobrescribe un fichero con un texto, si ya ha sido creado anteriormente o lo crea si no existe.
- **`AppendFile()`** que añade un texto dado al final de un fichero, si éste ha sido creado anteriormente o lo crea si no existe.

Ejemplo

Al escribir el siguiente conjunto de sentencias en la ventana Eval:

```
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");
```

Se obtienen tres nuevos ficheros en `C:\Bayes\Ejemplos_Manual\Cap02_Manejo_Basico`.

3. Programación en TOL

En este apartado se supone que el lector tiene algunos conocimientos de programación por lo que nos centraremos más en los detalles del lenguaje.

3.1 ¿Por qué las características de TOL?

Una de las características de **TOL** es la evaluación retardada (lazy). Esto se debe al manejo de series temporales definidas en conjuntos temporales infinitos y por tanto se requiere que la evaluación de las operaciones sea lo más tarde posible, cuando la evaluación se refiera a un subconjunto finito.

Por ejemplo, si definimos una serie temporal que comienza al principio del calendario (**TheBegin**) y finaliza cuando termina (**TheEnd**):

```
Serie PulseElec = Pulse(y2004m03, Monthly);
```

TOL, no evalúa la expresión, sino que guarda en memoria la definición. Si deseamos crear una nueva serie que sea el doble de la anterior:

```
Serie PulseElecDouble = 2* PulseElec;
```

De nuevo **TOL** aguarda a que la definición de ambas series se encuentre definida en un intervalo temporal finito. Por ejemplo, podemos acotar las series anteriores en el periodo de tiempo entre el año 2000 y el año 2005:

```
Serie PulseElec2000 = SubSer(PulseElec,y2000,y2005);
Serie PulseElecDouble2000 = SubSer(PulseElecDouble,y2000,y2005);
```

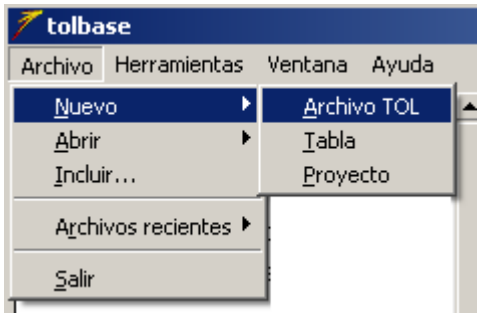
Ahora **TOL** evalúa la última expresión en caso de ser necesario. En caso de intentar evaluar una serie infinita (`Serie PulseElec`) por ejemplo para graficarla, **TOL** limita internamente la evaluación en un conjunto temporal finito (`DefaultDates()`).

3.2 Funciones

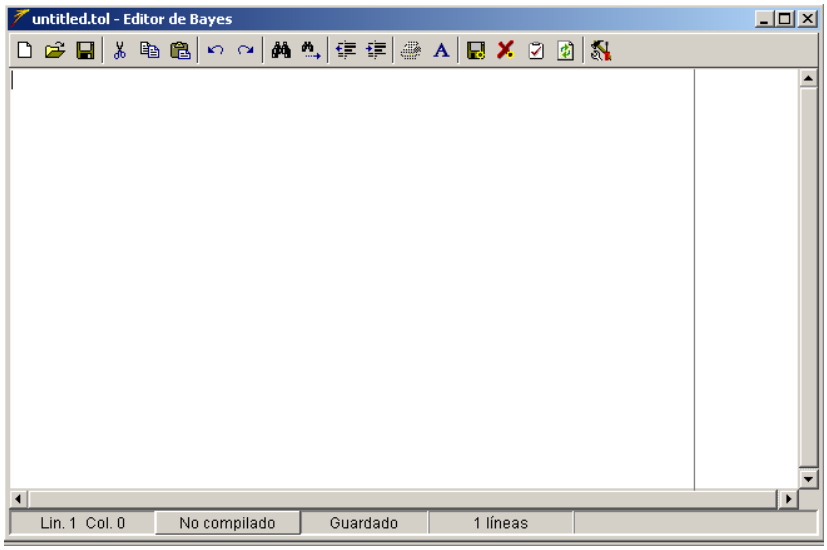
Una parte esencial del correcto diseño de un programa de computador es su modularidad, esto es su división en partes más pequeñas de finalidad muy concreta, **funciones**.

3.2.1 Crear una función con TOL

A continuación vamos a crear una nueva función en **TOL**. Para ello, abrir un nuevo fichero pinchando en Archivo – Nuevo - Archivo TOL:

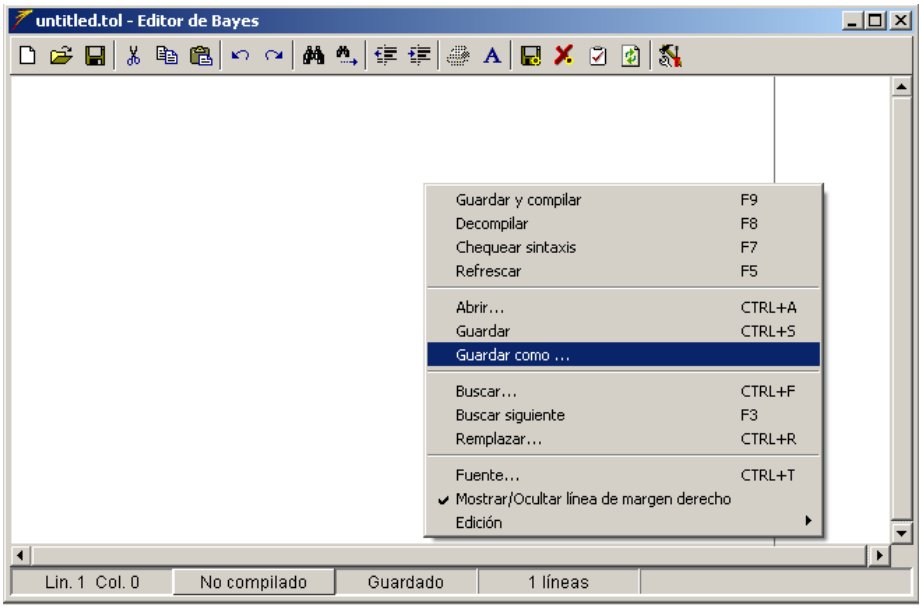


que de forma automática, recibe el nombre de `untitled.tol`:

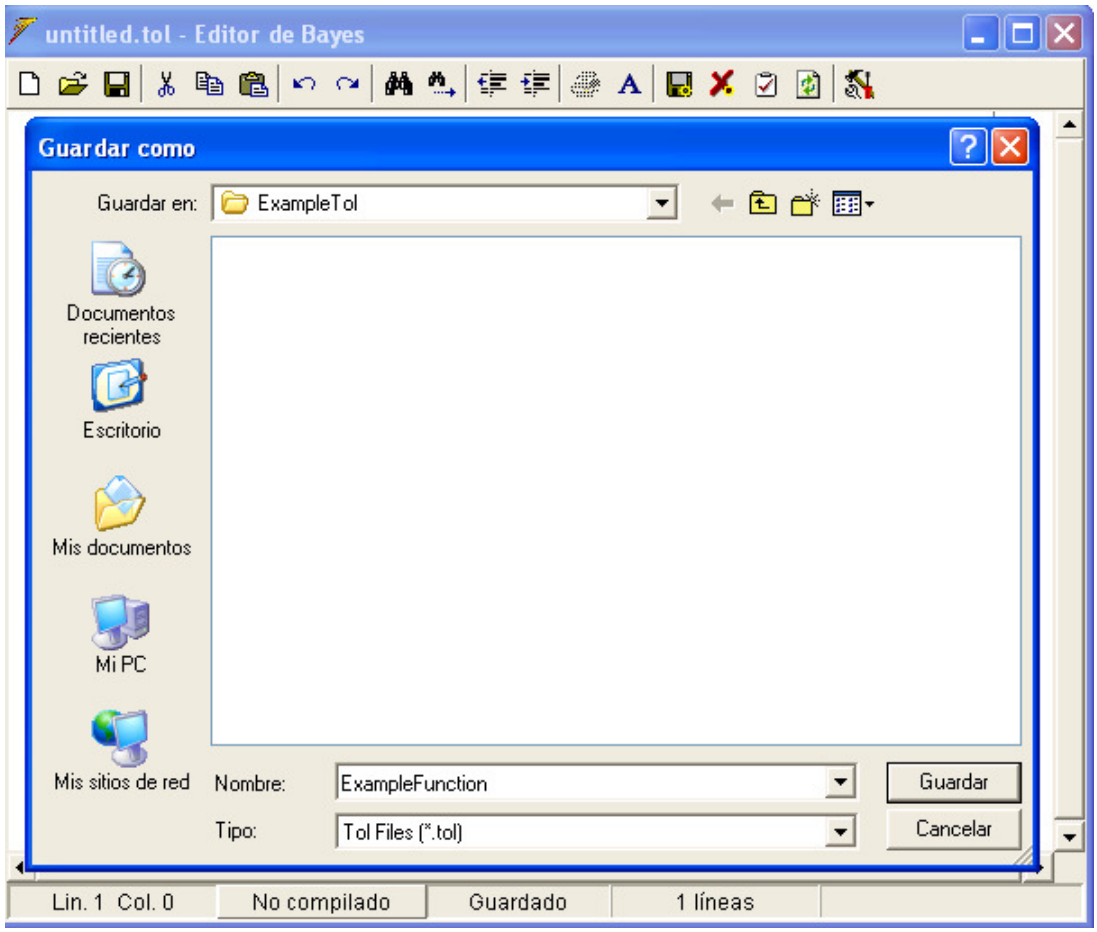


Es aconsejable dar un nombre a este nuevo fichero que nos permita identificarlo fácilmente más adelante y guardarlo en el directorio de trabajo que se desee antes de comenzar a escribir la nueva función. Por ello, crear en el disco duro del computador, una carpeta con el nombre `ExampleTol`. Para guardar este fichero, se puede

- utilizar el icono guardar  del menú principal del nuevo fichero, o bien
- utilizar la opción **Guardar como …** del menú que aparece al pulsar con el botón derecho del ratón sobre el **editor** de código del nuevo fichero:



se despliega un diálogo para guardar el fichero **TOL** con el nombre deseado, en este caso, `ExampleFunction.tol` y lo guardamos en el directorio `ExampleTol`:



Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636

Observación: El **editor** de código **TOL** proporcionado por el entorno **Tolbase** no es el único editor de texto que nos permite crear funciones, proyectos y modulos en **TOL**. Es también posible utilizar cualquier otro editor de textos disponible en el sistema operativo en el que se esté trabajando, por ejemplo en **Windows** se puede utilizar:

- El editor edit de DOS,
- El notepad,
- El wordpad, salvando siempre en modo texto o
- El procesador de textos word, salvando siempre en modo texto como en el caso anterior.

Podemos observar que una de las ventajas que proporciona utilizar el **editor** de código **TOL** es que proporciona una sintaxis realzada con el fin de facilitar la lectura del código al usuario:

- Diferenciando los textos que van entre comillas, `"text one"`,
- Resaltando en negrita los diferentes tipos de objetos (**`Serie`**, **`Set`**, **`Text`**, **`TimeSet`**, **`Matrix`**, **`Ratio`**, **`Polyn`**, **`Real`**, **`Date`** y **`Code`**), etc.
- Los comentarios marcados con los **caracteres `//`** o con los **caracteres `/*` y `*/`**, por ejemplo:

```
////////////////////////////////////////////////////////////////////////////
// Serie, time series,
////////////////////////////////////////////////////////////////////////////
```

El formato para escribir una función es:

```
Tipo NombreFuncion(Tipo Argumento1,…,Tipo ArgumentoN)
{
  Sentencia 1;
  …
  Sentencia K-1;
  Sentencia K
}
```

Debemos tener en cuenta que **TOL** retorna siempre la última sentencia. Es más, para cualquier conjunto de sentencias agrupadas entre llaves `{Sentencia 1;…; Sentencia K}`, **TOL** retorna siempre el resultado de evaluar la última sentencia. Por eso, **la última sentencia no termina en `;` como el resto**.

Esta característica es la que hace innecesaria en **TOL** el operador **return** como en otros lenguajes de programación.

Ejemplo Queremos crear una función que retorne el factorial de un número entero.

```
////////////////////////////////////////////////////////////////////////////
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);
```

No es necesario asignar un nombre a todas las funciones, a veces, estamos interesados en crear una función únicamente para asignar un valor a una variable.

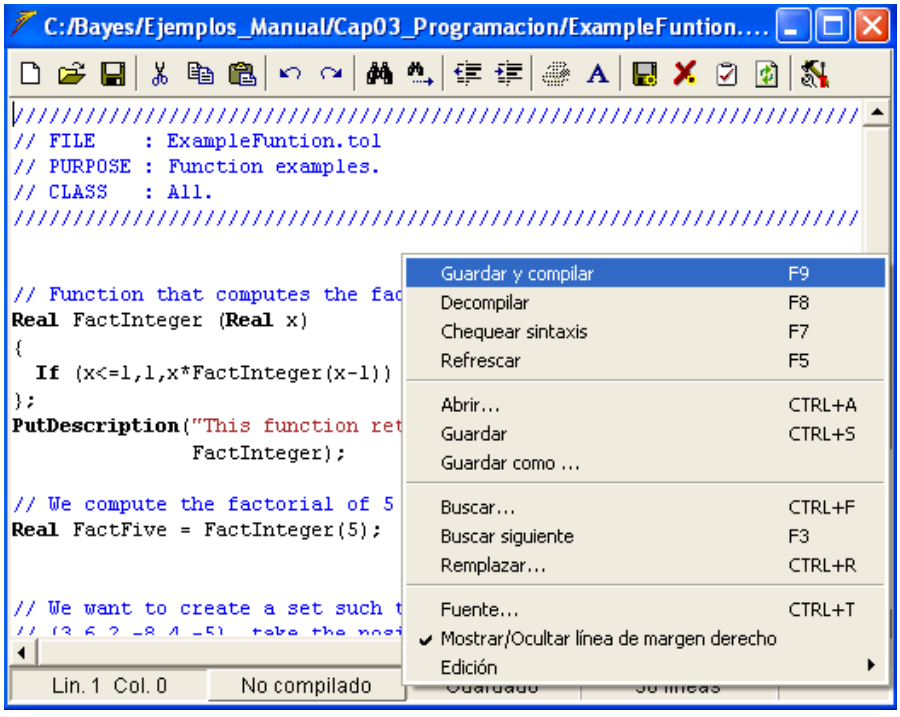
Ejemplo Queremos crear un conjunto que dado el siguiente conjunto de números reales {3,6,2,-8,4,-5}, tome los elementos positivos y en caso de elementos negativos retorne el valor desconocido.

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

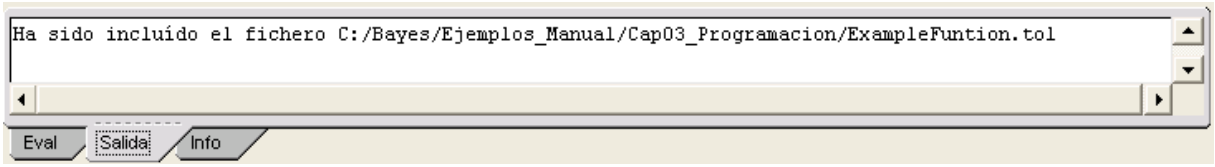
3.2.2 Compilar una función con TOL

Una vez introducido el código en el fichero y para compilarlo y ejecutarlo se puede:

- Utilizar el icono, guardar y compilar del menú principal de **Tolbase** ,o bien
- Utilizar la opción **Guardar y compilar** del menú que aparece al pulsar con el botón derecho del ratón sobre el editor **TOL**.



En la ventana de salida, podemos entonces ver si hemos cometido errores de sintaxis en el código, así como la confirmación de que el fichero ha sido compilado y ejecutado correctamente,

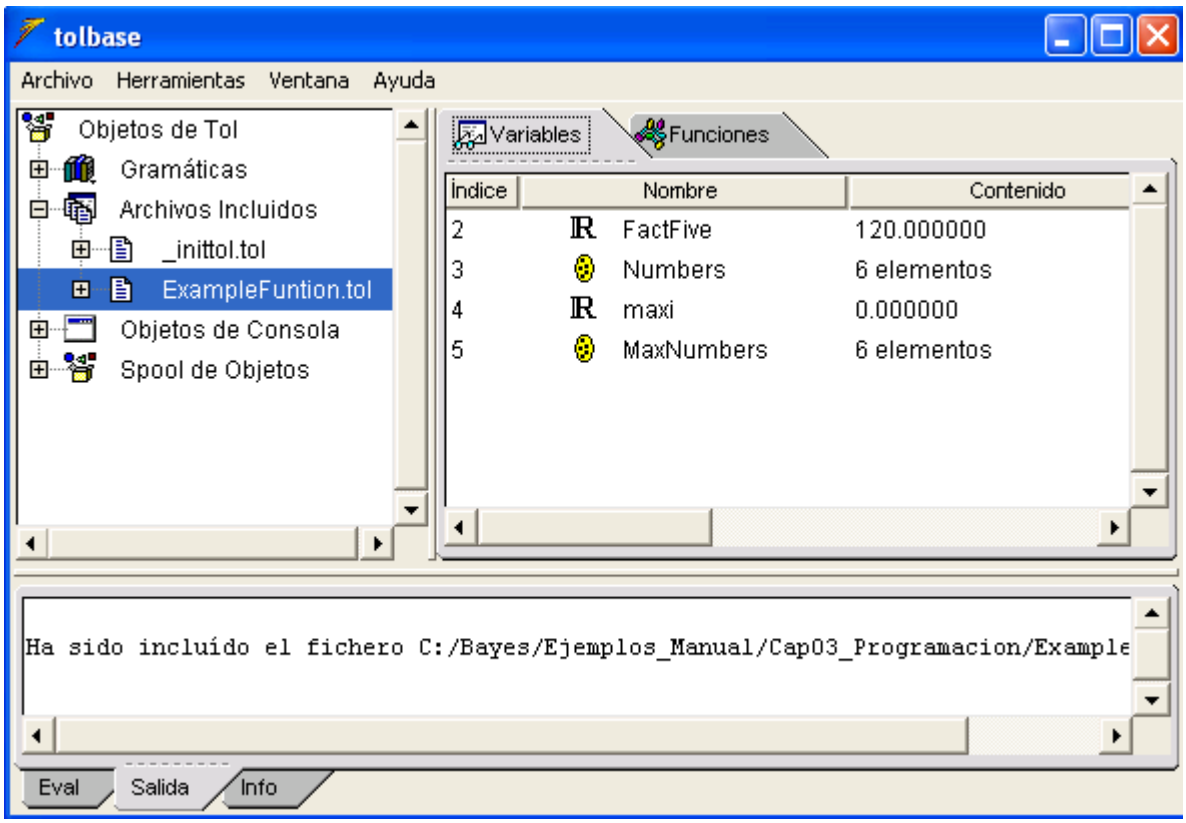


A través del **inspector de objetos** del entorno **Tolbase** se pueden consultar los diferentes objetos creados como resultado de la compilación y ejecución de este fichero **TOL**. Para comprender el funcionamiento del inspector de objetos puede establecerse un paralelismo entre éste y un inspector de archivos de **Windows**:

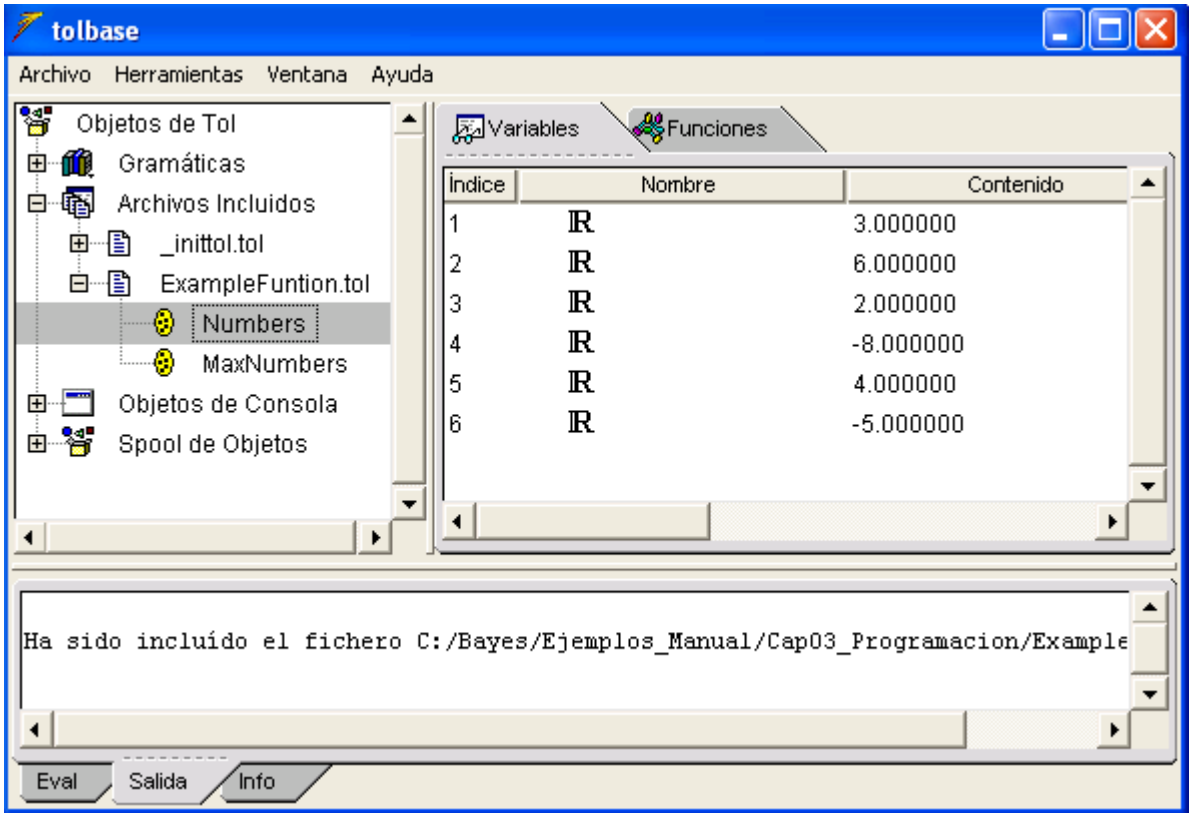
- El **inspector de objetos** muestra el contenido de conjuntos **TOL** como el inspector de archivos muestra el contenido de directorios.

- Para cada objeto dentro de un conjunto se muestran sus propiedades como en el inspector de archivos se muestran las propiedades de los ficheros.

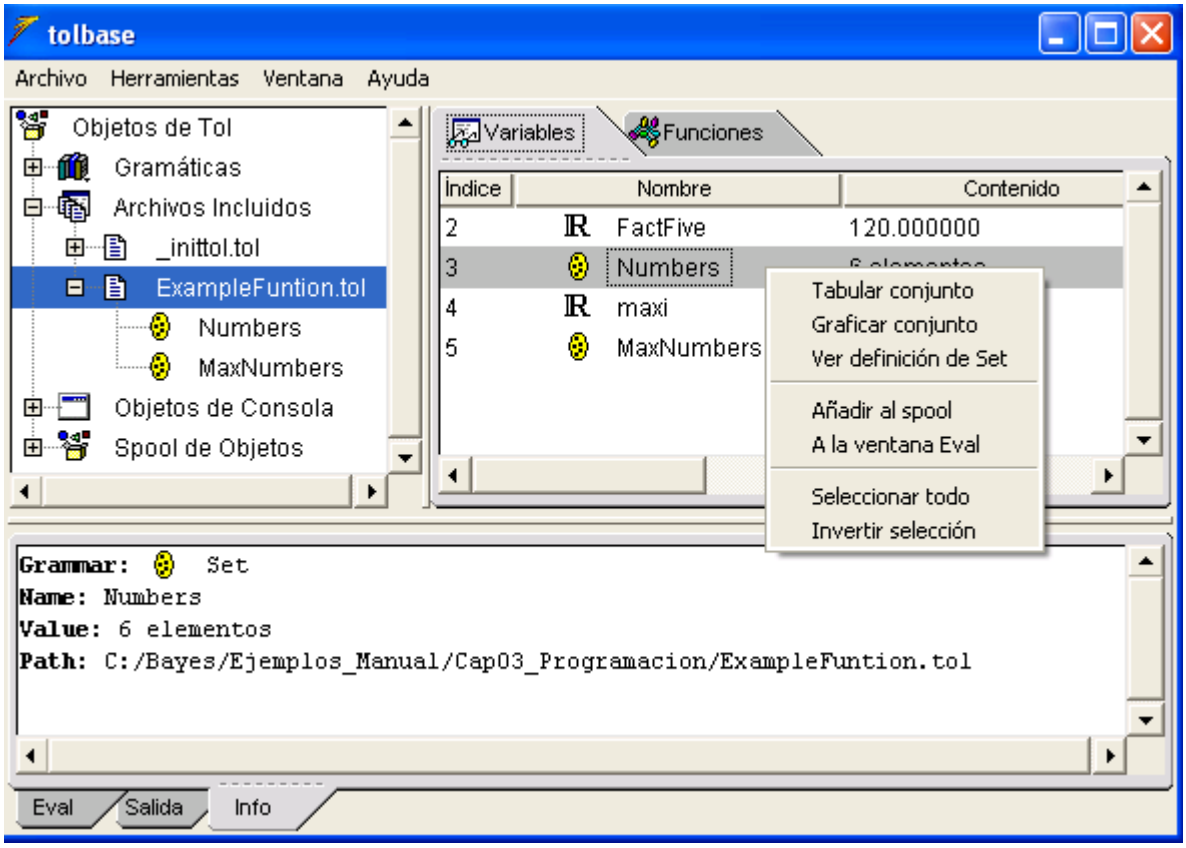
Cada fichero **TOL** se representa como un conjunto que contiene todos los objetos declarados dentro de él. En la siguiente figura se muestran los objetos declarados dentro del fichero `ExampleFunction.tol`:



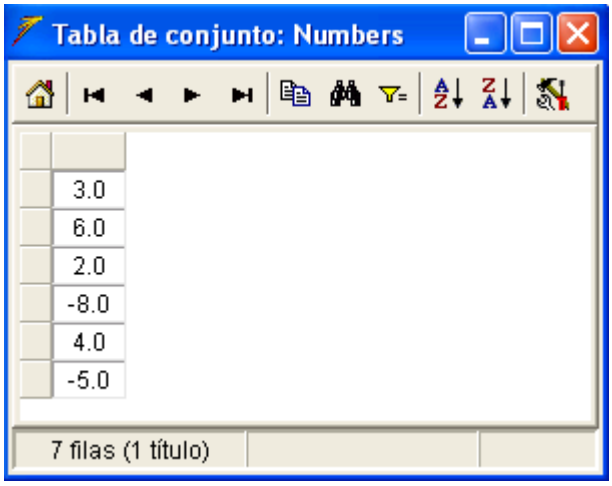
A través del **inspector de objetos** puede observarse, por ejemplo, que la variable denominada `Numbers` representa a un objeto de tipo conjunto (**`Set`**) que contiene seis reales(`3,6,2,-8,4,-5`). Se puede explorar el contenido de dicho conjunto a través del **inspector de objetos**, como se muestra en la siguiente figura:



Utilizando el **inspector de objetos** es posible graficar, tabular, ver la definición, entre otras opciones de las variables creadas. Por ejemplo, se puede construir una tabla con el contenido de un conjunto (`Numbers`). Para ello, pulsamos con el botón derecho del ratón sobre dicho objeto y obtenemos un menú con los métodos aplicables:



En el ejemplo, el conjunto `Numbers` se compone de seis números reales, el resultado de su visualización en forma de tabla se muestra en la siguiente figura.



3.2.1 Funciones Lógicas

Las funciones lógicas retornan cierto (`true`), falso (`false`) o desconocido (`?`). Dada la importancia que tienen las funciones lógicas en la construcción de estructuras de programación, recordamos aquí las más comunes:

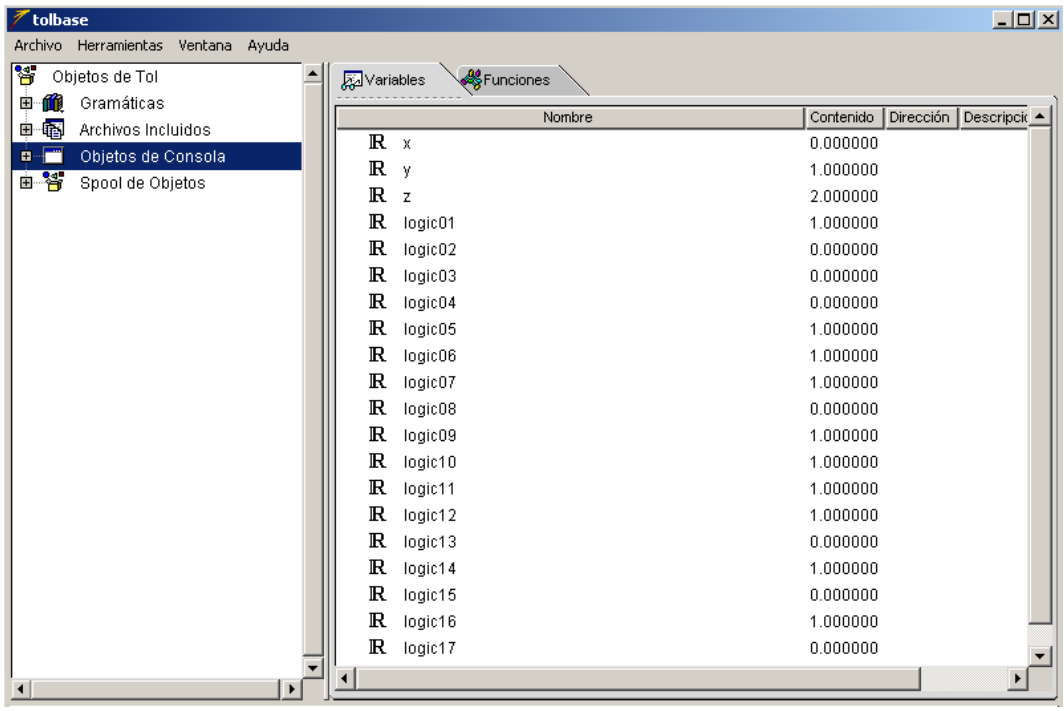
- la negación (**operador** `!` y función `Not()`),
- el y lógico (**operador** `&` y función `And()`) ,
- el o lógico (**operador** `|` y función `Or()`),
- las funciones de comparación como el igual (**operador** `==` y función `Eq()`),
- el diferente (**operador** `!=` y función `NE()`),
- el menor que **operador** `<` y función `LT()`),
- el menor o igual que (**operador** `<=` y función `LE()`),
- el mayor que (**operador** `>` y función `GT()`) ,
- el mayor o igual que (**operador** `>=` y función `GE()`).

En **TOL** se considera que cualquier variable `Real` cuyo valor sea distinto de cero tiene el valor lógico cierto (`true`). No todos los valores lógicos tienen sentido para todos los tipos de variables de **TOL** (para saber que valores lógicos se encuentran disponibles, se pueden visualizar en las funciones existentes para el tipo de variable en cuestión).

Ejemplo Aplicación de funciones lógicas a números reales:

```
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);
```

Los resultados de este ejemplo se pueden consultar a través del inspector de objetos:



3.3 Sentencias de programación

Las **sentencias de programación** permiten realizar un proceso repetidas veces y tomar decisiones. Son las denominados **secuencias de expresión**, **sentencias de selección** y **sentencias de iteración**.

Una **sentencias de iteración** se utiliza para realizar un proceso repetidas veces, que se ejecutará mientras se cumpla unas determinadas condiciones. Las más sencillos se construyen usando las funciones `For()` y `EvalSet().`

Las **sentencias de selección** permiten ejecutar una de entre varias acciones en función del valor de una expresión lógica o relacional. Se tratan de estructuras muy importantes ya que son las encargadas de controlar el **flujo de ejecución** de un programa. Se construyen usando la función `If()` y la función `Case().`

3.3.1 EvalSet()

EvalSet() es el tipo de bucle más útil en lenguaje **TOL**. Su forma general es la siguiente:

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

Explicación: Para el conjunto `NombreConjunto` evalúa elemento por elemento el conjunto de sentencias `code hacer.`

La sentencia `Eval()` puede emplearse también para evaluar una expresión, polinomios y fracciones.

Ejemplo A continuación presentamos un ejemplo utilizando un bucle **`EvalSet()`**. Se define la función `EvalSum01()`, que utiliza un **`EvalSet()`** y acumula la suma en una variable `sum` utilizando el **operador** `:=` de reasignación.

```
////////////////////////////////////////////////////////////////////////////
   Real EvalRandGen(Real seed, Real n)
////////////////////////////////////////////////////////////////////////////
// PURPOSE : Generates n ramdom numbers using the seed.
//           Function implemented using a EvalSet() schema.
//           The numbers are writen 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
};
```

3.3.2 For()

For() es quizás el tipo de bucle más utilizado en la mayoría de los lenguajes de programación. Su forma general es la siguiente:

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

Explicación: Para todo entero `n` desde el valor `inicial` hasta el valor `final` de uno en uno devuelve el conjunto resultados `iteracción en n.`

Donde el tercer argumento de la función **`For()`** es una función sin nombre (tipo **`Code`**) de **`Real`** en **`Real`** que recibe `n` y devuelve el resultado de evaluar la función de tipo **`Code`** en `n`.

Ejemplo En este ejemplo se presenta el ciclo **`For()`** anidado:

```
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
  }
);
```

Traza por la **ventana de mensajes**:

```
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
```

3.3.3 If()

La función de control **`If()`** consta de tres operandos (ternario) y tiene la siguiente forma general:

```
If(Expresión, Expresión_1, Expresión_2)
```

Explicación: Se evalúa `Expresión` y

- Si el resultado es **true** se ejecuta `Expresion_1`
- Si el resultado es **false** se ejecuta `Expresion_2`

Hay que recordar que `Expresión` puede ser simple o compuesta (bloque { ... }).

Ejemplo En este ejemplo se muestra el uso de la función de control **`If()`** simple que retorna el segundo argumento si el primero es cierto y en otro caso, retorna el tercero:

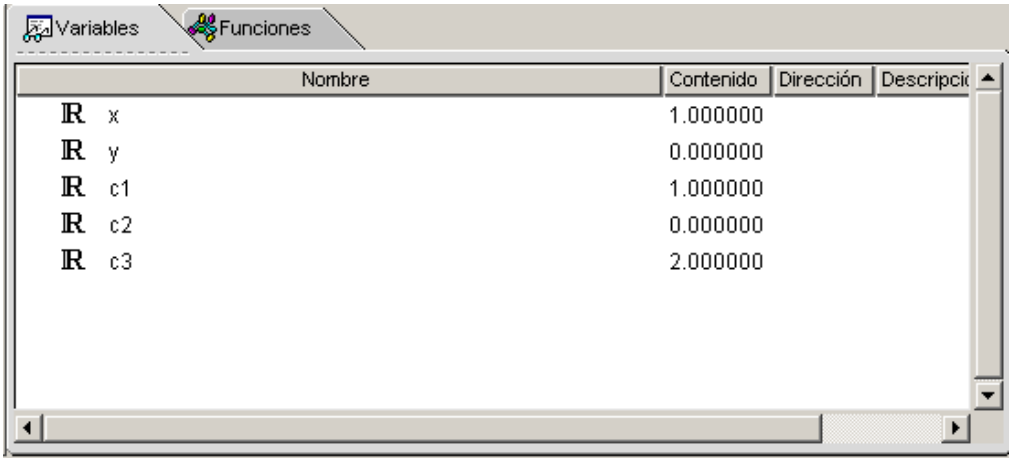
```
Real x = 1;
Real y = 0;
Real c1 = If(x>y,x,y);
Real c2 = If(x<y,x,y);
```

Ejemplo

En este ejemplo se muestra el uso anidado de la función de control **`If()`**

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

En el inspector de objetos, podemos observar los resultados de los dos ejemplos anteriores:



3.3.4 Case()

La función de control **`Case()`** consta de tres o más operandos y tiene la siguiente forma general:

```
Case(Condicion1,  Expresion1, [Condicion2, Expresion2,…])
```

Explicación: Se evalúa `Condicion1` y

- Si el resultado es **true** se ejecuta `Expresion1`
- Si el resultado es **false** se evalúa `Condicion2`
  - Si el resultado es **true** se ejecuta `Expresion2`
  - Si el resultado es **false** se evalúa `Condicion3…`

Hay que recordar que `Expresion` puede ser simple o compuesta (bloque { ... }).

Ejemplo En este ejemplo se muestra el uso de la función de control **`Case()`** para comparar dos números reales:

```
Real x = 1;
Real y = 0;
Real c1 = Case(GT(x,y),x,
               EQ(x,y),3,
               LT(x,y),y);
```

4. Set

En **TOL** se pueden construir conjuntos (**Set**) como colecciones de elementos del mismo o de diferentes tipos.

4.1 Definición de conjuntos

Existen cuatro tipos de conjuntos, conjuntos simples, conjuntos de conjuntos, conjuntos estructurados y conjuntos de conjuntos estructurados (tablas).

4.1.1 Conjuntos simples

A continuación pueden verse, a modo de ejemplo, algunas definiciones de conjuntos.

Ejemplos

- Un conjunto de reales, definido mediante la función **SetOfReal()**,

```
Set  RealSet1 = SetOfReal(5, 1.4, .23, 456, 83.85);
```

- Un conjunto de fechas, definido mediante la función **SetOfDate()**,

```
Set DateSet = SetOfDate(y1995m12d3, y1996m11d15, y1994m1d4, y1994m12d04);
```

- Un conjunto de textos, definido mediante la función **SetOfText()**,

```
Set TextSet = SetOfText("alfa01", "beta01", "afa02", "beta03", "afa04");
```

- Un conjunto de polinomios, definido mediante la función **SetOfPolyn()**,

```
Set PolynSet = SetOfPolyn(3*B^7, 87*B^3+32+4*B^2);
```

- Un conjunto de fracciones polinomiales, definido mediante la función **SetOfRatio()**,

```
Set RationSet = SetOfRatio((3*B^7) / (87*B^3+32+4*B^2), 1 / (1-B));
```

- Un conjunto de conjuntos temporales, definido mediante la función **SetOfTimeSet()**,

```
Set WeekSet = SetOfTimeSet(Daily, Weekly);
```

- Un conjunto de series, definido mediante la función **SetOfSerie()**,

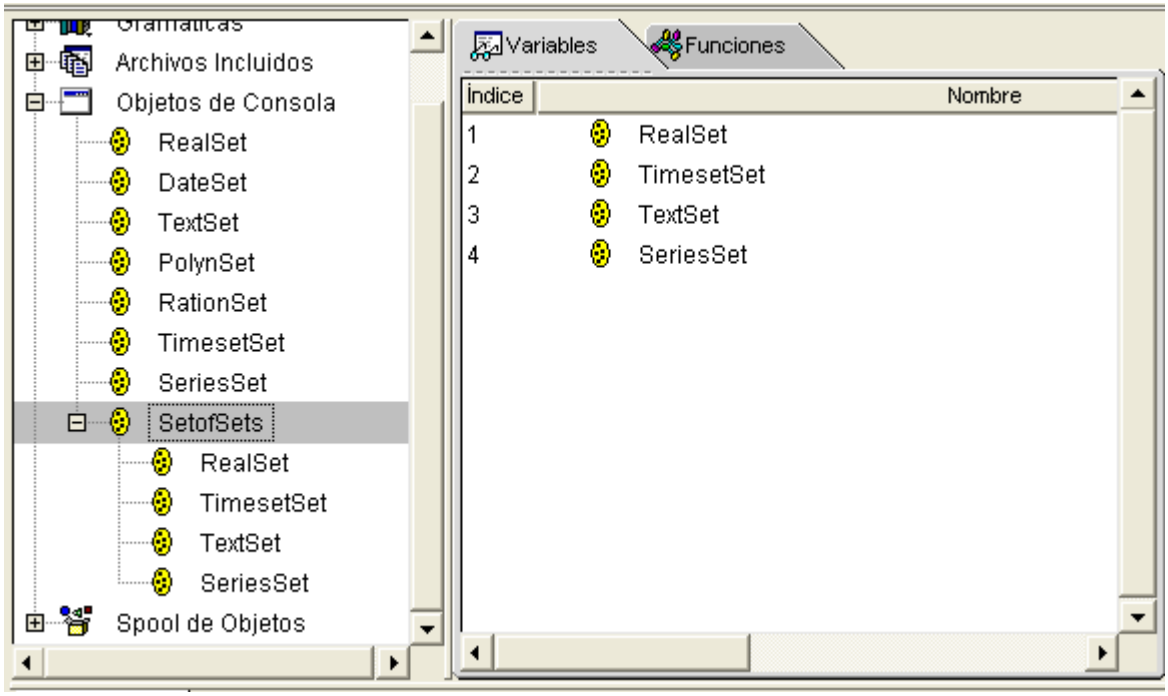
```
Serie S1 = Pulse(y2001,Daily);
Serie S2 = Pulse(y2002, Daily);.
Set SeriesSet = SetOfSerie(S1, S2);
```

4.1.2 Conjuntos de conjuntos

TOL también permite construir un conjunto de conjuntos, utilizando la función **`SetOfSet()`**, por ejemplo, con algunos de los conjuntos definidos anteriormente,

```
Set SetofSets = SetOfSet(RealSet, WeekSet, TextSet, SeriesSet);
```

En la siguiente figura puede observarse, a través del **inspector de objetos**, los conjuntos creados:



4.1.3 Conjuntos estructurados

En **TOL** se pueden declarar estructuras,

Ejemplo

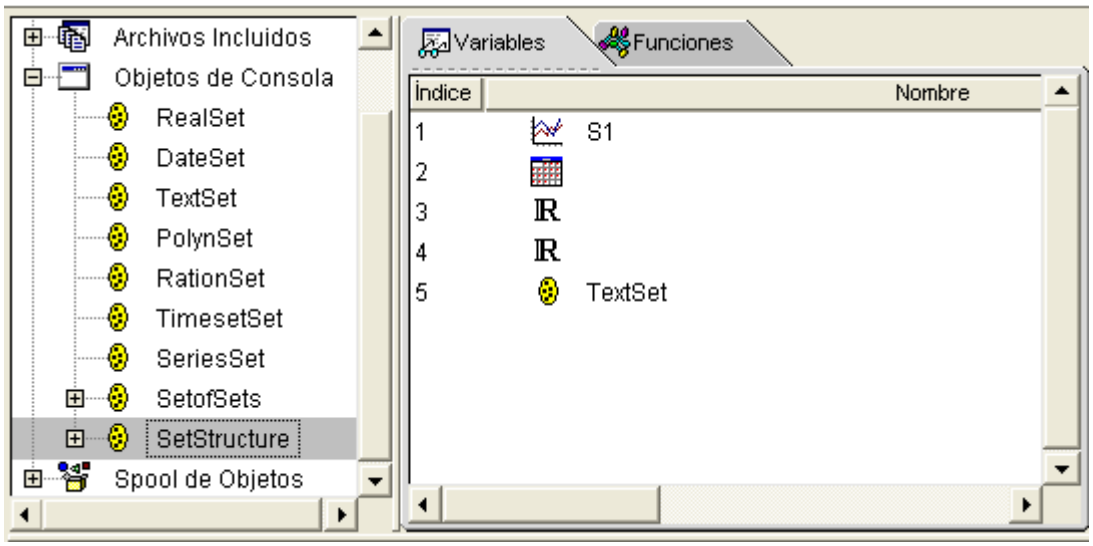
El siguiente comando declara una estructura de conjuntos formados por una serie temporal (**`Serie`**), una fecha (**`Date`**), dos números reales (**`Real`**) y un conjunto (**`Set`**).

```
Struct Structure{Serie Ser, Date Dates, Real Real1, Real Real2, Set Sets};
```

A partir de dicha estructura podríamos definir un conjunto sobre ella, usando

```
Set SetStructure = Structure(S1, y2001, 200, 1, TextSet);
```

En la siguiente figura puede observarse, a través del **inspector de objetos**, esta estructura:



4.1.4 Otras definiciones de conjuntos

También pueden crearse conjuntos mediante otras funciones, por ejemplo,

- La función **`EvalSet()`**, que permite generar un conjunto con el resultado de evaluar un conjunto y aplicar una serie de sentencias sobre el.
- La función **`For()`**, que permite generar un conjunto con el resultado de evaluar una función de parámetro real entre dos enteros dados.
- La función **`Range()`**, que retorna un conjunto de números entre dos dados siguiendo una cierta intervalación,
- La función **`Select()`**, que selecciona un subconjunto de otro conjunto con los elementos que cumplen cierta condición lógica.

Ejemplos

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

La siguiente sentencia define una función de selección que permite seleccionar los números reales mayores

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

utilizando esta función de selección se podrían extraer de un conjunto previamente definido `RealSet` aquellos mayores de dos como

```
Set Choice = Select(RealSet, Selection);
```

Ejemplo

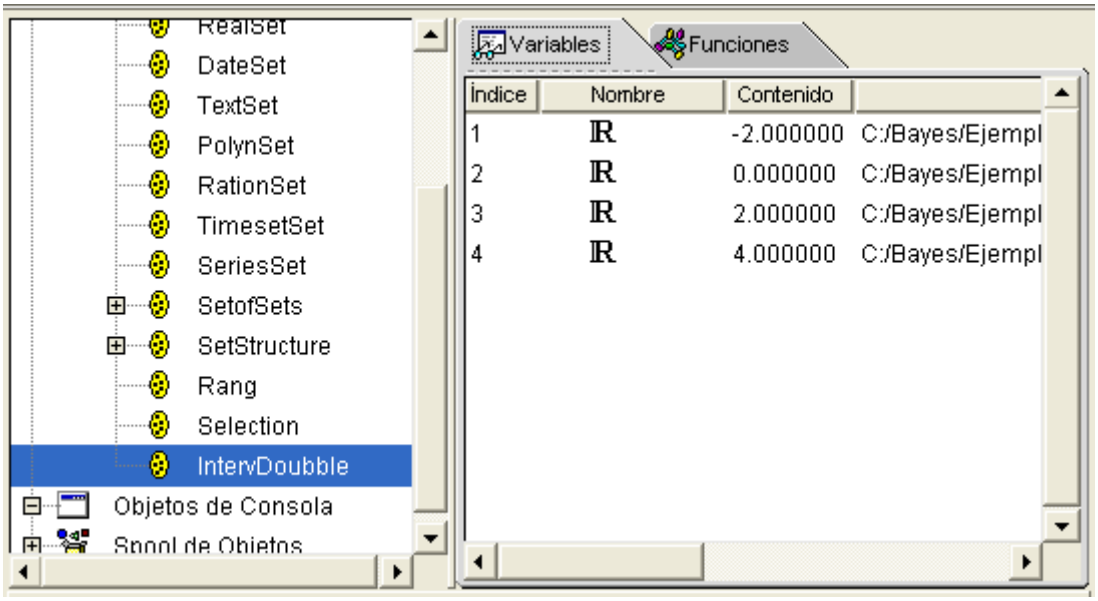
Definamos la función `Doubble` que retorna el doble de un número,

```
Real Doubble(Real x){ 2*x };
```

La siguiente función, retorna el conjunto de números que son el doble de los enteros comprendidos entre el `-1` y el `2`.

```
Set IntervDoubble = For(-1, 2, Doubble);
```

En la siguiente figura podemos observar, a través del **inspector de objetos**, el contenido de este conjunto `IntervDoubble`:



Se pueden construir productos cartesianos, por ejemplo, la expresión

```
Set CartesProdSet = Rang^3;
```

genera el resultado de multiplicar el conjunto `Rang` por si mismo dos veces y la expresión

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

es el producto cartesiano de los tres conjuntos especificados como parámetros.

Además, en **TOL** se pueden ordenar conjuntos dado un cierto criterio de ordenación mediante función **`Sort()`**. El criterio se especifica como una función que compara dos parámetros o argumentos y que:

- retorna `-1` cuando el primer argumento es menor que el segundo,
- retorna `0` cuando ambos elementos se sitúan al mismo nivel según dicho criterio y

- retorna `1` cuando el primer argumento es mayor que el segundo.

Ejemplo Las siguientes funciones permiten ordenar números de menor a mayor y de mayor a menor, respectivamente.

```
Real  Order1(Real x, Real y) { Sign(x-y) };
Real  Order2(Real x, Real y) { Sign(y-x) };
```

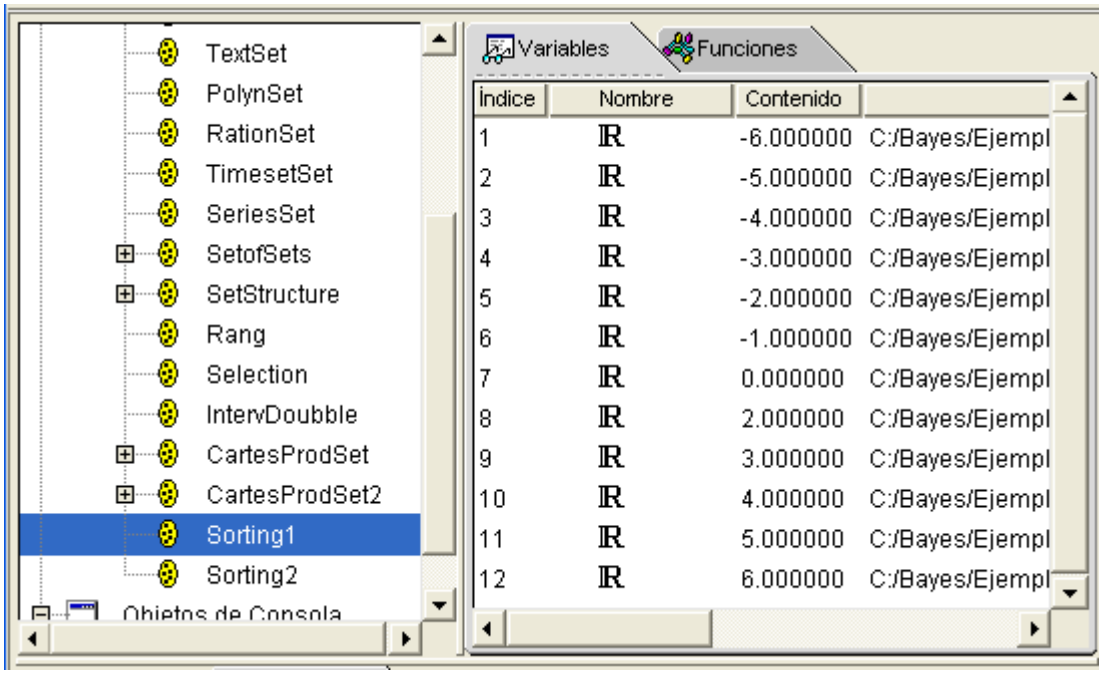
A continuación se muestra como ordenar conjuntos de números reales utilizando **`Sort`**`()` y los dos criterios de ordenación comentados:

```
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);
```

Otra forma de realizar este ejemplo mediante funciones sin nombre:

```
Set SortLessGreat = Sort(SetOfReal(6,2,-1,-6,0,3,-2,-3,4,-4,5,-5),
                         Real(Real x, Real y){Sign(x-y)});
Set SortGreatLess = Sort(SetOfReal(6,2,-1,-6,0,3,-2,-3,4,-4,5,-5),
                         Real(Real x, Real y){Sign(y-x)});
```

En la siguiente figura puede observarse, a través del **inspector de objetos**, el contenido del primer conjunto `Increase` ordenado de menor a mayor:



4.2 Operaciones con conjuntos

Existen otras muchas operaciones en **TOL** sobre conjuntos. Hemos de tener especial cuidado con el empleo de las operaciones, ya que los conjuntos en **TOL** tienen una doble función: se pueden emplear como conjuntos ó como vectores.

Operaciones de conjuntos cuando se emplean como conjuntos:

- la unión de dos conjuntos utilizando el **operador** `+`

```
Set Summ  = RealSet + DateSet;
```

- la diferencia entre dos conjuntos utilizando el **operador** `-`

```
Set Substract = RealSet - SetOfReal(1.5, .23, 83.85);
```

- la intersección de conjuntos utilizando el **operador** `*`

```
Set Intersection = RealSet * Range(2,5,1);
```

- la extracción de los elementos no repetidos utilizando la función `Unique()`

```
Set Elements = Unique(Choice);
```

Operaciones de conjuntos cuando se emplean como vectores:

- la concatenación de conjuntos utilizando el **operador** `<<`

```
Set Appendd   = RealSet << SetOfReal(1.5,.23,83.85);
```

5. TimeSet

Es un tipo de datos con fechados básicos (`Y(x) = Year, M(x) = Month, D(x) = Day, H(x) = Hour, Mi(x) = Minute, S(x) = Second`), operaciones básicas (`Unión, Intersección` y `Diferencia`), que permiten generar nuevos conjuntos temporales, como `WD` (`WeekDay`) y otros. El lenguaje **TOL** está basado en una representación infinita del tiempo. El hecho de contar con una representación del tiempo tiene importantes implicaciones, ya que:

- no sólo es posible manipular en TOL series temporales con periodicidades habituales diarias, semanales, decenales, mensuales, bimensuales, anuales, etc.,
- sino que también es posible la descripción de periodicidades irregulares, por ejemplo, cierto o ciertos días de la semana, ciertos meses o ciertos días del mes, combinaciones de días concretos de la semana por meses del año, días concretos del año tanto sueltos como por intervalos, periodos de Pascua, etc. que pueden definirse de forma ad-hoc para cada problema en concreto.

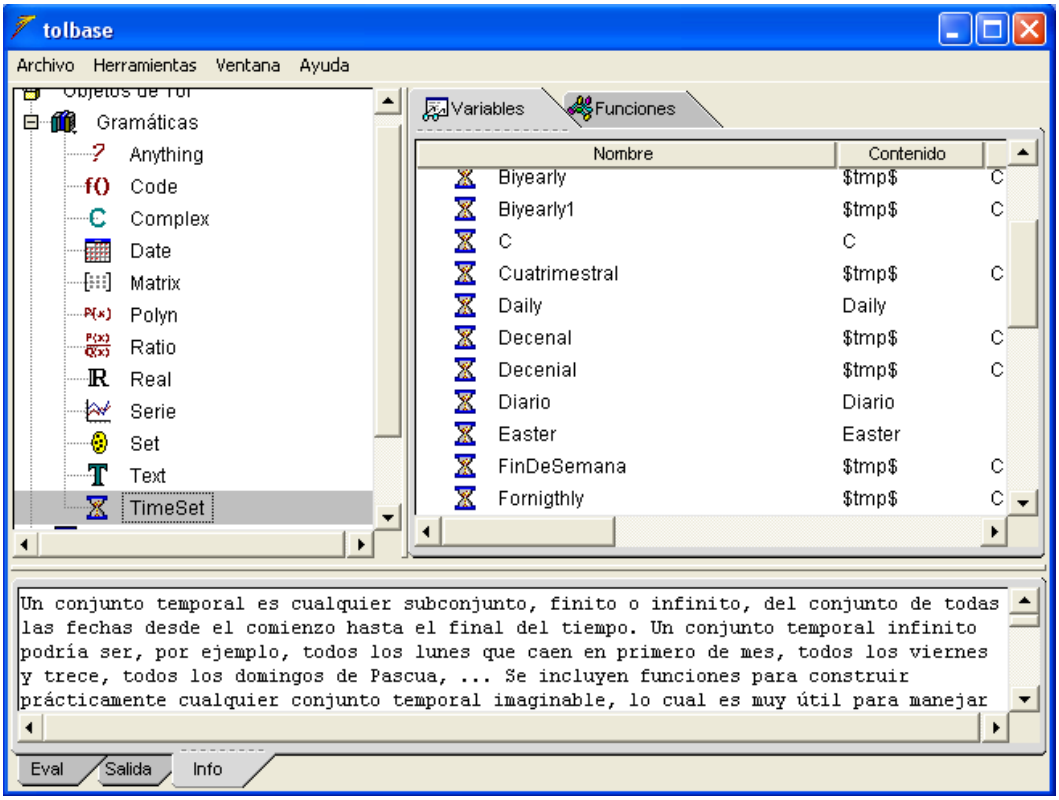
El tiempo es una variable continua que para trabajar con ella, es necesario partirla en intervalos. A esta partición del tiempo en intervalos se la conoce en ***Bayes*** con el nombre de **fechado**.

Con el tipo **`TimeSet`** encontramos dos tipos de objetos, los conjuntos temporales (finitos) por un lado y los fechados (infinitos) por otro.

Para crear una serie temporal en **TOL** se debe definir sobre fechado. La necesidad de tener un fechado viene dada por el estudio:

- Cada cuánto tiempo se repiten las mismas condiciones básicas para la ocurrencia de los datos. Ejemplo: para la serie Ventas diarias: cada lunes tenemos las mismas condiciones, espacio (en días) entre un lunes y el siguiente: 7 días que corresponde con el periodo de la serie.
- Si la recogida es diaria aplicaremos el fechado Diario, si es todas las semanas el Semanal, etc. pero si tenemos un caso particular de recogida de datos irregular y queremos crear un modelo ARIMA que recoja el comportamiento de esas ventas hemos de crear un fechado particular, para contar con un requisito imprescindible de esta clase de modelos que es el periodo.

Al iniciar **TOL** contamos con una serie de fechados predefinidos, como Diario, Semanal, Mensual, Easter (domingo de Pascua), etc. Para ver que fechados están ya definidos vamos a Gramáticas - TimeSet como muestra la figura:



Por ejemplo, el conjunto temporal de todos los Lunes (`Semanal, WD(1)`) es un fechado, pero el conjunto temporal de los lunes del año 2000 (`WD(1)*Y(2000)`) no es un fechado.

Podemos caracterizar un fechado de la siguiente manera:

- Para una fecha perteneciente a un fechado siempre podemos calcular su sucesor dentro de ese fechado, cosa que no ocurre en el conjunto temporal.
- El fechado es un conjunto temporal infinito.

Dentro del álgebra del tiempo se distingue entre: los conjuntos temporales y los operadores para tratarlos. Dentro de los conjuntos temporales se puede distinguir entre:

- Elementos primitivos, predefinidos según la representación diaria.

Ejemplo

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

- Conjuntos temporales derivados de elementos primitivos y la aplicación de operadores sobre ellos.

Ejemplo

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

- Y respecto a los operadores y funciones:
  - Operadores: unión (+), diferencia (-) e intersección (*).
  - Funciones: la función sucesor (**Succ()**) y sus derivadas (**Range(), Periodic()**).

5.1 Representación Diaria

Se define los conjuntos **C** y **W** que representan el conjunto universal (por tanto, de todos los días) y el conjunto vacío, respectivamente. Se consideran también las siguientes familias de conjuntos temporales:

- **D(i)**, i=1,..., 31, donde **D(i)** representa el conjunto de todos los días i-ésimos de cualquier mes.

Ejemplo: **D(1)** es el conjunto temporal de todos los días Uno de cada Mes.

- **WD(i)**, i=1,..., 7, donde **WD(i)** es el conjunto de días que pertenecen al día i-ésimo de la semana.

Ejemplo: **WD(1)** es el conjunto temporal de todos los Lunes.

- **M(i)**, i=1,..., 12, donde **M(i)** es el conjunto de días que pertenecen al mes i-ésimo del año.

Ejemplo: **M(1)** es el conjunto temporal de todos los días de Enero.

- **Y(i)**, que representa el año i-ésimo en el actual calendario gregoriano.

Ejemplo: **Y(2004)** es el conjunto temporal de todos los días del año 2004.

- **Los intervalos cerrados,** que contienen todos los días entre dos extremos. Cuando el extremo izquierdo es **TheBegin** se indica falta de acotación por la izquierda, mientras que cuando el extremo derecho es **TheEnd** se indica falta de acotación por la derecha. Así, **In(TheBegin, TheEnd) = C** y si el extremo inferior es mayor que el extremo superior el conjunto resultante es **W**.

Ejemplo

```
// Set of all days on the year between Day 1 of January 1999 and 15 of
// January 1999
TimeSet Fifthteen99 = In(y1999m01d01,y1999m01d15);
```

Dentro de esta representación se puede utilizar unidades de tiempo más pequeñas, la mínima unidad de tiempo representada en **TOL** es el segundo, también están el minuto y la hora cuya representación es:

- **S(i)**, i=0,..., 59, donde **S(i)** representa el conjunto temporal de todas las fechas en el segundo i-ésimo.

Ejemplo: `S(0)` es el conjunto temporal de todos los primeros segundos de un minuto.

- `Mi(i)`, i=0,..., 59, donde `Mi(i)` representa el conjunto temporal de todas las fechas en el minuto i-ésimo.

Ejemplo: `Mi(0)` es el conjunto temporal de todos los primeros minutos de la hora.

- `H(i)`, i=0,..., 23, donde `H(i)` representa el conjunto temporal de todas las fechas en la hora i-ésima.

Ejemplo: `H(0)` es el conjunto temporal de la primera hora del día.

5.2 Operaciones algebraicas

Las operaciones algebraicas básicas para la construcción de nuevos conjuntos temporales son:

- **Unión** (**operador** `+`): Si `A` y `B` son dos conjuntos temporales, entonces `A + B` se define como: { d / d pertenece a `A` o d pertenece a `B`}.

Ejemplo

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

- **Intersección** (**operador** `*`): Si `A` y `B` son conjuntos temporales, entonces `A * B` se define como: { d / d pertenece a `A` y d pertenece a `B`}.

Ejemplo

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

- **Diferencia** (**operador** -): Si `A` y `B` son conjuntos temporales, entonces `A - B` se define como: { d / d pertenece a `A` y d no pertenece a `B`}.

Ejemplo

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

5.3 Funciones

En **TOL** existen funciones predefinidas sobre temporales. Los conjuntos temporales constituyen un álgebra y se pueden obtener nuevos conjuntos a través de operadores de traslación.

5.3.1 Sucesor

Se define el conjunto **`Succ`**`(Ct, n, CtUnidades)` como el conjunto obtenido calculando el sucesor enésimo a cada elemento de Ct en las unidades dadas.

Ejemplo

```
TimeSet SeconDayMonth = Succ(D(1),1,Daily); // TimeSet of second day of each month.
```

5.3.2 Rango

Se define la función **Range**(Ct, n, m, CtUnidades) como:

$$Range(Ct,n,m,unidades) = \bigcup_{i=n}^{m} Succ(Ct,i,unidades)$$

Ejemplo

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

5.3.3 Intervalos periódicos

Se define la función **Periodic**(fecha, n, CtUnidades) como el conjunto compuesto por una fecha y sus traslaciones a un número de fechas múltiplo de n dentro del fechado de unidades dado.

Ejemplo

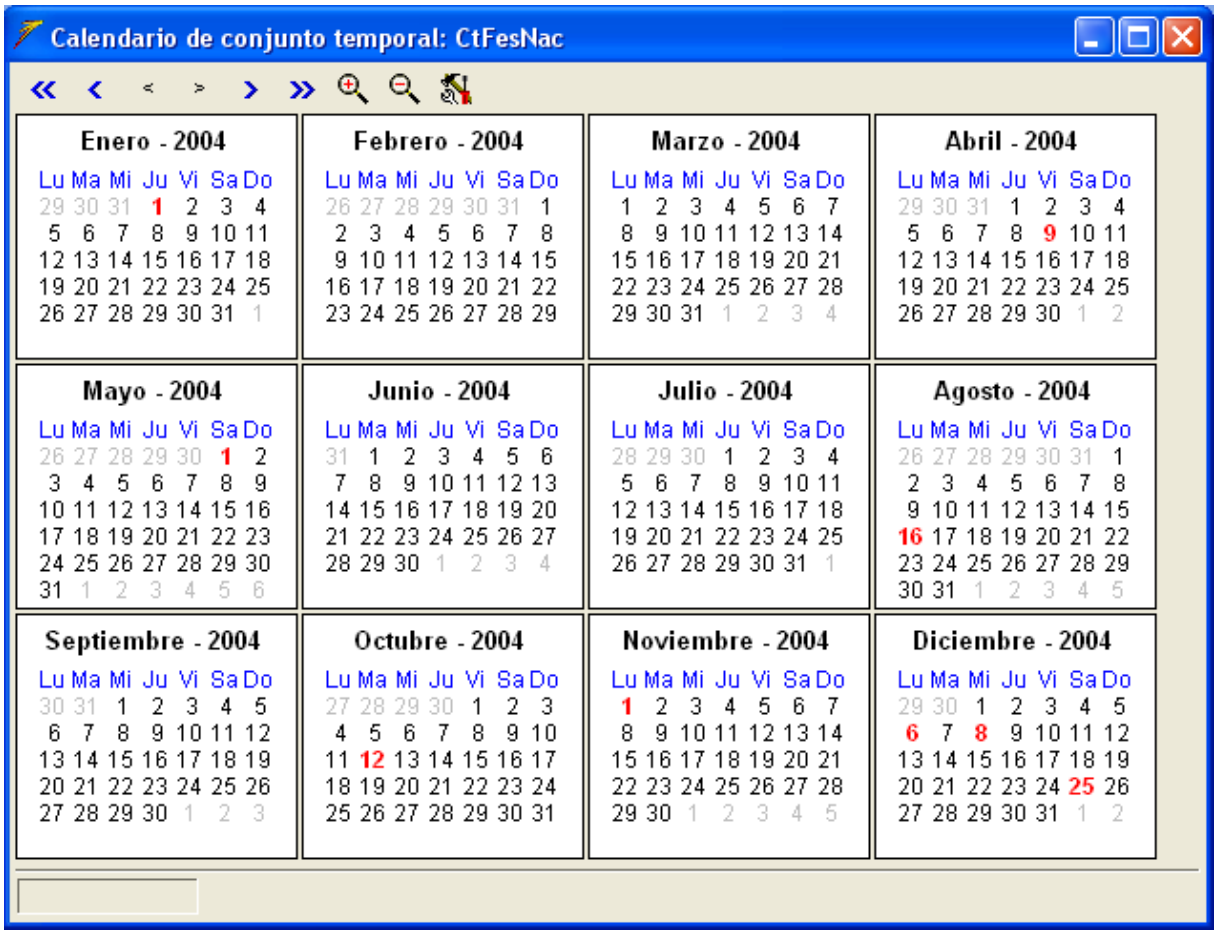
```
// TimeSet of the ten successors of the first day of January 1999.
TimeSet TenDaysAfter =  Periodic(y1999m01d01, 10,Daily);
```

Ejemplo.

El siguiente ejemplo muestra el conjunto temporal de los festivos nacionales y cuando alguno de estos festivos cae en Domingo, se traslada al Lunes. Se incluyen los siguientes:

```
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;
```

El resultado de este ejemplo para el año 2004 se muestra en la siguiente figura:



Ejemplo. Conjuntos temporales de Lunes a Viernes y Laborables..

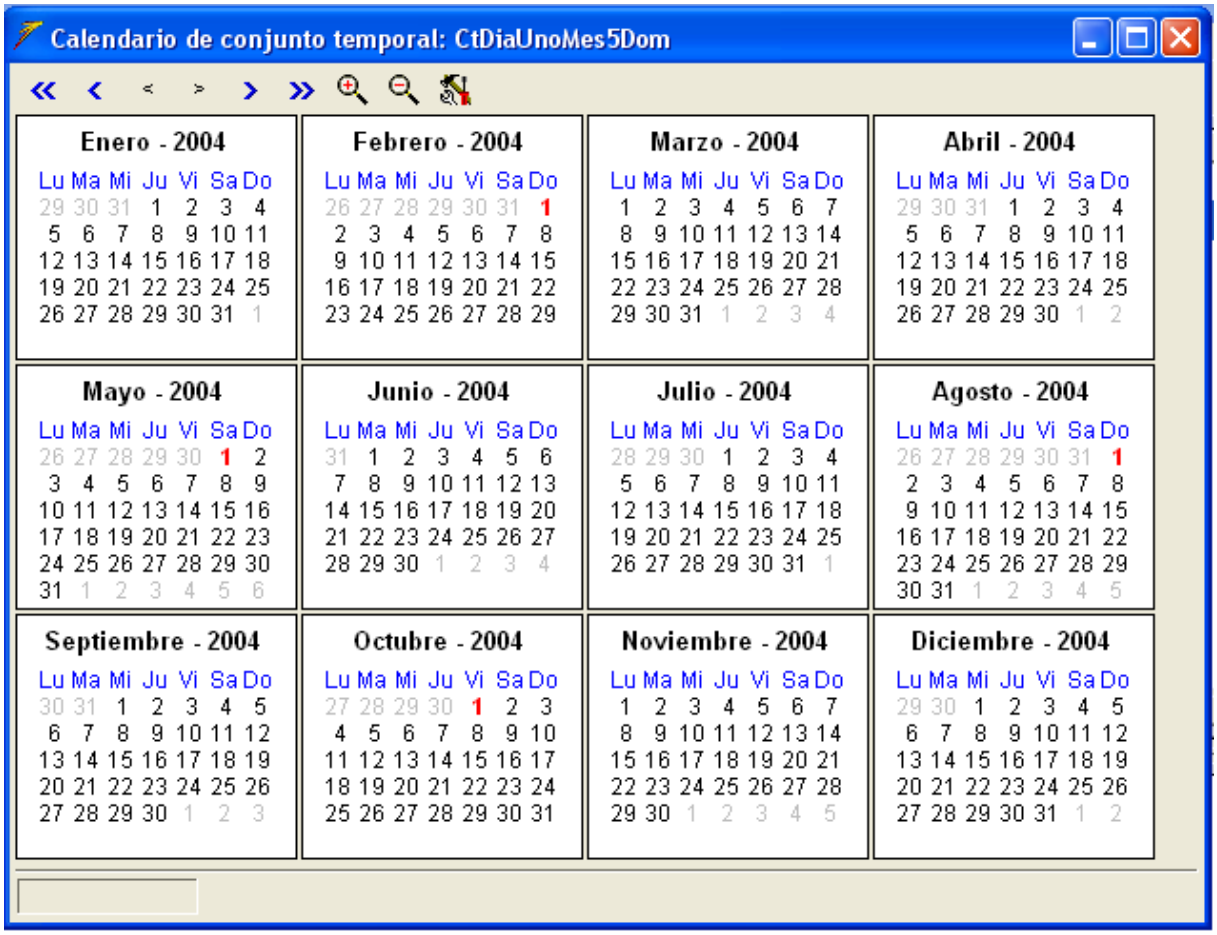
```
TimeSet CtMonFri = C - WD(6) - WD(7);    // TimeSet Mondays to Fridays
TimeSet CtLabour  = CtMonFri - CtNatHol; // TimeSet labour days
```

Ejemplo. Meses que contienen 5 Domingos. En este caso se señala los días Uno de estos meses:

```
TimeSet CtFifthSunMonth = (D(29)+D(30)+D(31)) * WD(7);
TimeSet CtFirstDayMonth5Sun = Succ(CtFifthSunMonth, -1, D(1));
```

El resultado de este ejemplo para el año 2004 se muestra en la siguiente figura:

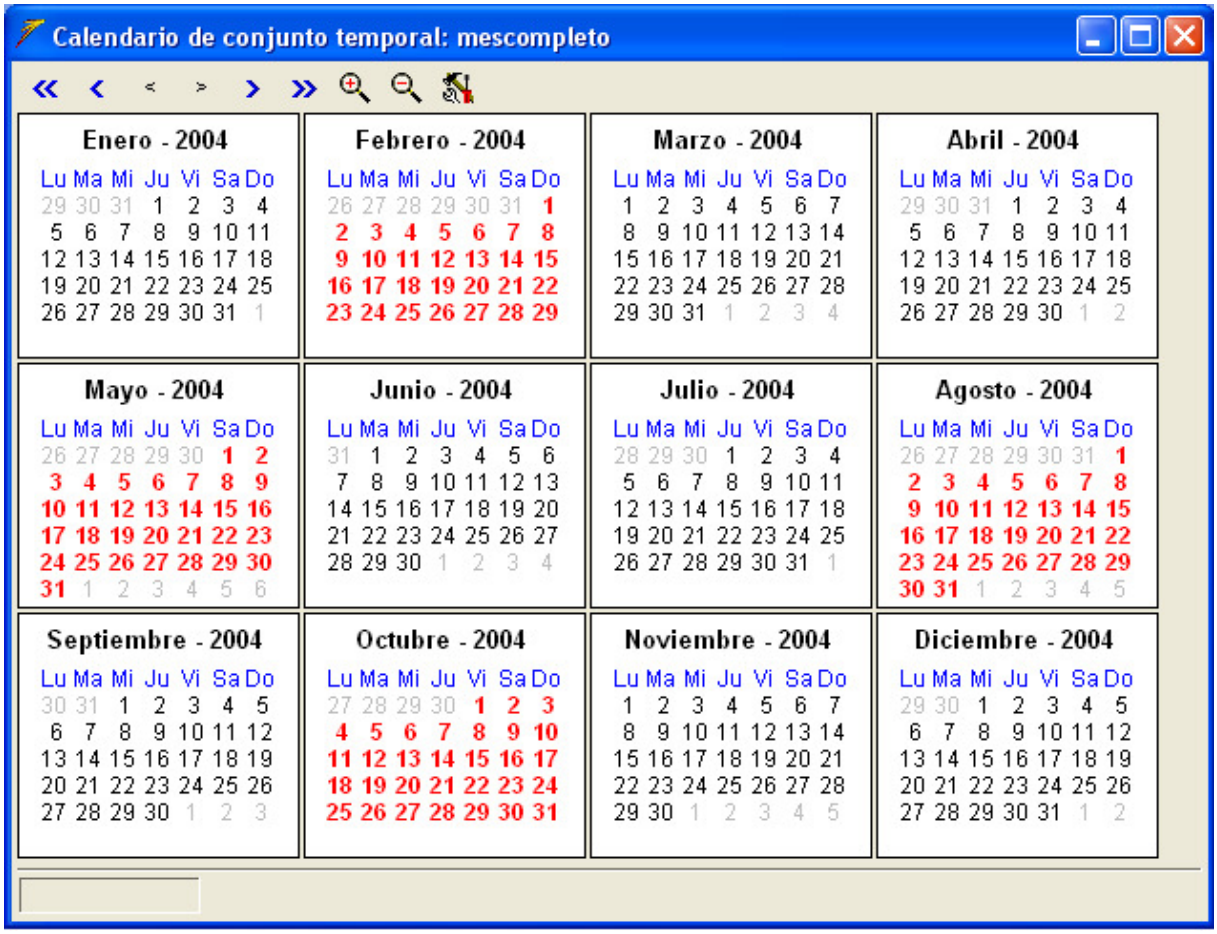
Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636



Para obtener el conjunto temporal correspondiente a todos los días del mes, de los meses de 5 domingos, podríamos definir los siguientes conjuntos:

```
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));
```

El resultado para el año 2004 se muestra en la siguiente figura:

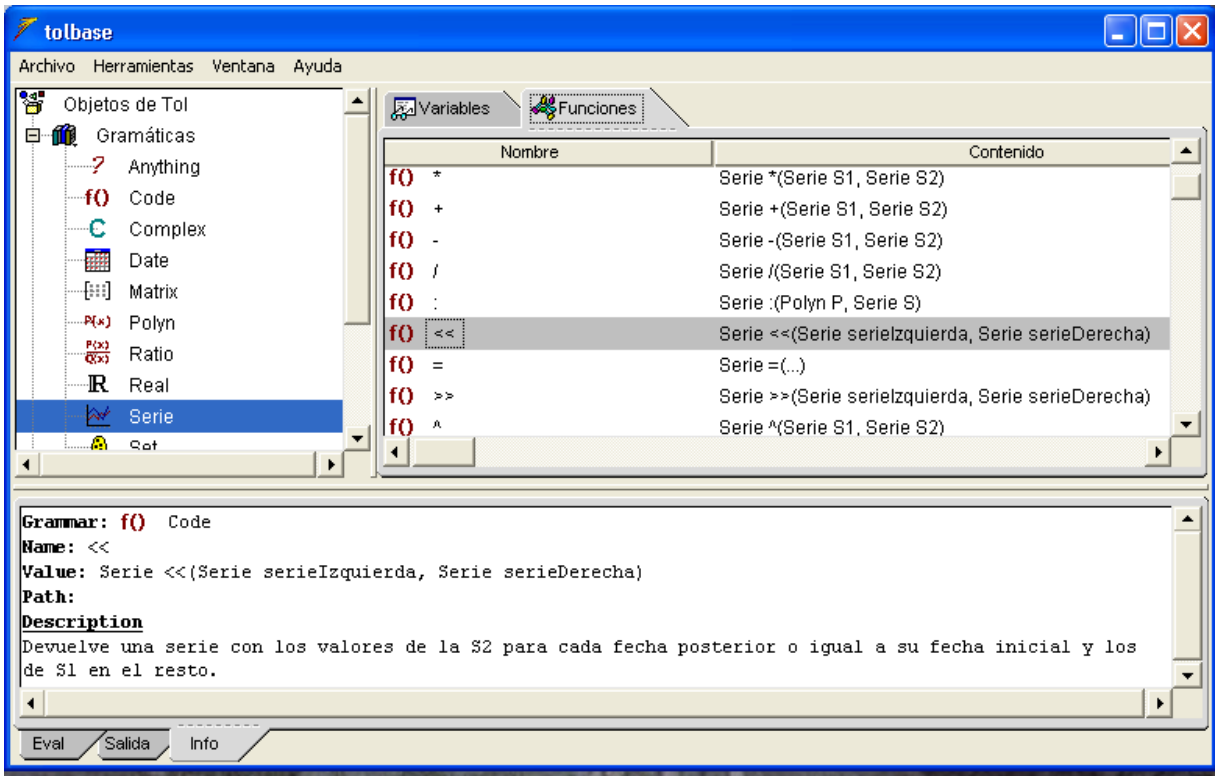

Calendario de conjunto temporal: mescompleto
Enero - 2004
Lu Ma Mi Ju Vi Sa Do
29 30 31 1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31 1
Febrero - 2004
Lu Ma Mi Ju Vi Sa Do
26 27 28 29 30 31 1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
Marzo - 2004
Lu Ma Mi Ju Vi Sa Do
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 1 2 3 4
Abril - 2004
Lu Ma Mi Ju Vi Sa Do
29 30 31 1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 1 2
Mayo - 2004
Lu Ma Mi Ju Vi Sa Do
26 27 28 29 30 1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31 1 2 3 4 5 6
Junio - 2004
Lu Ma Mi Ju Vi Sa Do
31 1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 1 2 3 4
Julio - 2004
Lu Ma Mi Ju Vi Sa Do
28 29 30 1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31 1
Agosto - 2004
Lu Ma Mi Ju Vi Sa Do
26 27 28 29 30 31 1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31 1 2 3 4 5
Septiembre - 2004
Lu Ma Mi Ju Vi Sa Do
30 31 1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 1 2 3
Octubre - 2004
Lu Ma Mi Ju Vi Sa Do
27 28 29 30 1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
Noviembre - 2004
Lu Ma Mi Ju Vi Sa Do
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 1 2 3 4 5
Diciembre - 2004
Lu Ma Mi Ju Vi Sa Do
29 30 1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31 1 2

6. Series

TOL permite el tratamiento y la manipulación algebraica de series temporales, para realizar operaciones, gráficos, estimaciones y previsión de datos. . Se ha de tener en cuenta que las estadísticas disponibles en **TOL** para series temporales requieren de la especificación del instante de tiempo al cual se refiere cada valor concreto. Ese instante puede venir representado por una variable TimeSet.

6.1 Operaciones, funciones y polinomios

TOL posee todas las funciones aritméticas básicas como la suma, resta, multiplicación, división, concatenaciones de series, funciones estadísticas (**AvrS()**, **StDsS()**, **MaxS()**), generación de series aleatorias (**Rand()** y **Gaussian()**); funciones trigonométricas y logarítmicas, funciones lógicas y de comparación, otro tipo de transformaciones (**Exp()**, **Pow()**, **Sqrt()**,...). Todas ellas, con su descripción y argumentos, pueden localizarse de la siguiente manera:



La siguiente tabla resume algunos de los estadísticos más relevantes como funciones de series temporales en el espacio de los números reales:

Nombre	Descripción
`AsimetryS()`	Retorna el coeficiente de asimetría de una serie.
`AvrS()`	Retorna la media de los valores de una serie.
`CountS()`	Retorna el número de valores de una serie.
`FirstS()`	Retorna el primer valor de una serie.
`KurtosisS()`	Retorna el coeficiente de curtosis de una serie.
`LastS()`	Retorna el último valor de una serie.
`MaxS()`	Retorna el máximo valor de una serie.
`MedianS()`	Retorna la mediana de una serie.
`MinS()`	Retorna el mínimo valor de una serie.
`StDsS()`	Retorna la desviación típica de una serie.
`SumS()`	Retorna la suma de los valores de una serie.
`VarS()`	Retorna la varianza de una serie.

Estos estadísticos se emplean en series temporales finitas. En caso de tener una serie infinita, se debe especificar entre qué par de fechas se desea calcular el estadístico.

6.1.1 Operaciones entre series

Para ver el funcionamiento de algunas de las funciones, vamos a construir un par de series aleatoriamente:

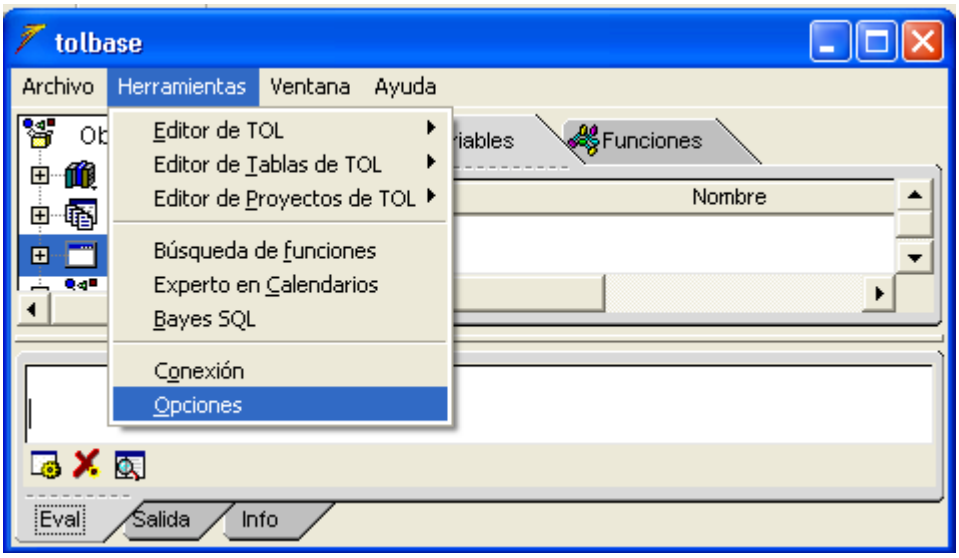
```
Serie s1 = Gaussian(1,0.1,Daily);
Serie s2 = Gaussian(-1,0.5,Daily);
```

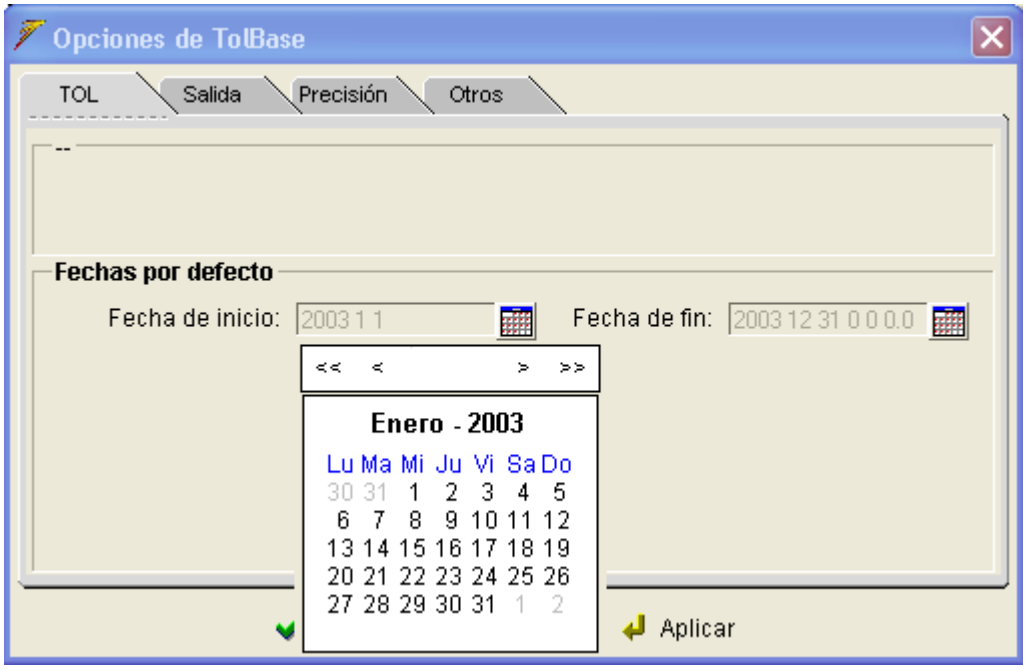
Observaciones:

- Para hacer operaciones entre dos series estas deben estar en el mismo fechado pero no tienen porque estar definidas entre las mismas fechas. Las operaciones se llevarán a cabo solo en las fechas que tengan en común.

- La series generadas mediante `Gaussian()` y `Rand()` quedarán definidas entre las fechas que tenga **TOL** por defecto. Estas pueden modificarse desde **Herramientas Opciones** y seleccionando las fechas que nos interesan pinchando en los calendarios  que aparecen junto a **Fecha de inicio** y **Fecha de fin**, como se muestra a continuación

Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636





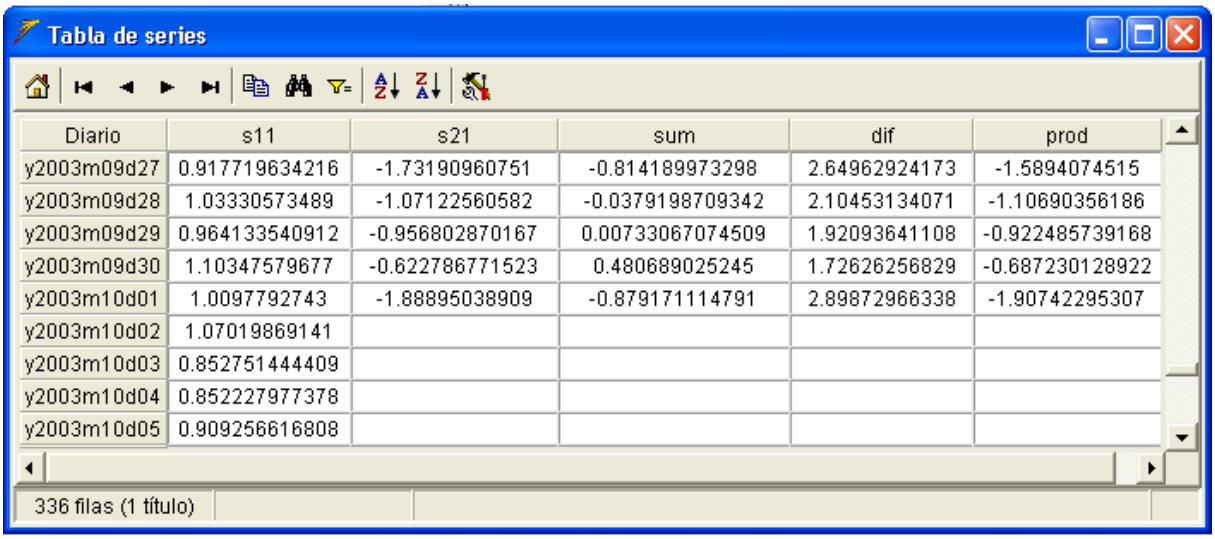
- Las series definidas como se acaba de describir no se calculan hasta que no se delimitan entre dos fechas. Para ello, empleamos el comando **SubSer()**, para acotarla entre dos fechas.

Ejemplo

```
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;
```

La siguiente tabla muestra que las operaciones se realizan componente a componente y sólo entre aquellas fechas en las que ambas series están definidas, es decir:

- La serie `add` comienza en el máximo{Enero 2003, Enero 2003} y finaliza en el mínimo{Diciembre 2003,Octubre 2003}. Análogamente sucede con las series `dif` y `prod`.



Diario	s11	s21	sum	dif	prod
y2003m09d27	0.917719634216	-1.73190960751	-0.814189973298	2.64962924173	-1.5894074515
y2003m09d28	1.03330573489	-1.07122560582	-0.0379198709342	2.10453134071	-1.10690356186
y2003m09d29	0.964133540912	-0.956802870167	0.00733067074509	1.92093641108	-0.922485739168
y2003m09d30	1.10347579677	-0.622786771523	0.480689025245	1.72626256829	-0.687230128922
y2003m10d01	1.0097792743	-1.88895038909	-0.879171114791	2.89872966338	-1.90742295307
y2003m10d02	1.07019869141				
y2003m10d03	0.852751444409				
y2003m10d04	0.852227977378				
y2003m10d05	0.909256616808				

336 filas (1 título)

Entre algunas de las funciones que se pueden utilizar entre series caben también destacar:

Suma de una serie de series:

```
Serie SummSeries = Sum(s1,s2);
```

Producto de una serie de series:

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

Potencia de serie por número:

```
Serie PowerSeries = s1**3;
```

Potencia de serie por serie:

```
Serie PowerSerSer  = s1**s2;
Serie PowerSerSer2 = s1^s2;      // ** is equivalent to ^
Serie PowerSerSer3 = Pow(s1,s2); // Pow is equivalent to previous ones
```

Máximo de series:

```
Serie MaxSerie = Max(s1,s2);
```

Mínimo de series:

```
Serie MinSerie = Min(s1,s2);
```

Redondeo:

```
Serie Rounded = Round(s1);
```

Truncamiento:

```
Serie Trunc = Floor(s2);
```

6.1.2 Funciones para series

TOL suministra funciones de cambio de fechado, concatenación de series, operaciones estadísticas, transformaciones trigonométricas, etc.

Las funciones de series contienen en sus argumentos fechas o conjuntos temporales además de series y el resultado de aplicarles una función es una nueva serie. Definiremos las siguientes funciones:

SubSer(Serie, Date, Date) permite restringir los datos conocidos de una serie a los datos que se conozcan de ella comprendidos entre dos fechas dadas. El primer argumento es la serie que se desea restringir, el segundo y tercer argumento son las fechas inicial y final de la serie resultante.

Ejemplo

```
// 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) concatena las dos series de los argumentos respecto de la fecha dada. La serie resultante toma los valores de la primera serie de los argumentos si sus fechas son menores o iguales que la fecha dada, y los valores de la segunda serie para el resto de las fechas.

Ejemplo

```
// We append both series from June 2005
Serie Appendd  = Concat(StepMay, PulseAugust,y2005m06);
```

<<(Serie, Serie) devuelve una serie cuyos valores son valores de la segunda serie por cada fecha mayor o igual que la última fecha del fechado de la primera serie, o valores de la primera serie en caso contrario.

Ejemplo

```
// We append both series:
Serie Appendd2   = StepMay<<PulseAugust;
```

`>>(Serie, Serie)` devuelve una serie cuyos valores son valores de la primera serie por cada fecha menor o igual que la primera fecha del fechado de la segunda serie, o valores de la segunda serie en caso contrario.

Ejemplo

```
// We append both series:
Serie Appendd3   = StepMay>>PulseAugust;
```

`DatCh(Serie, TimeSet, Code)` devuelve una serie con el fechado indicado. La transformación de los valores originales se indica o bien por un estadístico o bien por un conjunto de sentencias. Tiene como restricción que el nuevo fechado tiene que ser armónico con el fechado original de la serie. Esto significa que cada fecha del nuevo fechado tiene que contener un número entero de fecha del antiguo, este número no tiene que ser siquiera constante.

Ejemplo

Sea S una serie en fechado diario, **`DatCh`**`(S, Semanal, AvrS)` devuelve una serie con fechado semanal y cuyos valores son la media de los valores de cada semana.

6.1.3 Polinomios

TOL permite traslaciones temporales a través del operador de retardo **Backward,** **`B`** y del operador de adelanto **Forward** **`F`**. El **operador** **`:`** permite aplicar polinomios a series temporales retornando una serie que viene determinada por medio del polinomio y de la serie original.

- El **operador** **`B`** de retardo (Backward) sobre una serie temporal es una nueva serie temporal desplazada hacia atrás en el tiempo una unidad temporal

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

- El **operador** **`F`** de adelanto (Forward) sobre una serie temporal es una nueva serie temporal desplazada hacia adelante en el tiempo una unidad temporal

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

Estos dos operadores se manejan como polinomios, se aplican tantas veces como se requiera y admiten operaciones con números reales.

Podemos aplicar estos operadores recursivamente (`B^m` se puede aplicar para construir una nueva serie `z'(t)=z(t-m)` ó `z'(t)=z(t+m)`, siendo `m` cualquier número entero). Además, admiten operaciones con reales (`z'(t)=(0.3*B):z(t)`). Las aplicaciones más conocidas de estos operadores son las medias móviles, series de incrementos, etc. El siguiente ejemplo ilustra estos conceptos:

Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636

Tiempo	T_{-1}	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_1	T_1	T_1
SD=(1-B):S			-2	1	-1	1	1	-2	0	0	1		
SD2=(B^2):S				2	0	1	0	1	2	0	0	0	1
SB=(B):S			2	0	1	0	1	2	0	0	0	1	
S		2	0	1	0	1	2	0	0	0	1		
SF=(F):S	2	0	1	0	1	2	0	0	0	1			
SF2=(F^2):S	0	1	0	1	2	0	0	0	1				

Ejemplo

```
// 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)+(F^2+3.5*F^4)):alfa;
// We construct a serie stepwise like through an step variable:
Polyn Stepwise = F+B^2+F^3+B^4+F^5+B^6+F^7;
Serie Step Stepwise = Stepwise:Step(y2005m3d01,Monthly);
```

6.2 Series deterministas en TOL

Una serie determinista, también denominada serie artificial, es una lista infinita de valores conocidos, que representa contenidos temporales o sucesos cualitativos. Las series deterministas se emplean en el análisis de intervención para representar comportamientos puntuales, hechos esporádicos o simplemente efectos calendarios en las series observadas.

Las series deterministas más habituales son:

- El **pulso** que toma valor 1 en una fecha y valor cero en todas las demás. Tiene por objetivo determinar la medida de la influencia que un suceso específico ha tenido sobre la variable objeto del análisis, cuando se espera que esa influencia se manifieste de forma puntual. Un pulso se genera con la función **`Pulse(Date, TimeSet)`**.

Ejemplo

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

- La **compensación** que toma valores 1 y -1 en fechas correlativas y valor cero en todas las demás. Una compensación es la diferencia entre dos pulsos consecutivos, la cual proporciona una medida de la influencia del acontecimiento representado cuando el efecto inicial puede verse total o parcialmente rectificado por un efecto de

signo contrario en fecha inmediatamente posterior. Una compensación se genera con la función **Compens(Date, TimeSet).**

Ejemplo

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

- El **escalón** que toma valor 1 a partir de una fecha hábil, y valor cero en todas las precedentes. Un escalón será la representación de un evento constante sobre la variable objeto de análisis a partir de un instante de tiempo. Un escalón se genera con la función **Step(Date, TimeSet)**.

Ejemplo

```
Serie StepG = Step(y2000, Monthly);
```

- La **tendencia** con valor cero hasta una fecha hábil en la que toma valor 1, creciendo aritméticamente con razón 1 en fechas posteriores. Una tendencia se genera con la función **Trend(Date, TimeSet)**.

Ejemplo

```
Serie TrendG = Trend(y2000, Monthly);
```

Otro grupo de variables artificiales son las variables calendario. Tienen por objetivo representar fenómenos relacionados con el almanaque y las agendas laborales, que ejercen una influencia sistemática sobre la variable objeto de análisis. Las variables calendario se pueden generan por medio de la función de series **CalVar(TimeSet, TimeSet)**, y la función **CalInd(…,…)** entre otras.

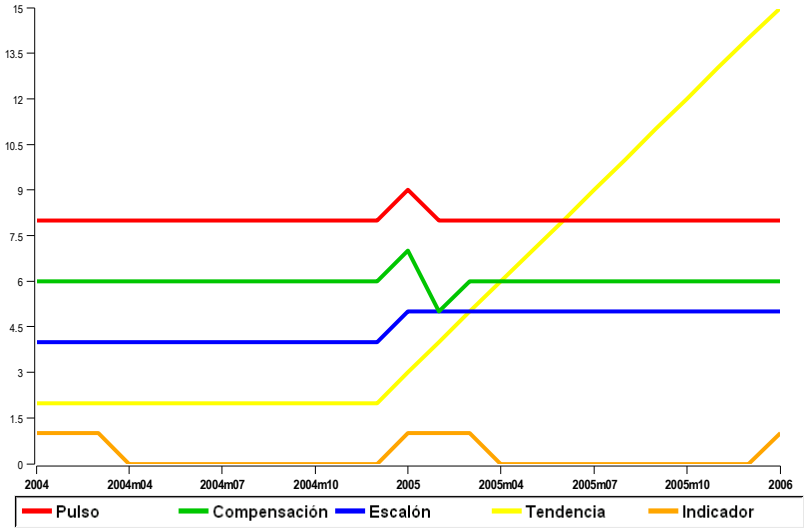
Veamos un ejemplo con declaración en **TOL** :

```
Serie IndicatorG    = CalInd(Y(2000), Monthly);
```

Y para verlas en un gráfico las acotamos entre dos fechas:

```
Serie pulse     = SubSer(8 + Pulse(y2005, Monthly),y2004m01,y2006m01);
Serie compense  = 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);
```

Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636



Observaciones:

- Al diferenciar un pulso obtenemos una compensación:

```
Serie PulseDif = (1-B):PulseG;
```

- Al diferenciar un escalón obtenemos un pulso:

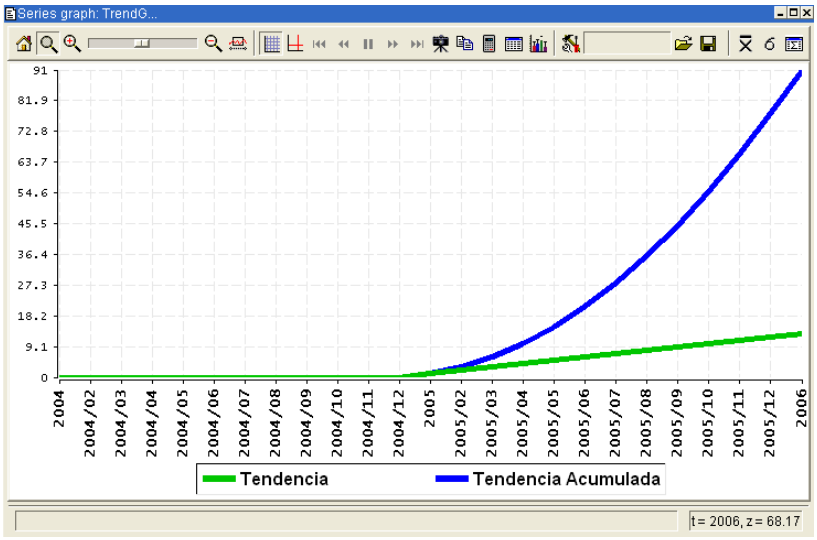
```
Serie StepDif = (1-B): StepG;
```

- Al diferenciar una tendencia obtenemos un escalón:

```
Serie TrendDif = (1-B): TrendG;
```

- Tendencia acumulativa con la función **DifEq**:

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



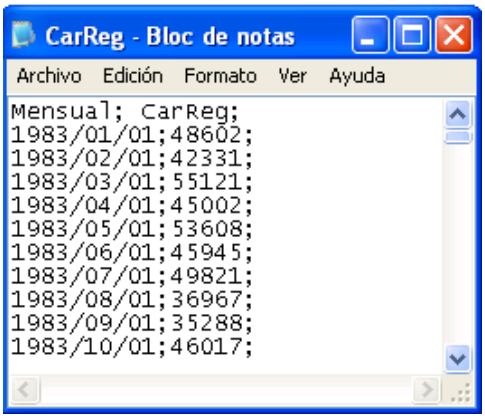
6.3 Introdución a la modelización

En este apartado, introduciremos brevemente cómo modelizar una serie temporal con **TOL** y emplear los modelos que mejor se ajusten para la previsión de datos. En un proceso de modelización, se deben seguir los siguientes pasos:

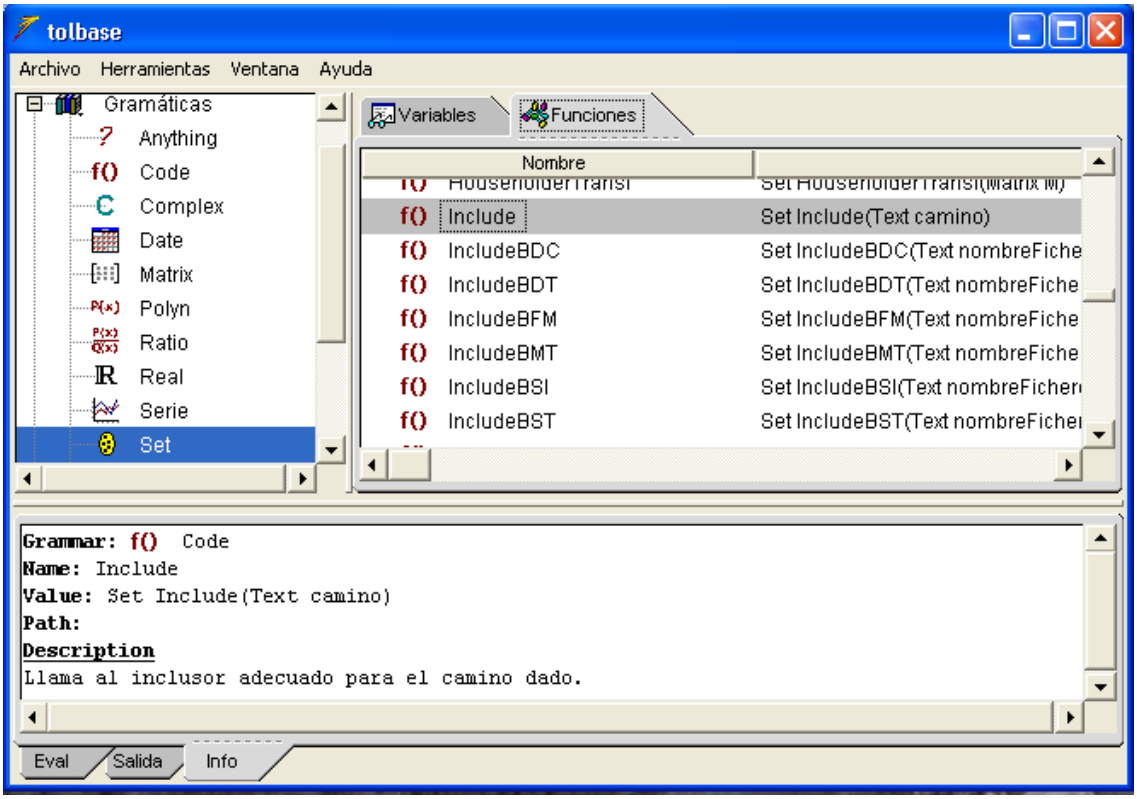
1. Análisis y planteamiento del problema.
2. Toma de datos.
3. Fase de identificación.
4. Estimación del modelo.
5. Validación.
6. Previsión.

6.3.1 Lectura de datos

Como dijimos anteriormente **TOL** almacena los datos en ficheros con formato **BDT** que es el formato básico que se utilizará para el almacenamiento de series temporales. Está basado en un fichero ASCII plano, que consta de una columna con el fechado (diario, mensual, anual,...) y las restantes columnas se corresponden con valores de series en dicho fechado. Las distintas columnas se separan unas de otras con punto y coma.



Recordemos que para leer el fichero desde utilizamos la función `IncludeBDT()` u otras similares (`Include()`, `IncludeBDC()`), cuya descripción y parámetros podemos consultarlos de la siguiente manera:



Es importante señalar que para la lectura del fichero es necesario detallar la ruta de directorios donde se encuentra el mismo.

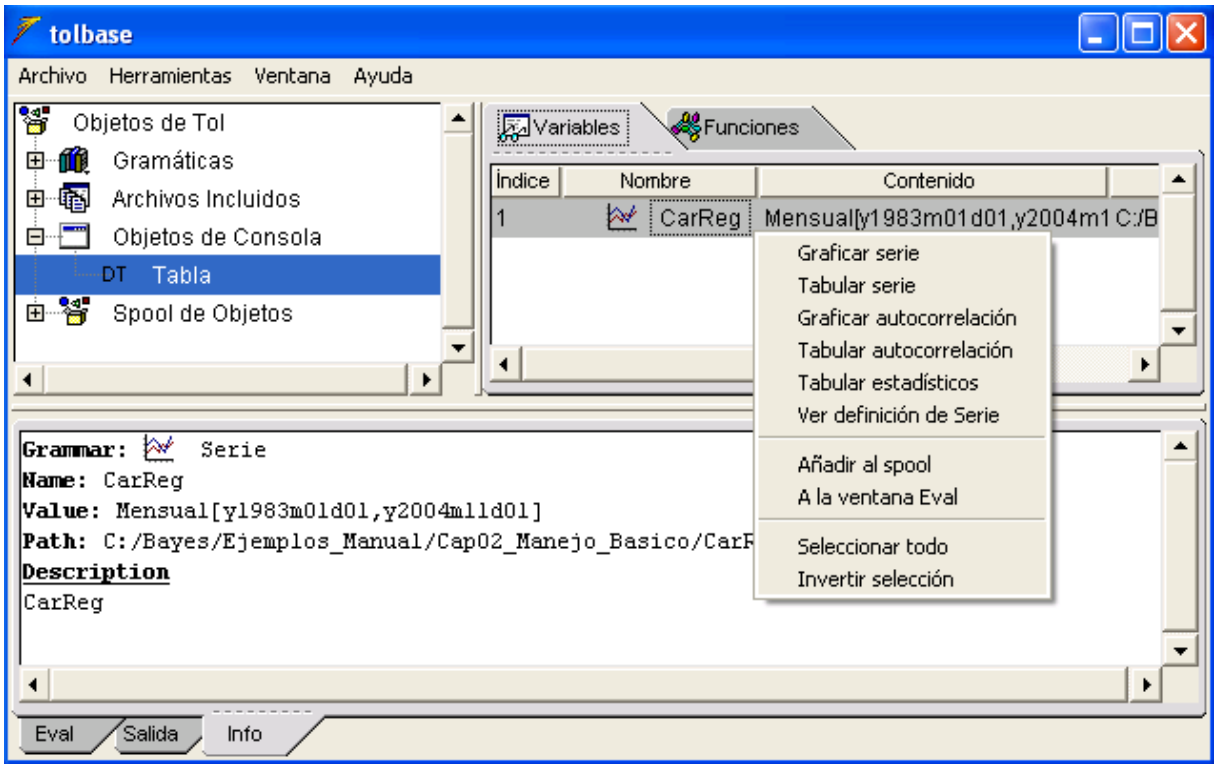
```
Set Table = IncludeBDT("C:\Bayes\Ejemplos_Manual\Cap06_Series\CarReg.bdt");
```

Vamos a trabajar con el fichero `CarReg` que contiene datos sobre la matriculación de automóviles. Se trata de una serie mensual desde enero de 1983 hasta Noviembre de 2004.

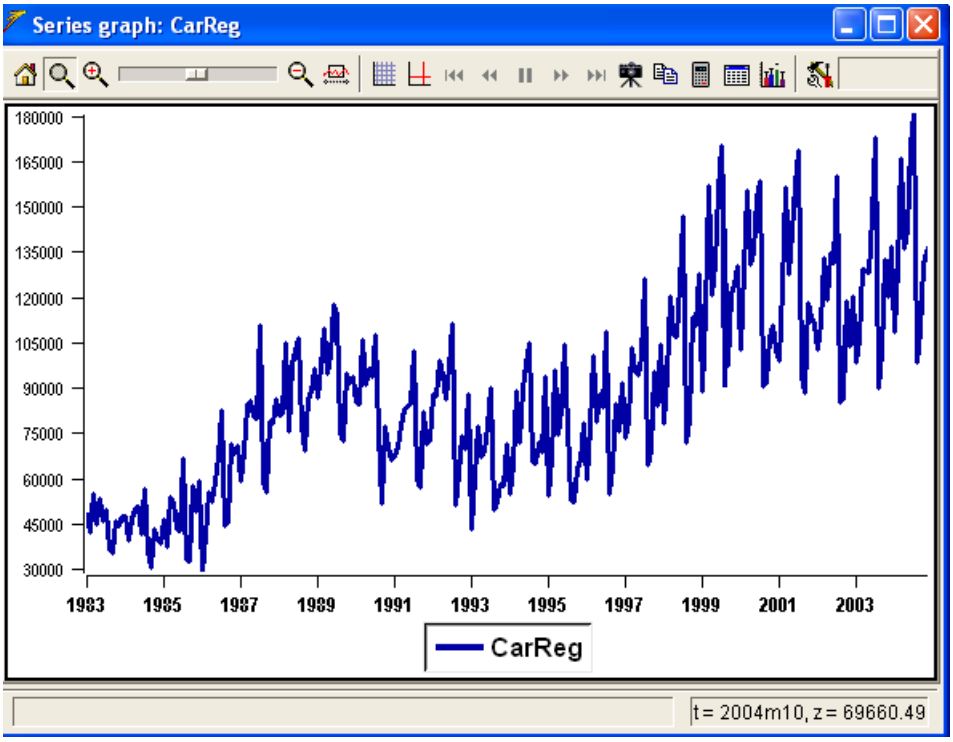
La mayor parte de las series temporales están generadas por modelos no estacionarios siendo necesarias transformaciones que las conviertan en procesos estacionarios, con objeto de utlilizar las ventajas que ofrecen éstos últimos para su modelización. En una primera fase de la elaboración de un modelo ARIMA se requieren fundamentalmente como instrumentos básicos de identificación la función de autocorrelación simple y parcial estimadas. Una vez identificado el modelo se obtienen unos valores estimados para los parámetros, y se lleva a cabo la fase de validación que va dirigida a establecer si se produce o no esa adecuación entre datos y modelo. Finalmente, en la fase de predicción, se realizan pronósticos en términos probabilísticos de valores futuros de la variable.

6.3.2 Fase de identificación. Visualización

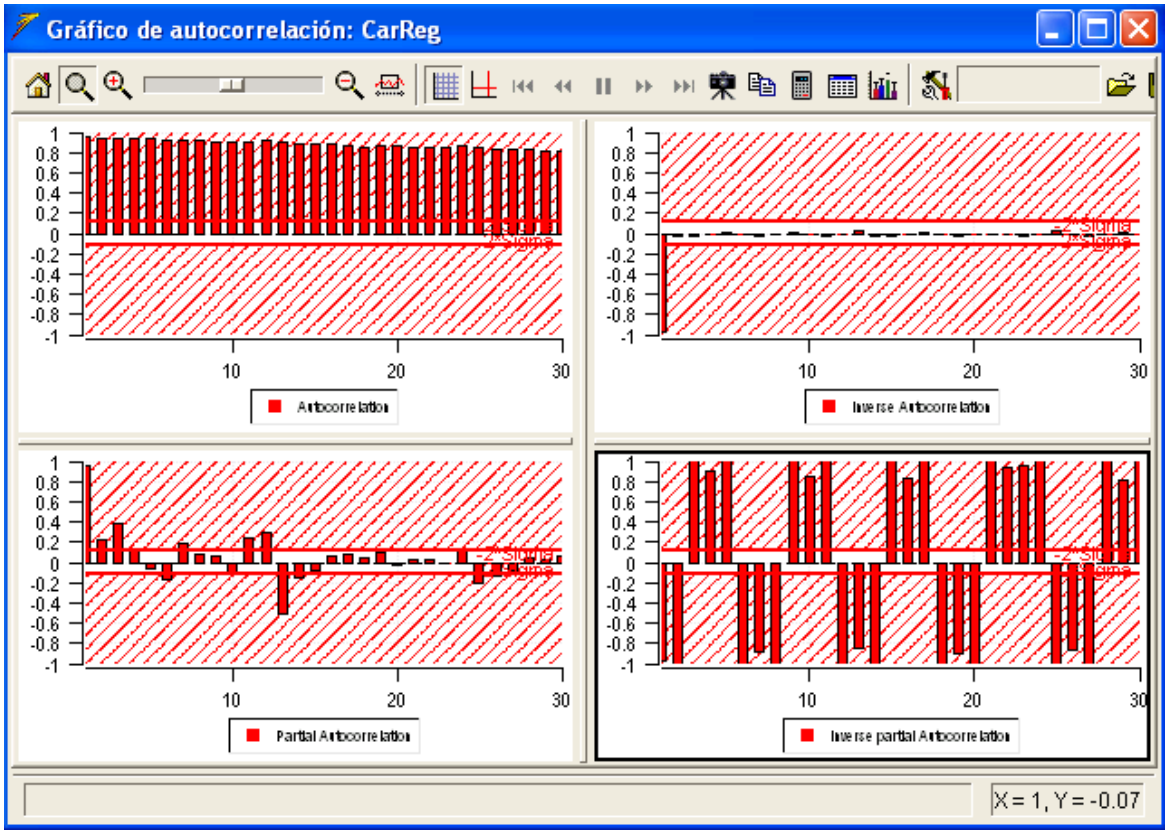
Una vez incluido el fichero, este aparece en Objetos de consola con el nombre que le hemos dado al conjunto, en nuestro caso **Tabla**. Dentro del conjunto seleccionamos la(s) serie(s) que queremos visualizar y pulsando el botón derecho del ratón aparecen entre otras opciones la de graficar y tabular la serie y las funciones de autocorrelación, así como la de ver algunos de sus estadísticos más importantes.



A continuación mostramos el gráfico de la serie, dentro de la interfaz gráfica tenemos también opciones que permiten cambios de escalas, selección de partes de la serie, cambio del número de marcas, etc.



En el siguiente gráfico mostramos las funciones de autocorrelación simple y parcial de la serie.



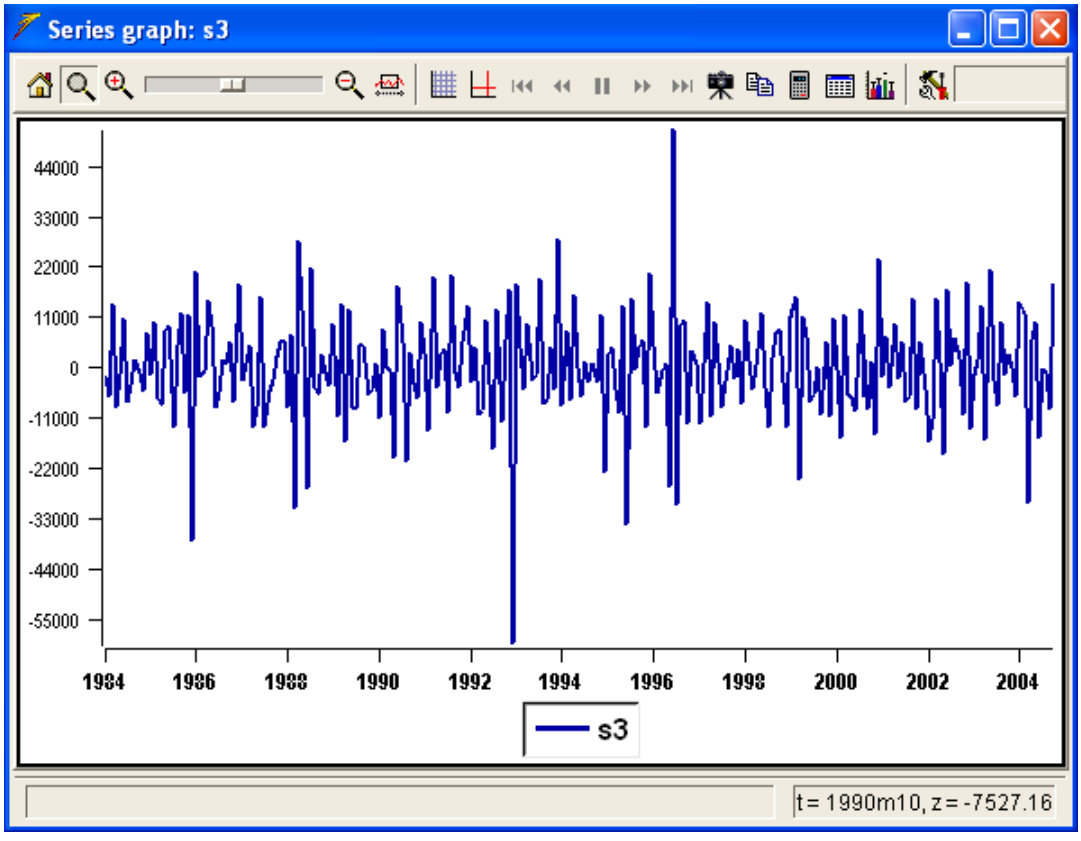
A la vista de los gráficos de la FAS y FAP de la serie se decide aplicar una diferencia regular de orden 1, una diferencia estacional anual y establecer el modelo que nos parece más apropiado. Para hacer este tipo de transformaciones se tienen que aplicar ciertos tipos de polinomios a las series.

```
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.
```

Otra forma de definir estas transformaciones, en lugar de hacerlo secuencialmente, sería utilizando productos de polinomios en **TOL** :

```
Polyn diff = (1-B)*(1-B^12);
Serie s3 = diff:CarReg;
```

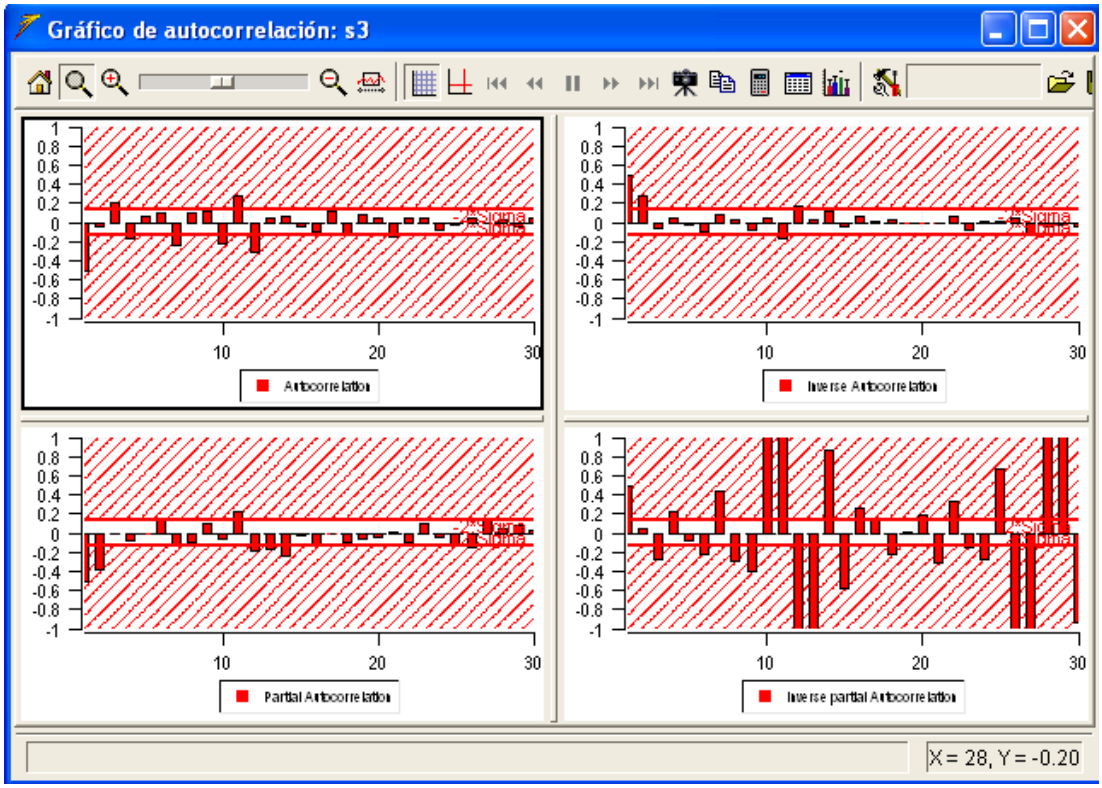
A continuación en la siguiente figura mostramos la serie s3 libre de la estacionalidad y tendencia. Se trata de un proceso estacionario donde se observan una serie de datos atípicos que sería necesario intervenir a través de variables artificiales.



Definimos entonces las siguientes variables que vamos a introducir en nuestro modelo y cuyo objetivo será captar todos estos efectos.

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

En la serie diferenciada estacional y regularmente observamos de nuevo su FAS y FAP para decidir los posibles modelos a estimar.



A la vista de estos gráficos proponemos ajustarle a la serie un proceso SARIMA$(2,1,0)x(0,1,1)_{12}$.

6.3.3 Fase de estimación

Una vez identificado el modelo que sigue la serie, procedemos a estimar los parámetros. Para ello, **TOL** dispone de la función

```
Set Estimate(Set modelDef [, Date desde, Date hasta], Set parametrosAPriori)
```

Esta función requiere:

- Como primer argumento un conjunto del tipo siguiente

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

Donde los parámetros representan:

Serie Output es la serie que pretendemos ajustar.

Real FstTransfor es un número real que representa la primera transformación Box-Cox.

Real SndTransfor es un número real que representa la segunda transformación Box-Cox.

Real Period representa la estacionalidad que presenta la serie.

Real Constant si queremos ajustar el modelo con constante.

Polyn Dif será el producto de los polinomios diferencia regular y diferencia estacional que necesitamos aplicar a la serie para convertirla en una serie estacionaria.

Set AR es un conjunto de polinomios con dos elementos. El primero es el polinomio AR de la parte regular y el segundo es el polinomio AR de la parte estacional.

Set MA es un conjunto de polinomios con dos elementos. El primero es el polinomio MA de la parte regular y el segundo es el polinomio MA de la parte estacional

Set Input es el conjunto de variables explicativas (por ejemplo intervenciones).

Es importante aclarar que este conjunto se construye utilizando la función:

Set InputDef(**Polyn** Omega, **Serie** X) donde Omega es el polinomio que se aplica al input que vamos a introducir en el modelo (que en general va a ser una constante), cuyos coeficientes deben ser estimados junto al modelo.

Set InputNonLin representa el conjunto de inputs no lineales.

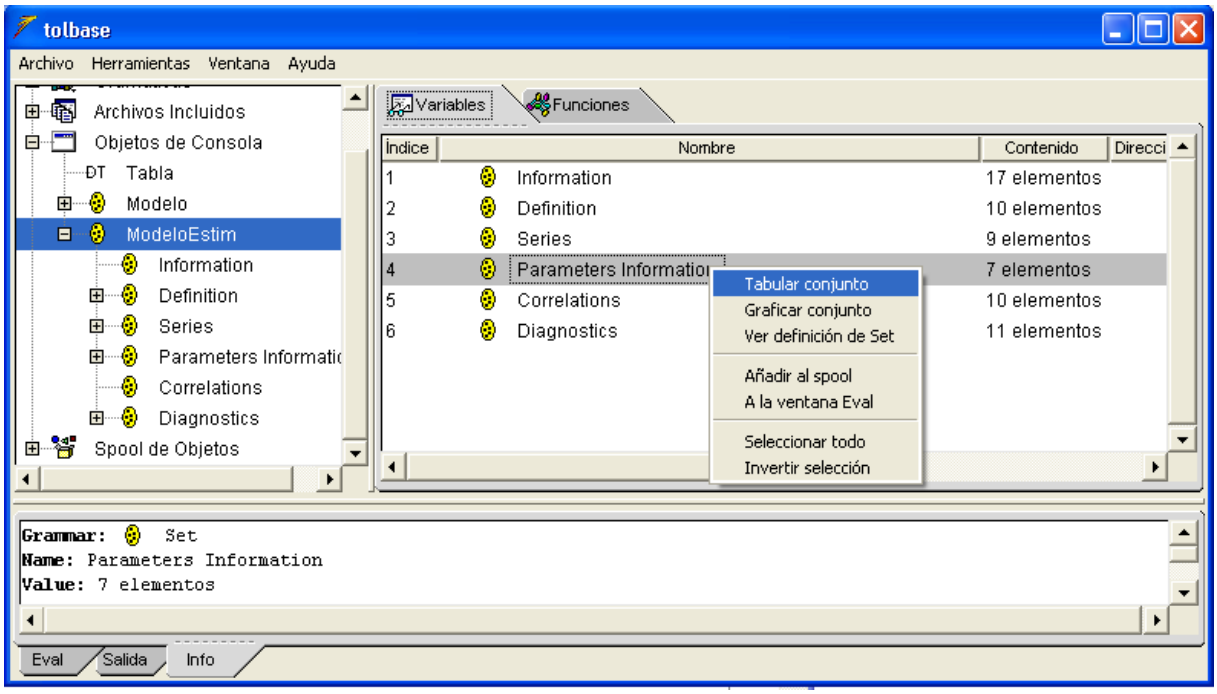
- Como segundo y tercer argumento (los cuáles son opcionales) las fechas inicial y final que queremos tener en cuenta para realizar la estimación, (en general se utilizan todos los datos disponibles).
- Como último argumento (también opcional) se utiliza **Set** parametrosAPriori

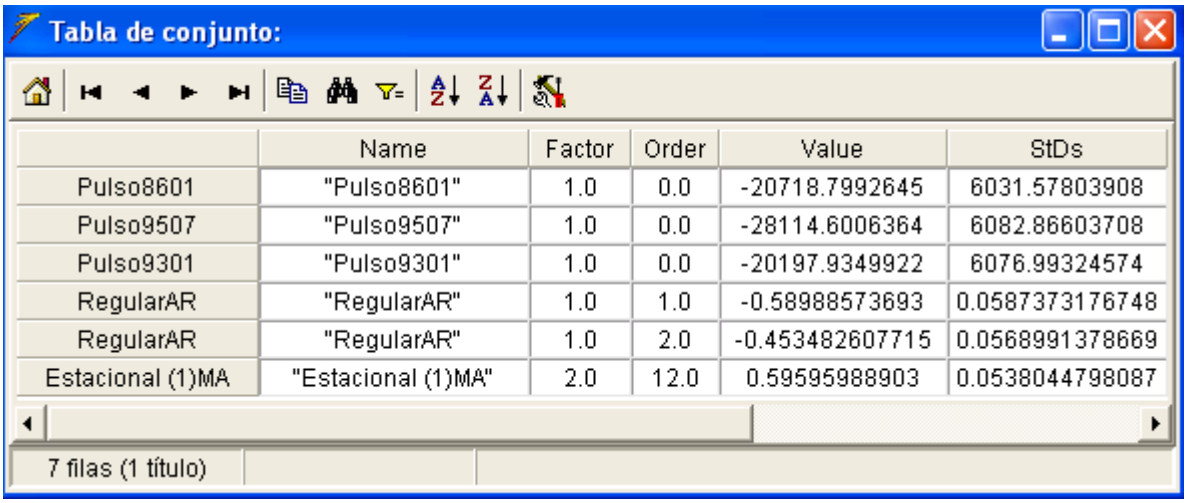
Para nuestro caso:

```
Set Modelo = 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
                       SetOfPolyn(1,1-0.1*B^12), // MA
                       SetOfSet(InputDef(0.1, Pulse8601),
                                InputDef(0.1, Pulse9607),
```

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

Una vez compilado obtenemos la siguiente salida





	Name	Factor	Order	Value	StDs
Pulso8601	"Pulso8601"	1.0	0.0	-20718.7992645	6031.57803908
Pulso9507	"Pulso9507"	1.0	0.0	-28114.6006364	6082.86603708
Pulso9301	"Pulso9301"	1.0	0.0	-20197.9349922	6076.99324574
RegularAR	"RegularAR"	1.0	1.0	-0.58988573693	0.0587373176748
RegularAR	"RegularAR"	1.0	2.0	-0.453482607715	0.0568991378669
Estacional (1)MA	"Estacional (1)MA"	2.0	12.0	0.59595988903	0.0538044798087

6.3.4 Fase de validación

El objetivo perseguido al elaborar un modelo SARIMA es encontrar un modelo que sea lo más adecuado posible para representar el comportamiento de la serie estudiada. Así, un modelo ideal sería el que cumpliera los siguientes requisitos:

- Los residuos del modelo estimado se aproximan al comportamiento de un ruido blanco.

- El modelo estimado es estacionario e invertible.
- Los coeficientes son estadísticamente significativos, y están poco correlacionados entre sí.
- Los coeficientes del modelo son suficientes para representar la serie.
- El grado de ajuste es elevado en comparación al de otros modelos alternativos.

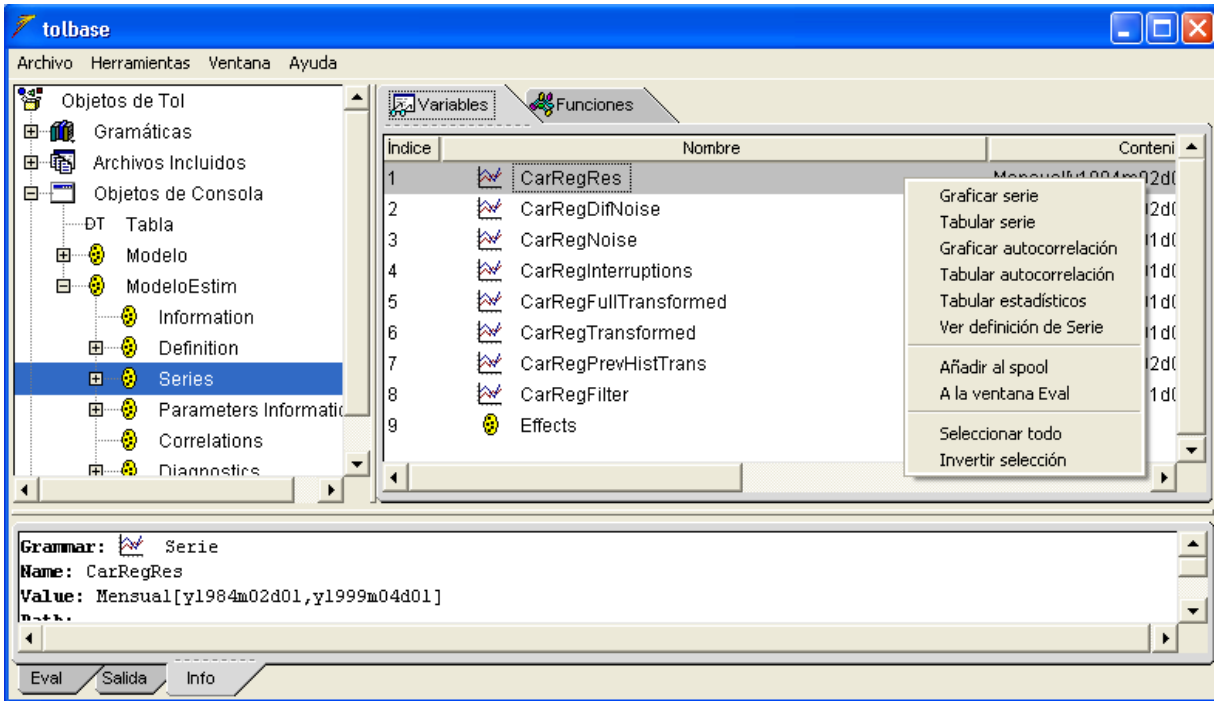
La finalidad de la fase de validación, consiste precisamente en analizar la adecuación entre el modelo y los datos. Para ello realizamos un análisis de los residuos que consiste en comprobar que los residuos se comportan como un ruido blanco.

Es importante aclarar algunas de las diferentes series que muestra la salida de este programa:

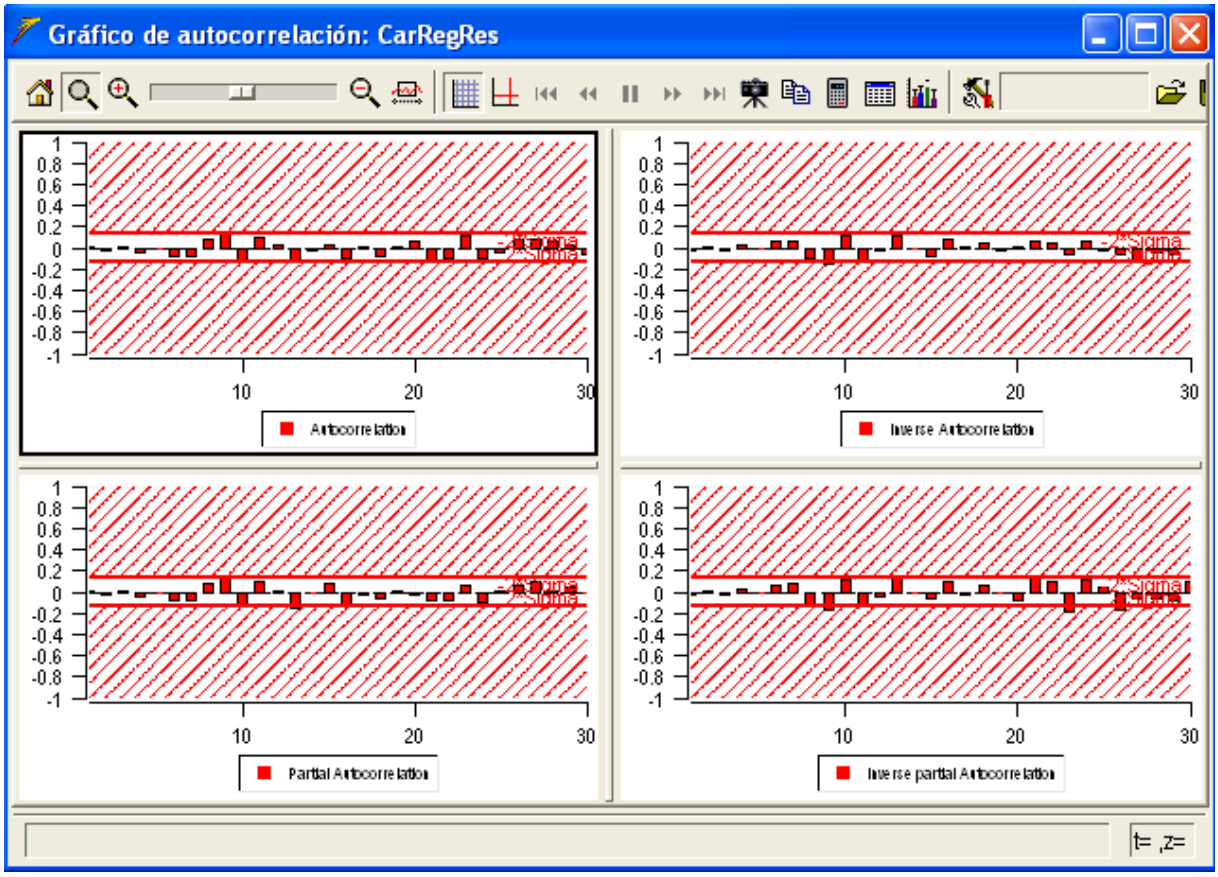
`CarRegRes` se trata de los residuos de la serie que se obtiene como resultado de haber estimado la serie original con todos sus efectos.

`CarRegNoise` se trata de la serie original pero sin los efectos de las variables artificiales.

`CarRegDifNoise` se trata de la serie `CarRegNoise` diferenciada de forma regular.



Para la fase de validación utilizamos la serie de los residuos, `CarRegRes`. Si representamos su FAS y FAP observaremos que se corresponden a las de un ruido blanco.

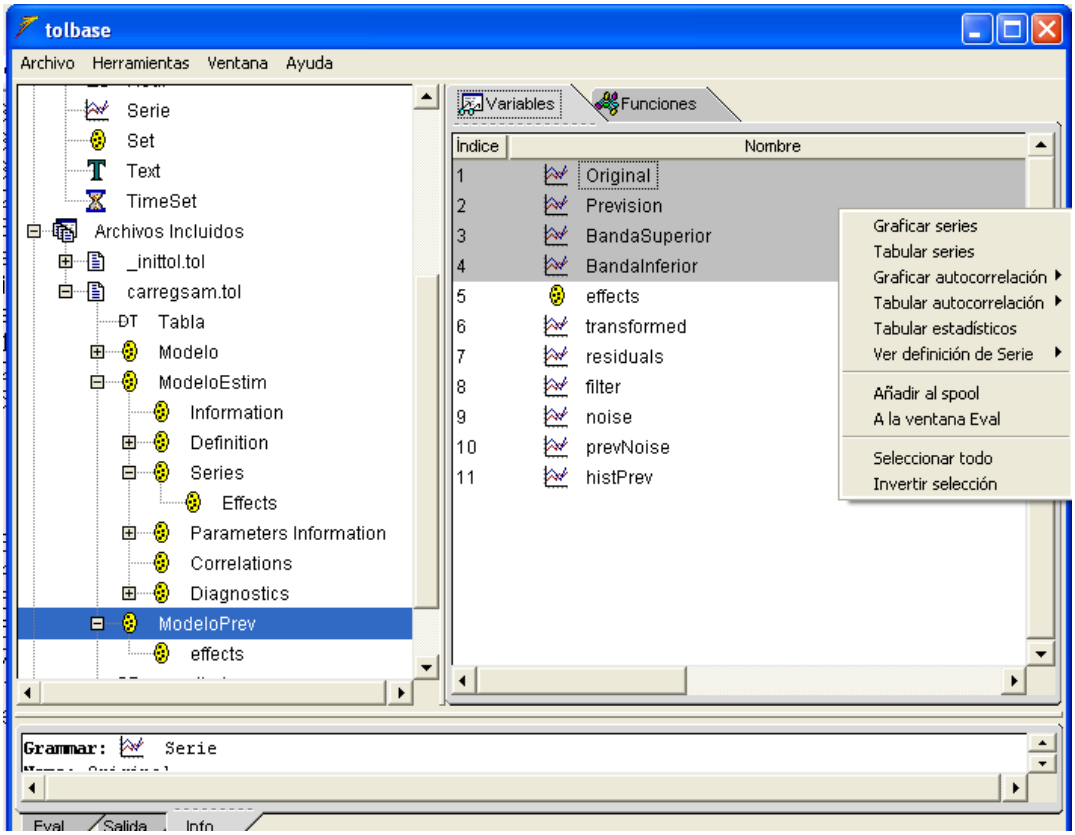


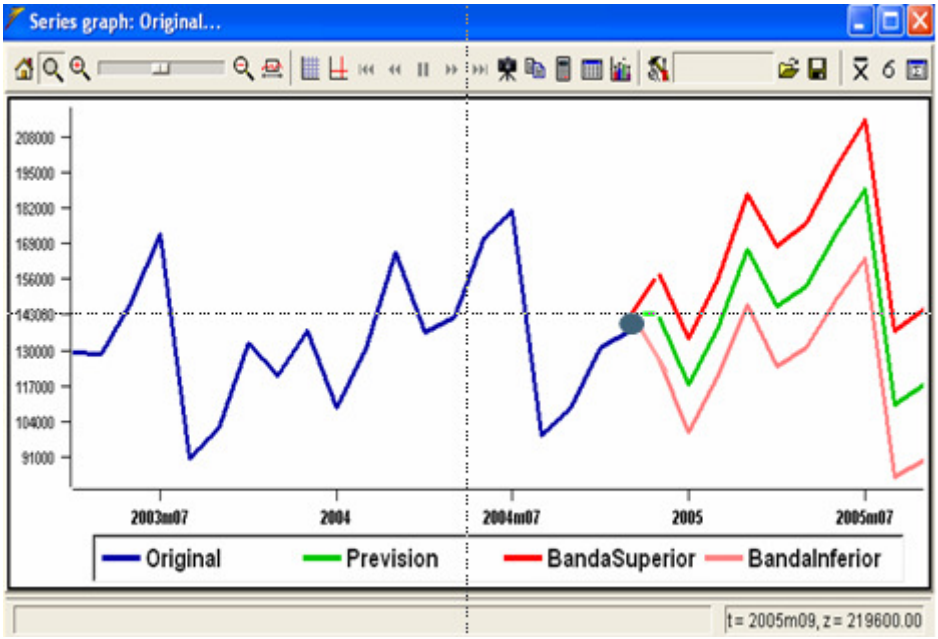
6.3.5 Fase de predicción

Las tres primeras fases de la elaboración de un modelo ARIMA constituyen un proceso iterativo cuyo resultado final es la obtención de un modelo estimado que sea compatible con la estructura de los datos. Una vez que se ha conseguido este resultado, la fase siguiente consiste en utilizar este modelo estimado en la predicción de valores futuros de la variable objeto de estudio. Por ejemplo, queremos predecir el año siguiente al término de los datos de la serie. Para ello, empleamos la función `CalcForecasting(modelo, Inicio Serie, Final Serie, Número de previsiones, alfa);`. Donde `alfa` es el ajuste de las bandas de confianza.

```
Date IniSer = First(CarReg);    // We define the date where the time serie starts
Date FinSer = Last(CarReg);     // We define the date where the time serie ends
Set ModeloPrev = CalcForecasting(ModeloEstim, IniSer, FinSer, 12,0.05);
```

Veamos un gráfico con la previsión del próximo año con los datos de los dos últimos años y las bandas de confianza al 95%:





Es importante también destacar que antes de que apareciese la metodología Box-Jenkins, el análisis clásico de series temporales se basaba en métodos de descomposición. Los movimientos que presenta la evolución de una serie temporal, se concebían como la resultante de la composición de cuatro efectos o fuerzas denominadas tendencia, ciclo, estacionalidad y componente irregular. Las cuatro fuerzas o componentes son no observables y se supone que su integración

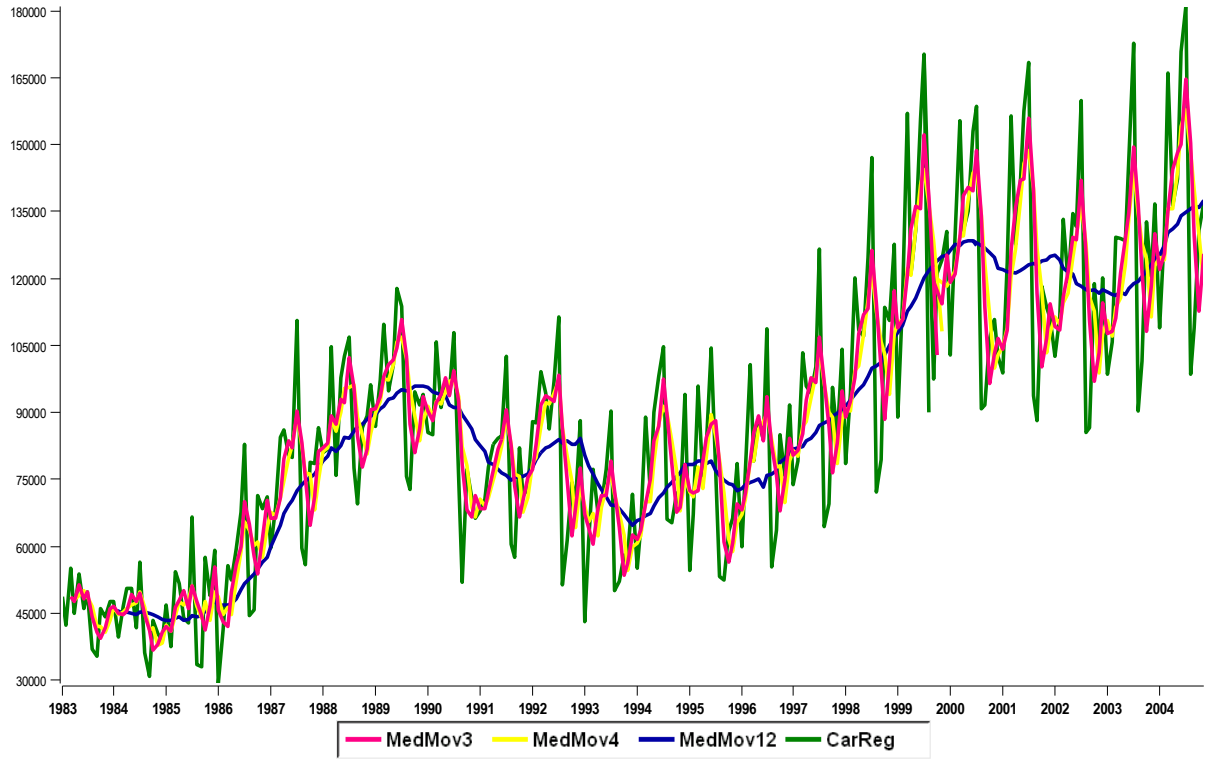
Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636

determina la evolución de la serie temporal. Una vez introducido esto, cabe señalar que en la evaluación de componentes tendenciales se utilizaban entre otros los llamados **métodos de medias móviles**. Estos métodos consisten en promediar cada valor de la serie con algunas de las observaciones que le preceden y le siguen. De esta forma se eliminan algunas variaciones accidentales y la serie resultante, se identifica con el componente tendencial. Un ejemplo de cómo introducir esta función en **TOL** es:

Creamos una función de usuario que devuelva la media móvil de orden `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 1nd 12 average moving function

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

Veamos el resultado de la aproximación de las series `MedMov3`, `MedMov4` y `MedMov12` para la serie `CarReg`:

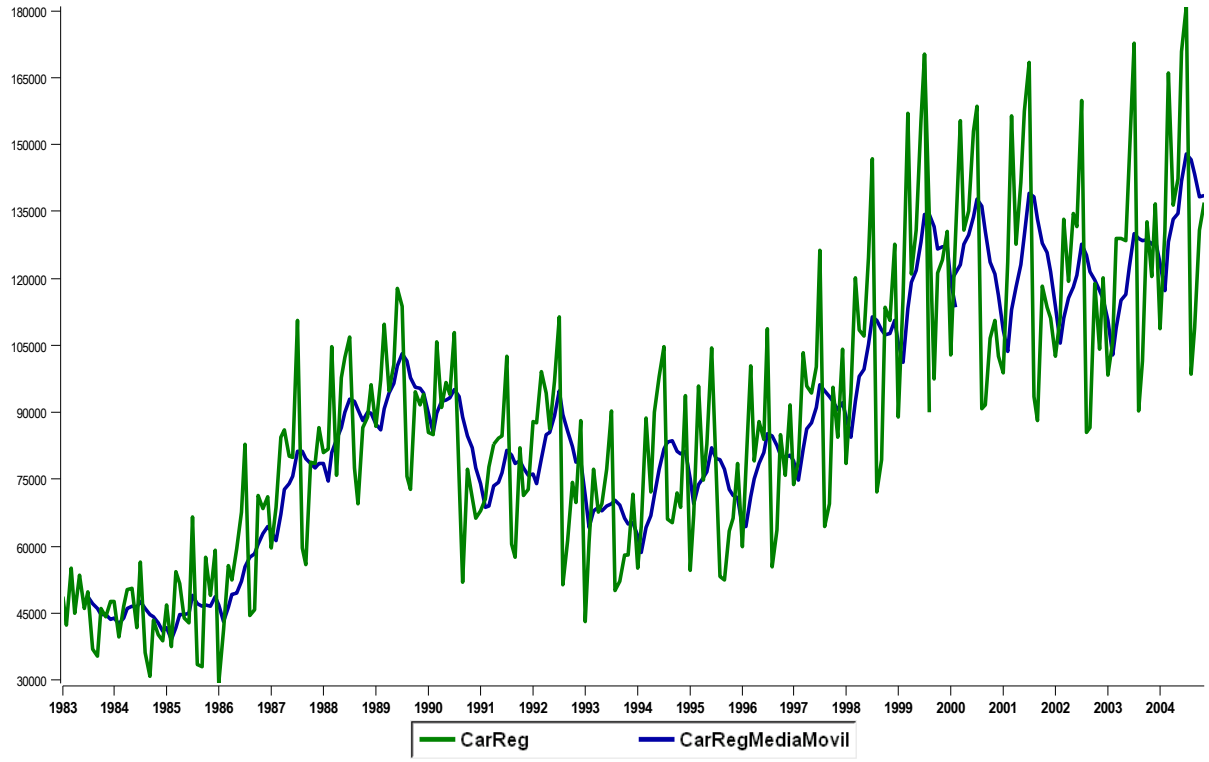


- `MovAverBack6()` es una función definida por el usuario que genera una media móvil con el valor actual y los valores de 6 periodos por detrás, su definición y uso es el siguiente:

```
Polyn PolyBack6 = 1+B+B^2+B^3+B^4+B^5+B^6;   // Definición del Polinomio
```

```
Serie MovAverBack6(Serie Ser){                    // Definición de la función
                                (PolyBack6:Ser)/7
                              };
Serie CarRegMediaMovil = MovAverBack6(CarReg); // Aplicar la media móvil
```

Veamos el resultado de la aproximación de la serie `CarRegMediaMovil` para la serie `CarReg:`



Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636

7. Otros tipos de variables

En los capítulos precedentes se han descrito los tipos de datos más importantes del lenguaje **TOL**. En este capítulo, detallaremos algunas de las operaciones y funciones más frecuentes para variables de tipo **Polyn**, **Ratio** y **Text**.

7.1 Funciones y operaciones con P(x) Polyn

Se han descrito anteriormente los polinomios de retardo y adelanto. Detallamos a continuación los polinomios Delta y desconocido:

- El **operador** **Delta** de diferencia regular sobre una serie temporal es una nueva serie temporal que representa el incremento por unidad de tiempo de la serie original.

Expresión: $Delta : z(t) = z(t) - z(t-1)$

- El **operador** **-?** desconocido sobre una serie temporal es una nueva serie temporal con valores omitidos.

Estos operadores se aplican tantas veces como se requiera y admiten operaciones con números reales.

Entre las funciones más importantes de manipulación de polinomios, se encuentran:

- La función **+** (suma), ***** (producto), **-** (resta), **/** (división) y **^** (potencia) entre diferentes polinomios.

Ejemplo:

```
// 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-B^5);
// The polynom 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 = Increase^3;
```

- La función **Derivate** (derivada de un polinomio), **Expand** (expansión de grado n de una razón de polinomios), **Integrate** (integral de un polinomio), **SetProd** (productorio de todos los polinomios de un conjunto), **SetSum** (sumatorio de todos

los polinomios de un conjunto), **Sub** (polinomio formado por los monomios de un polinomio de grados comprendidos entre los límites dados).

Ejemplo: Construimos diferentes polinomios

```
// We consider the two following polynomials:
Polyn TrimBackward = 1-B^3-B^6-B^9-B^12;
Polyn AnualBackward = 1-B^12;
// 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);
```

7.2 Funciones y operaciones con P(x)/Q(x) Ratio

Son fracciones racionales de polinomios. La principal utilidad es resolver ecuaciones en diferencias del tipo $P(B)Z_t = Q(B)A_t$. Los operadores básicos son:

- El operador **CompensOut** sobre una serie temporal, es una nueva serie temporal que representa la diferencia entre el valor actual y el valor inmediatamente anterior de la serie original.

Expresión: $CompensOut : z(t) = z(t) - z(t-1)$

- El operador **IDelta** (inversa del operador delta) sobre una serie temporal, es el operador inverso al anterior. El operador **StepOut** sobre una serie temporal, es equivalente al operador **IDelta**.

Expresión: $IDelta : z(t) = \frac{z(t)}{z(t) - z(t-1)}$

- El operador **TrendOut()** sobre una serie temporal, es una nueva serie temporal que representa la serie original, entre el resultado del valor actual menos dos veces el valor inmediatamente anterior de la serie original más el valor previo al anterior.

Expresión: $TrendOut : z(t) = \frac{z(t)}{z(t) - 2 * z(t-1) + z(t-2)}$

- El operador **Unknown** desconocido sobre una serie temporal es una nueva serie temporal con valores omitidos.

Para aplicar estos operadores a una serie temporal, se emplea el operador **DifEq**.

Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636

Ejemplo

Aplicamos los operadores anteriores a la serie temporal **CarReg** que se encuentra ubicada en C:\Bayes\Ejemplos_Manual\Cap07_Otros. Para añadir el fichero, escribir en la ventana Eval:

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

Entre las funciones más importantes de manipulación de ratios, se encuentran:

- La función **+** (suma), ***** (producto), **–** (resta), **/** (división de polinomios), **:** (división de ratios) y **^** (potencia) entre diferentes fracciones de polinomios.

Ejemplo

```
// We consider the following two rations:
// a) Ratio CompensOut:
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 substract ration:
Ratio substract = compens - fraction1;
```

7.3 Funciones y operaciones con Text

Un objeto de tipo texto puede contener cualquier cadena de caracteres ASCII. La forma de crear una variable de tipo texto es poner el texto entre comillas. **TOL** permite diversos tratamientos de las variables de texto (búsqueda de cadenas, transformaciones, etc.).

Entre los operadores básicos de **TOL** para textos cabe destacar la inclusión de formato HTML para la generación de informes.

Describiremos únicamente funciones básicas de manejo de textos, de conversión de cualquier tipo de objetos a objetos de tipo texto.

- La función FormatReal es un ejemplo de conversión de objetos de tipo real a tipo texto.

Ejemplo

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

- La función GetFileExstension es un ejemplo de conversión de captura de extensiones (se pueden capturar caminos, nombres de ficheros, etcétera).

Ejemplo

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

- La función Grammar retorna el tipo de una variable.

Ejemplo

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

- La función ListOfDates retorna el conjunto de fechas entre dos fechas dadas en el fechado indicado.

Ejemplo

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

- La función ToLower es un ejemplo de cambio a letras minúsculas de un texto.

Ejemplo

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

Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636

8. Recomendaciones generales de programación

Tanto en este como en cualquier otro lenguaje es necesario una forma común de programar por multitud de motivos:

- La **reutilización** de código por parte de otras personas.
- Hacer la **lectura** del mismo más fácil y **transparente**.
- Mayor **accesibilidad** a la hora de retomar el código después de ser programado.

Además la caracterización de formas comunes a todos los códigos produce un enriquecimiento del mismo.

8.1 Organización de un archivo .tol

Se pueden encontrar varias partes dentro de un archivo .tol:

- **Cabecera**: Es la parte en la que se hace alusión al archivo .tol y en la que se resume el contenido y objetivo del mismo.
    - **Archivo**: Nombre del archivo en minúscula lo más significativo posible e incluyendo la extensión.
    - **Propósito**: Se declara el contenido y objetivo del archivo. Se escribe todo en minúscula respetando la puntuación y las mayúsculas pero sin la utilización de acentos.
- **Cuerpo**:

Las distintas partes del cuerpo se separan a través de dos líneas en blanco. Los rótulos de los encabezados de cada parte irán en mayúsculas y separados por líneas inclinadas (**slash**). Los subrótulos al igual que los anteriores deben separarse por líneas de **slash** e irán con la primera letra en mayúsculas y el resto en minúsculas. Harán referencia a conjuntos de objetos que tengan una misma función o característica. Los títulos de los rótulos pueden ir en inglés o español indistintamente pero siempre en el mismo idioma.

- **Constantes**: Estas deben declararse con la primera letra en mayúscula. Todas las constantes deben describirse a través de `PutDescription` dejando una línea en blanco de separación entre ellas.

  Para ser más sencilla su utilización en el resto del código es conveniente en el caso de tener gran cantidad de constantes *empaquetar* las mismas, es decir usar una marca (en adelante ***tag***) sobre estas para identificarlas, usualmente las tres letras iniciales. Por ejemplo: En el caso de tener constantes de control podiamos tener varios tipos a saber, de traza, de error, de aviso, etc. Con lo que el paquete podría ser CtrTace, CtrError, CtrWarn, etc..

- **Estructuras**: Se declaran entre llaves o entre paréntesis, es decir, { o (,) o }.
- **Inclusiones**: Inclusión de archivos.

- **Funciones:** Pueden estar en el mismo fichero o en otros ficheros que se incluyen en la parte de inclusiones. Su nombre está ligado a un *tag*. Se especifican más adelante.
- **Procedimientos:** Son segmentos de código que no son funcionales y que engloban formas de hacer de otros tipos de funciones. Se especifican más adelante.

Evidentemente no todo archivo .tol tiene necesariamente todas las partes del cuerpo, dependerá del uso al que se le vaya a dar al archivo.

```
////////////////////////////////////////////////////////////////////////////
// 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.
////////////////////////////////////////////////////////////////////////////
```

8.2 Estructura de una función

8.2.1 Encabezado

Se declara la función entre dos líneas de **slash** `/`. A continuación indentado dos espacios se escribe el tipo de dato de retorno de la función, después dejando un espacio el nombre de la función con la primera letra en mayúscula y por último sin espacio intermedio se declaran los argumentos de la función en minúscula, escribiendo el tipo de dato de cada uno y dejando un espacio detrás de las comas.

- **Gramática:** Tipo de dato de retorno de la función.
- **Nombre:** El nombre de la función debe ser lo más corto posible y utilizando en primer lugar el **tag** correspondiente al grupo de funciones que pertenezca (si pertenece a alguno). Seguidamente se utilizará la estructura **verbo en infinitivo más objeto** para designar el cometido la función, es decir: una función que convierta cualquier cosa a texto la podríamos escribir con el nombre CnvAny2Txt.
- **Argumentos:** Los nombres de los argumentos deben ser lo más explícito posible, es decir, si un argumento es de tipo `Set` y es un conjunto de series el nombre de este argumento debería ser `SetSer`.

8.2.2 Cuerpo

Se declara el código delimitado entre llaves e indentado dos espacios. Las normas sobre definición de argumentos dentro de la función son idénticas a las anteriormente citadas. Para cláusulas **TOL** como `EvalSet`, `For`, `While`, `Select`, `Sort`, etc. la estructura es exactamente igual a la que se describe anteriormente.

8.2.3 Caso especial: Claúsula IF

La identación de código al usar la cláusula `If` se presenta de distintas maneras en función de la complejidad de las sentencias:

Ejemplo:

```
////////////////////////////////////////////////////////////////////////////
// Example 1.
////////////////////////////////////////////////////////////////////////////

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

Ejemplo:

```
////////////////////////////////////////////////////////////////////////////
// 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);
```

8.2.4 Descripción

En esta parte de la función se hace referencia a lo que hace la función intentando ser breve e intentando explicar lo mejor posible sus argumentos. El comienzo de la descripción debe explicar lo que retorna la función.

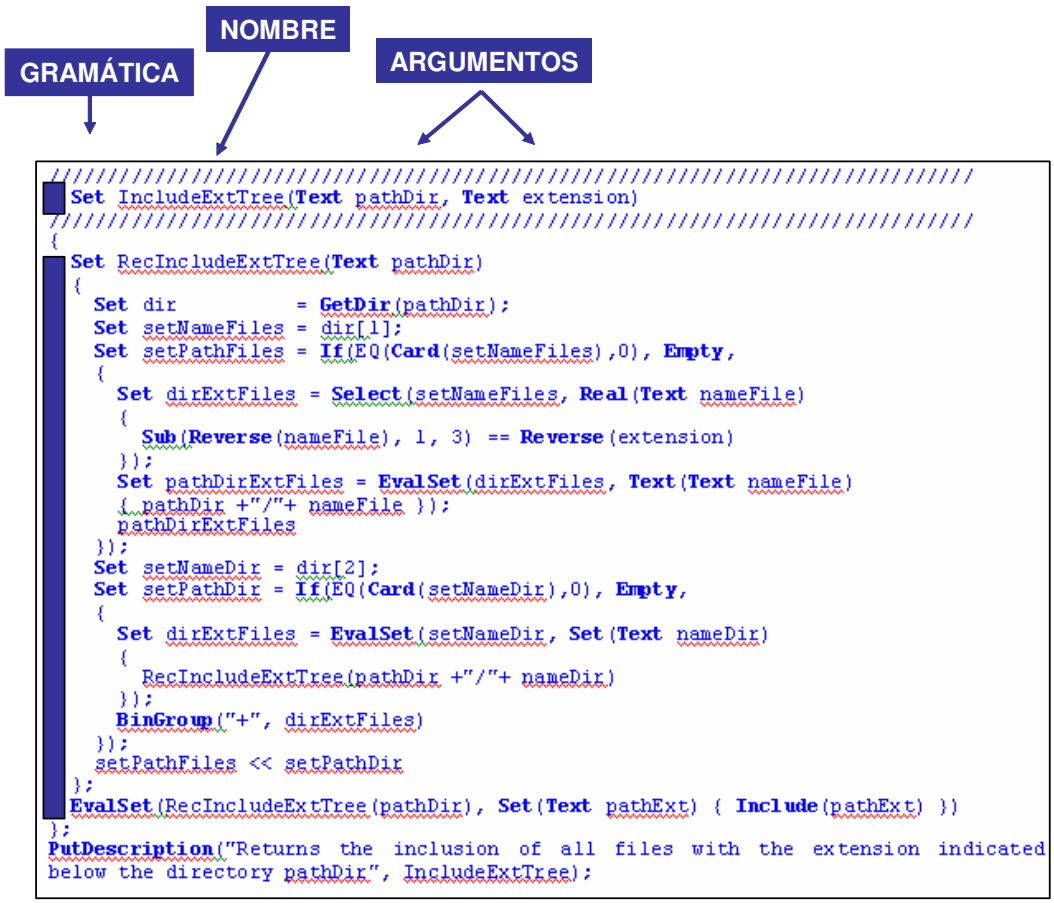
Va a continuación del cuerpo y entre dos líneas de ***slash***. La estructura del comando **`PutDescription`** es como sigue:

- **Descripción:** Se escribe la descripción de la función atendiendo a las anteriores normas. Cuando el texto exceda de una línea, se fuerza el salto y se alinea con las comillas de apertura.

- **Nombre de la función:** Se escribe el nombre de la función a la que hace referencia la descripción alineado con el paréntesis inicial.

A continuación se explicita una función que puede servir de ejemplo para lo anteriormente citado sobre funciones.

Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636




```
////////////////////////////////////////////////////////////////////////////
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);
```

9. ANEXO

En este anexo, describiremos algunos aspectos relevantes en cuanto a la generación de ficheros de datos en otros editores con **excel** y con editores de texto.

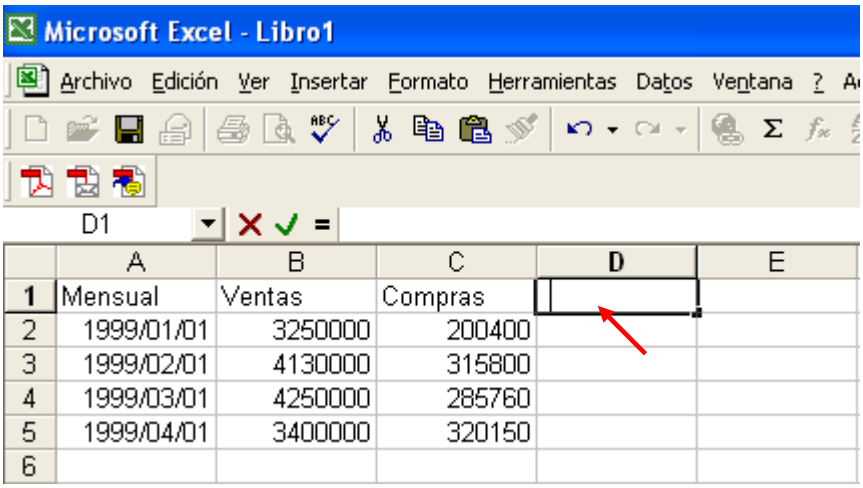
9.1 Generación de ficheros BDT con Excel

Una opción para generar este tipo de ficheros es a través de **excel**, seleccionando guardar como CSV delimitado por comas. Para tener al final de cada fila un punto y coma se necesita añadir una columna con un espacio (es decir, una columna que en cada una de las filas únicamente haya un espacio).

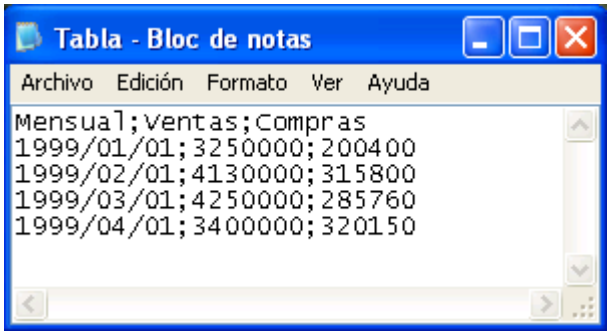
Ejemplo Queremos generar una tabla de datos que contenga los siguientes campos:

Mensual	Ventas	Compras
Enero 1999	3.250.000	200.400
Febrero 1999	4.130.000	315.800
Marzo 1999	4.250.000	285.760
Abril 1999	3.400.000	320.150

Escribimos los datos en un fichero **excel** y añadimos una columna que contenga en cada celda un espacio en blanco:



Seleccionamos guardar como CSV delimitado por comas (por ejemplo: en la carpeta `C:\Bayes`) con un nombre identificador (por ejemplo: `Tabla.csv`). Si abrimos el fichero resultante con el bloc de notas, obtenemos:



Para trabajar con este fichero, que tiene formato de tabla **TOL** (formato **BDT**), escribir en un fichero **TOL** la siguiente sentencia. Pinchar en Guardar y compilar (especificar un nombre para identificar posteriormente el archivo, por ejemplo `EjemploIncluirDatos.tol`

```
Set Tabla = IncludeBDT("C:\Bayes\Tabla.csv");
```

Esta sentencia indica que nuestro fichero tiene formato de tabla **BDT**, la separación de las distintas columnas viene indicada por un punto y coma (`;`) y los datos comienzan en Enero de 1999 (y1999m01d01) y finalizan en Abril de 1999 (y1999m04d01).

9.2 Ciclos con While

Esta sentencia permite ejecutar repetidamente, **mientras se cumpla una determinada condición**, una sentencia o bloque de sentencias.

La forma general es como sigue:

```
While(CondicionDeContinuacion, Iteracion)
```

Explicación: Se evalúa `CondicionDeContinuacion` y

- si el resultado es **false** no se evalúa `Iteracion` y se prosigue con la ejecución del programa.
- si el resultado es **true** se ejecuta `Iteracion` y se vuelve a evaluar `CondicionDeContinuacion` (evidentemente alguna variable de las que intervienen en `CondicionDeContinuacion` habrá tenido que ser modificada, pues si no el **bucle** continuaría indefinidamente).

En otras palabras, `Iteracion` se ejecuta repetidamente mientras `CondicionDeContinuacion` sea **true**, y se deja de ejecutar cuando `CondicionDeContinuacion` se hace **false**. Obsérvese que en este caso el **control** para decidir si se sale o no del **bucle** está antes de `Iteracion`, por lo que es posible que `Iteracion` no se llegue a ejecutar ni una sola vez.

Los ciclos **`While()`** siempre están asociados a variables que se reasignan con el **operador `:=`**.

Ejemplo

El siguiente ejemplo muestra un ciclo simple, donde

- `n` define el contador del ciclo
- `n<10` define la condición de parada
- y la `Iteracion` consiste en escribir el número de iteración

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

Traza por la **ventana de mensajes**:

```
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
```

Ejemplo Este ejemplo muestra un ciclo **`While()`** simple donde la condición de parada `LE(beta,alfa)` no depende de un incremento de una variable de uno en uno, como en el caso anterior, sino del decremento de una variable `alfa`, que en cada ciclo de iteración es dividida entre dos (`alfa := alfa/2`).

La variable `contador`, en este caso, sólo se utiliza a efectos de visualizar el número de iteraciones para alcanzar la condición de parada.

```
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
  }
};
```

Bayes Forecast
Gran Vía 39, 5ª planta, 28013 Madrid (Spain)
Tfn (34) 915327440, Fax (34) 915322636

Traza por la **ventana de mensajes**:

```
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
```

Ejemplo Este ejemplo presenta un ciclo **`While()`** anidado, donde

- el ciclo principal se define usando el contador `n` y consiste en la escritura del número de iteración **`WriteLn`**`("Iteration number " + `**`FormatReal`**`(n)).`
- el ciclo secundario viene definido mediante el contador `m` y escribe el número de subiteración `("Subiteration 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
   }
);
```

Traza por la **ventana de mensajes**:

```
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
```