INTRODUCTION

SEMoLa (Simple, Easy to use, MOdelling LAnguage), developed by Francesco Danuso at the Department of Agricultural and Environmental Sciences of the University of Udine (Italy) with many contributions by other researchers, is a simulation and modelling environment to create computer models for dynamic systems. It is logically formed by three components:

a) a non-procedural syntax for model coding;

b) a command set to create and evaluate computer simulation models;

c) a GUI environment to manage the simulation environment.

The commands allow to document the model, to display simulation results, to perform sensitivity and uncertainty analysis. The management of the modelling environment is accomplished by commands for set, display and clear the variables of the modelling environment, for quick editing the model files, to create files for multiple simulations, to display SEMoLa files and to log the working session. An error handling system and an extensive on-line help is also available.
SEMoLa is a simple simulation meta-language. Model coded in SEMoLa are translated into Basic or C++ code and then compiled into executable files. The Basic code is to be compiled by the Power Basic Console Compiler ١,٠ or higher (PowerBasic); the C++ code is to be compiled by MinGW ٢,٠,٠-٣
A package of tools like neural network generation, regression analysis, model units checking, has been developed to extend SEMoLa capabilities and offer to the user a more powerful modelling and analysis environment

Exogenous variable files (input variables) can be loaded and processed if requested. Multiple simulation runs (batch simulation) may be performed in relation to different initial values, parameter sets, exogenous variables or event scenarios (or combinations). In the simulation phase it is possible to obtain the sensitivity analysis, the parameter calibration (Gauss - Newton optimisation method) and the model validation against independent data.
SEMoLa allows the user to represent the system aspects in a conceptual rather than computational order. This makes the code more readable and the errors easier detectable. A SEMoLa model may directly produce presentation tables and lists of variables and parameters by using the specific commands.

The commands for compilation and simulation derive from the command **simula** (Danuso, ١٩٩٢).

FEATURES OF SEMoLa

- Non procedural programming language for dynamic models
- Generation of stand-alone, self-calibrating model executables
- Numerical integration of ordinary differential equations (euler, trapezoidal)
- Complete handling of the exogenous (input) variables
- Event handling capability (conditional, periodical, scheduled)
- Multiple simulations (batch simulations)
- Parameter calibration and optimisation (Gauss-Newton);
- Multiple calibration
- Sensitivity analysis;
- Model validation;
- Support for simulation experiments
- Support for uncertainty analysis and Monte Carlo simulations
- Errors checking in compilation and in run-time phases
- Model documentation (tables, listings, where used)
- Model structuring by sections
- Management of the modelling environment
- Automatization of procedures with command scripts
- Help for language syntax, procedure and commands
- Log files for SEMoLa sessions and model runs
- Multiple and non-linear regression
- Neural networks training and building with easy integration in simulation models
- Neural networks available as Basic and C++ source code
- Model code editor with specific highlighting
- Semantic and formal automatic debugging
- Automatic units (dimensional) consistency check
- Automatic documentation building for each model, in *.chm format
- Simulation experiments and scenario analysis
- Random variables generation from multiple distribution (normal and non-normal)
- User defined functions generation and management
- Fuzzy logic expert system development as user functions
- Statistical analysis on observed and simulated data
- Observed and simulated data plotting

SEMoLA FRAMEWORK

SEMoLa is a modelling framework for knowledge integration. It consist of

a) a non-procedural syntax for model coding (SEMoLa language);

b) a set of commands to perform all tasks in both intaractive and batch (script) mode;

c) a GUI environment to manage the simulation environment

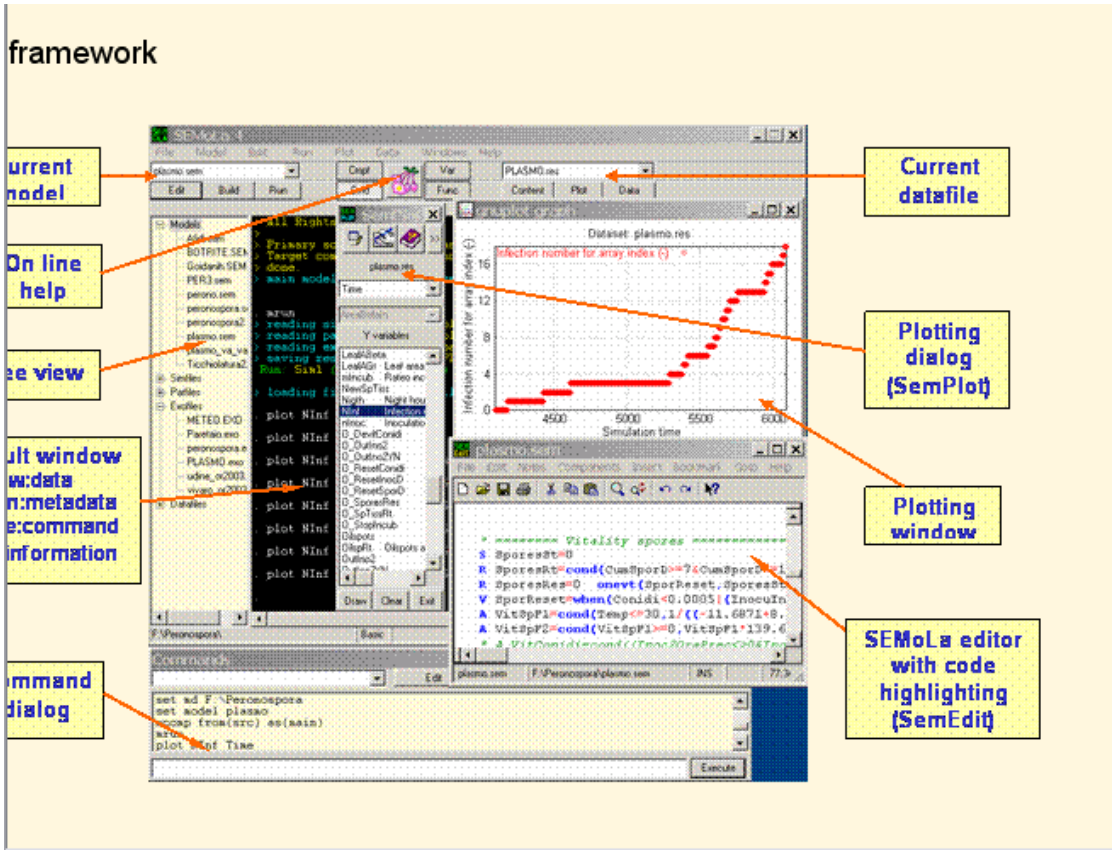
d) a built-in database management system (SemData)

e) a plotting capability (SemPlot, based on Gnuplot)

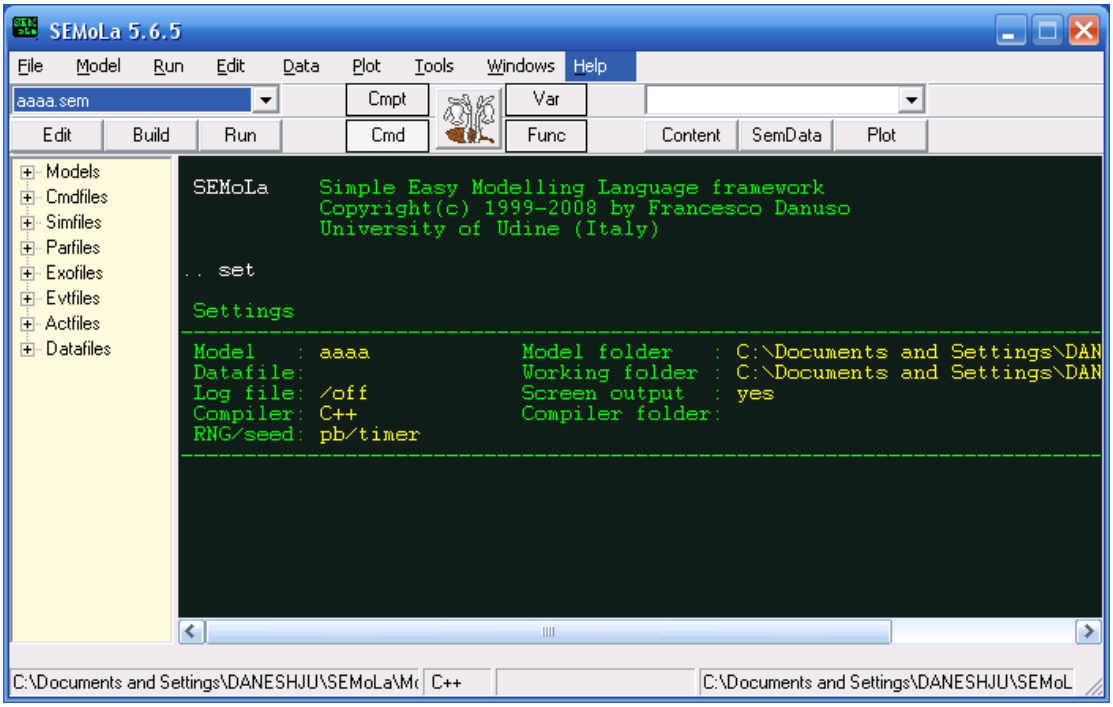
f) a specific editor (SemEdit)

A SEMoLa model is a text file in which every row completely describes a system component.

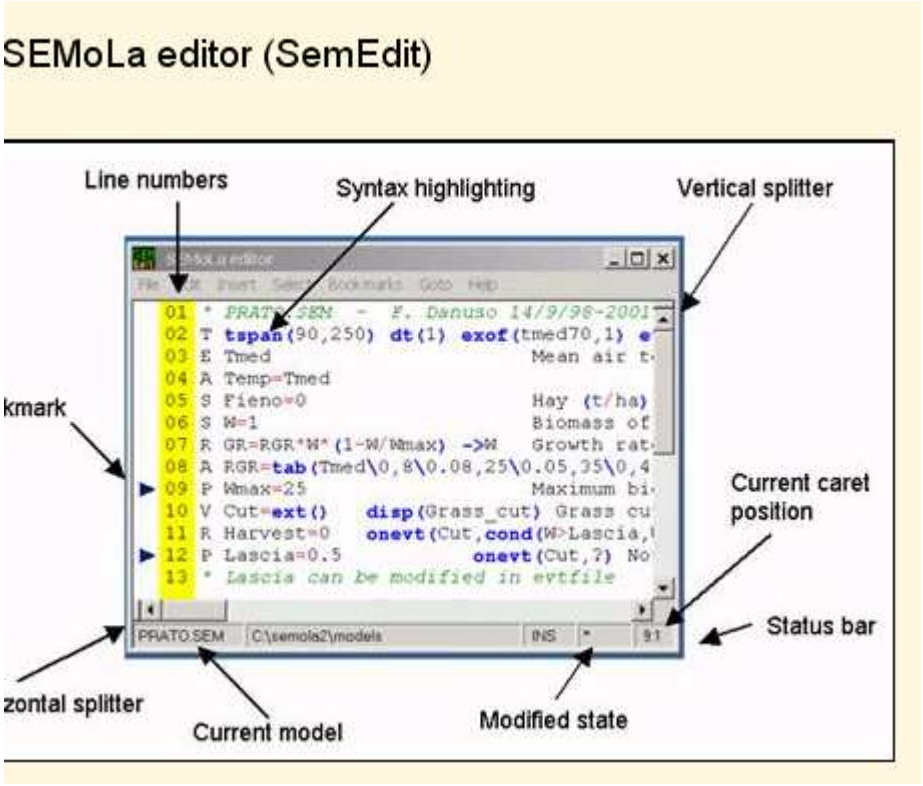
The framework



The console version



The SEMoLa editor (SemEdit)



COMMANDS

The tasks of SEMoLa can be obtained using dialogs of the GUI (Graphical user interface) or by commands. Commands are instruction given to the application to require a certain action. Commands can be used *interactively*: e.g., the user write the text of the command with its options, press Enter, see to the results of the action and continues the procedure with other commands. Commands offer the advantage to be used also in a *batch mode*, that is, the user write a list of commands (script) in a file and submit all the commands of the procedure in one step. Non need to wait for the result of each command. The procedure can be also repeated many times, even changing the input parameters.

The dialog of SEMoLa to input and manage the commands:

Command files

Using the command dialog above it is possible to create, manage and use lists of command, saved as command files (script). Command files are list of commands in a text file with **`cmf`** as name extension.

To create a command file from previously issued commands (those in the central window of the command dialog), insert a name in the combobox on the top left and press the button Save.

To edit a command file, select the command file in the combobox and press Edit.

To run a command file, select the command file and press Run.

Command files can be also managed by the commands **`csave`** and **`crun`**.

A command file can be launched also with parameters that can be used in the script code as macro ٪١%, ٪٢٪, ٪٣&, etc.

Language

A SEMoLa model is a text file formed by lines, each pertaining to a system element. Every line is identified by a letter as first word of the line. The SEMoLa compiler recognises nine types of statements: state declaration (S), auxiliary equations (A), rate equations (R), exogenous variable declaration (E), event declaration (V), parameter value assignment (P), run time options ($), section identifier (@) and comment line (‘) (Table ١).

The model code may be structured into logical sections. A section of the model is to be thought as a sub-system dealing with a particular material type of the system. The at-sign (@) declares a model section to be included in the current model.

Structure of the code line

```
id  name=expression  options  label (unit)
```

Meaning

class object value/relationship property metadata

Example model

```
* Crop growth model
S  C=5                              "Crop weight"       (t/ha)
R  CGR=RGR*C*(1-C/Cmax) ?->C        "Crop growth rate"  (t/ha/day)
P  RGR=0.1                          "Specific growth"   (1/day)
P  Cmax=150                         "Maximum weight"    (t/ha)
```

Simulation results

Satements of the SEMoLa language

State S declares a state of the system and initial value

Auxiliary	**A**	treats endogenous and exogenous information
Rate	**R**	declares a rate of the system, source and sink
Exovar	**E**	declares a needed exogenous variable (input)
Event	**V**	declares events and related actions on model
Parameter	**P**	declares value or expression for a parameter
Group	**G**	declares a group of elements
Options	**$**	options to be used as default in run-time phase
Section	**$**	indicates a new model section
Comment	**‘**	comments in the SEMoLa code

DATA STRUCTURES

SEMoLa uses the following data structures:

١) **Current dataset**: a table of data that can be saved on disk. It is like a table, with rows (time steps or observations or records) and columns (variables). Variables can be of types**: float** (double), **integer**, **string**.

٢) **Ambient variables**: are variables created and existing only in the time of a session. At the exit they are lost, unless they are converted to the current dataset and saved. They can be of **scalar**, **string** and **matrix** types.

٣) **Files on disks**

AMBIENT VARIABLES

Ambient variables are variables created and existing only in the time of a session. At the exit they are lost, unless they are converted to the current dataset and saved. They can be of scalar (a single numerical value, double precision), string (alphanumerical characters) and matrix (mono and bi dimensional arrays of double precision numbers) types and are created and managed by the commands `scalar, string and matrix`, respectively.
Many numerical and statistical commands, after been issued, leave in the ambient some specific variables that can be displayed again or reused by the next commands. These values are always rewritten by the successive use of the same commands. For example, the command
`. summarize` *varname*
generates a number of ambient variables that can be seen by:

```
. sca list
_nobs          =  ٢٠٦
_mean          =  ٣٣
_sum           =  ٦٧٩٨
_sd            =  ٠
_MS            =  ٠
_min           =  ٣٣
_max           =  ٣٣
```

Examples:

```
. scalar k=١٠              (creates a new scalar variable and set its value to ١٠)
. scalar b=٢               (creates another scalar variable and set its value to ٢)
. scalar c=b*k             (c is created and equals to ١٠)
. string abc="SEMoLa"
. string cde=mid("SEMoLa",٢،٣)   (the value of cde is "EMo")
```

Uses

With ambient variables is possible to:

١) perform calculations, even by complex expressions. Example: `sca x=sin(١٠-tan(٥٠))/(sqrt(٢٥))` (x = -.١٤٩٨٧٥٦٥٩٢١٨٤٥٤)
٢) re-use the results of the previous commands;
٣) perform matrix operations: Examples: `mat A=B·C` (matrix product); `mat C=A·trn(A)` (multiply a matrix by its transpose)
٤) use ambient variables in the generation of new variables of the current dataset:
   a. `scalar a=٥` (define the ambient variable a)
   b. `generate newvar=_i*a` (generate a new column with values ٥،١٠،١٥،٢٠،٢٥, ...; `_i` is the running index of rows);
٥) convert matrix into dataset by the command `matrix set;`
٦) convert a dataset to an ambient matrix by the command `matrix get;`
٧) Use the macro substitution. A macro in a expression or command is identified by the leading percent (%):
   a. `scalar a=٥`
   b. `string b="Myfile"+str(a)+".csv"` (concatenate strings)
   c. `use %b` (the command loads the file `Myfile٥.csv`)