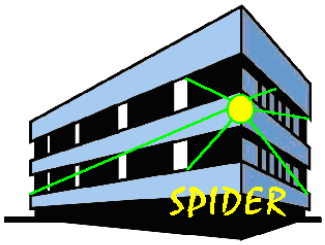
Development of analysis tools for structures protected by the damped cable system

European Commission
Energy, Environment
and Sustainable
Development



EVG1-CT-1999-00013 SPIDER

WORKPACKAGE 4 – TASK 4.1

Deliverable D8

Development of analysis tools for structures protected by the damped cable system

Stefano SORACE, Nicola ROVERE, Sebastiano SURACI
(Università di Udine)

Gloria TERENZI
(Università di Firenze)

EVG1-CT-1999-00013 SPIDER/4.1/ UDINE/SSO/02/VF-C

December 2001

Premise

This report sums up the activities carried out by Udine University within Task 4.1 of Work-package nr. 4. These activities concerned the improvement of the “j2d” software for the two-dimensional analysis of structures equipped with the damped cable system (DCS), and were particularly aimed at:

- providing greater computational capabilities compared to the first version of the code (among which: the incorporation of the vertical force components induced by cables; direct combinations of the static and dynamic output data; a substantial optimisation of the solving routines, so as to improve the numerical stability of the generated algorithms, and to remarkably reduce the elaboration times; etc);
- adding a newly conceived graphical user interface, to facilitate the use of the program both in input and post-processing phases.

Although many of these improvements have been already included, some important aspects are currently in progress. The general lines of the work to be still carried out are anyway completely planned, and presented in this report as well.

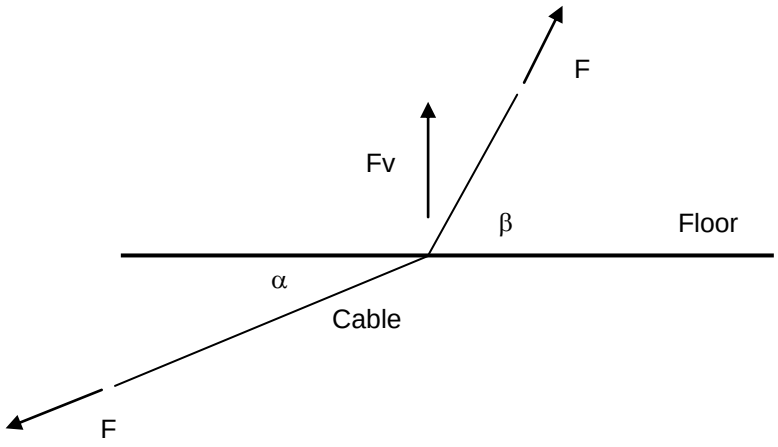
Section a – New analysis capabilities of j2d program

The j2d solver

The j2d solver has been enhanced in several ways, with the aim of expanding its capabilities integrating it closely with the new graphic user interface (GUI) front-end.

1. Improvements to core j2d solver

- **Vertical cable forces support**. Now j2d supports the evaluation of the vertical forces caused by cables. This has required a major reworking of the solver module.

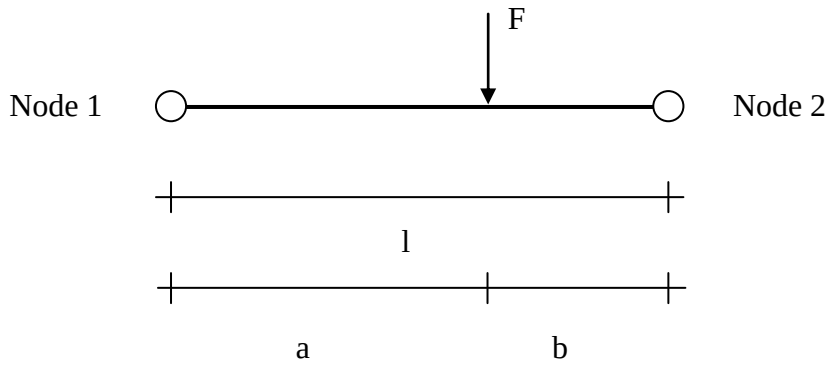


By referring to the figure above, the vertical force exerted by the cable at the floor level is given by:

$$F_v = F(sin\beta - sin\alpha)$$

The purpose of this improvement is to provide an evaluation of the stress state in beam elements caused by the vertical force components that arise in cable-floor intersections. As j2d was not initially designed to support this feature, it was important to accomplish its implementation by avoiding (or minimising) changes to the non-linear engine of the program. This is in fact the true "heart" of j2d, thoroughly tested and validated by comparative analyses conducted with other non-linear FE codes. In order to simplify the structural model, j2d does not include additional

degrees of freedom where cable and floor intersect each other. Moreover, to preserve model simplicity and to avoid the insertion of additional joints, the vertical force has been taken into account by a modified load vector for beam elements, which is explained in the figure below. For cases in which the cable intersects the beam element in proximity to a column, the vertical force is placed directly in the relevant beam-column joint.



The load vector for the beam element in the previous figure is:

$$L = F \cdot \begin{bmatrix} \dfrac{(3a+b)\cdot b^2}{l^3} \\ \dfrac{(a+3b)\cdot a^2}{l^3} \\ \dfrac{a\cdot b^2}{l^2} \\ \dfrac{a^2\cdot b}{l^2} \end{bmatrix}$$

The stress state induced by the vertical cable-forces is combined with the stress state generated by the non-linear analysis as follows:

$$\underline{S}_g(t) = \underline{S}_{nl}(t) + \sum_{i=o}^{n} F_c^i(t) \cdot \underline{F}_v^i$$

where:

- $\underline{S}_g(t)$ is the total stress vector at time t;
- $\underline{S}_{nl}(t)$ is the stress vector obtained by the non-linear analysis;
- $F_c^i(t)$ is the force of cable i at time t;
- $\underline{F}_v^i$ is the stress vector for cable i with unitary force.

The use of such a linear combination implies that the vertical force components do not alter the non-linear damped cable behaviour. This is acceptable in most cases, and makes it possible to avoid significant modifications to the non-linear kernel.

- **Stress and displacements conditions**. It is now possible to have outputs subdivided into the following categories:
  Analysis (static, eigenvalue, time-history);
  Load condition (static loads, ground acceleration, floor dynamic load).

- **Output data combination**. Stress states and displacements deriving from each load conditions are analysed and combined in order to make them easily browseable from the GUI front-end.
  Combination of output results is particularly important in time-history analysis since — once coupled with the use of a relational database in the GUI — enables users to easily query the results according to different criteria (e.g., maximum bending moment of a beam element during the first n seconds of ground motion, etc).

- **Optimisation of numerical-intensive routines**. Several optimisation provisions have been introduced in eigenvalue and time-history analysis so as to improve performance and stability.

2. Improvements to general j2d features and architecture

In order to allow a higher degree of integration between j2d solver and GUI front-end, some parts of the overall solver architecture have been redesigned. Much effort was produced to simplify software usage and to improve the quality of user experience. Some of the improvements listed below have not yet been completed, but are still under development.

- **Data-model refinement.** Data-model has been revamped in order to be compatible with GUI front-end. Details on tables, fields and relations are described in a following section.

- **Library and executable.** Now j2d is subdivided in two sections: a stand-alone executable and a dynamic library (DLL), which can be linked with graphic front-end for a closer interaction. This enables user feedback and interaction during analysis.

- **Data validation.** Input data validation is extremely important for a general use of the software, especially by non-experienced users. GUI front-end performs a preliminary check of user input, including range checking for fields and referential integrity constraints, whereas j2d engine validates the structural and numerical consistency of input data.

- **Error detection and reporting.** J2d now seamless handles error situations, and provides a detailed report so as to allow the user to correct input data accordingly.

- **Stability.** J2d is being extensively tested, with the help of memory debuggers, in order to avoid the stability problems that were found in the previous versions.

- **Upgrade of gigabase library**. An upgraded version of the gigabase library has been integrated in the solver.

Section b – New graphical interface of j2d program

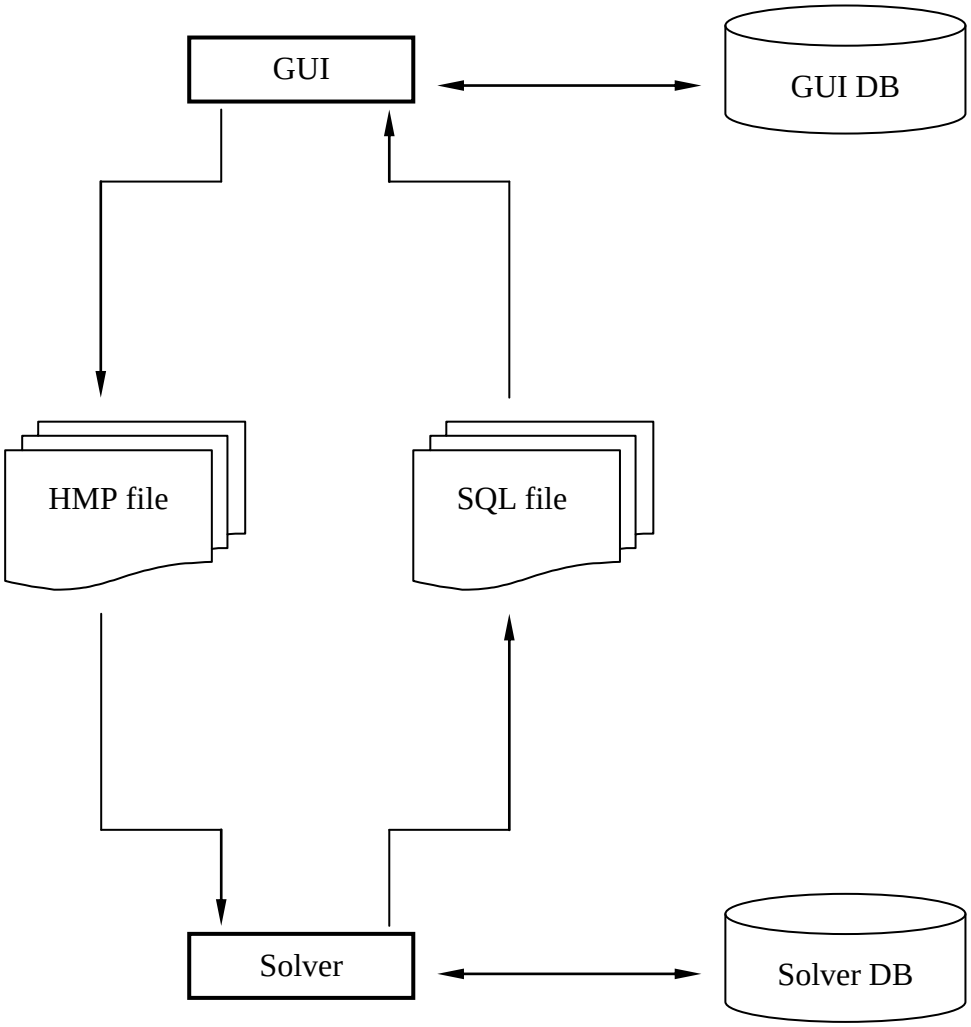
The j2d basic architecture

As anticipated in the premise, the j2d program has been split into two separated units: a solver and a graphical user interface (GUI).

The solver is a console application that can be regarded as an improved version of the previous j2d versions, with added new analysis functionalities and a new front-end. The GUI is a Windows 95/NT™ application, with its own widget's.

All the information regarding the model data and the analysis parameters can be defined or updated in the GUI, from which the solver can also be launched. Normally, the user should interact only with the GUI, but there is also the possibility of invoking directly the solver (although this option is discouraged).

Both the solver and the GUI have their own databases, which give the whole architecture room for remarkable improvability.
The GUI database keeps track of all the information about the model and the parameters of the analysis to be performed. When the user calls for a solution, the GUI writes down all relevant data into a communication file. This file is an ASCII one whose format is an enhanced version of the HMP format used by the previous j2d versions. When the file has been written, the GUI triggers the solver to solve the problem, and to write down the results into a second communication file. The latter consists of simple SQL statements, which will be finally read back by the GUI.
A schematic of the inter-communication process between GUI and solver is shown in Fig. 1. The solver database is used for internal purpose only.



Legend

GUI:	*j2d graphical user interface*
GUI DB:	*database owned by the GUI*
Solver:	*j2d solver*
Solver DB:	*database owned by the solver*
HMP file:	*GUI –t–solver data- transfer file (pre-processing phase)*
SQL file:	*solver–to–GUI dat- transfer file (post-processing phase)*

Fig. 1. Inter-communication process between graphical user interface (GUI) and solver in j2d.

The Graphical User Interface (GUI). Overview.

The GUI runs under operative systems belonging to the Microsoft Windows 95/98/ME, NT/2000 and XP (™) families. No other operative system is, and will be supported.

The application has been developed through the use of the Borland C++ Builder V5.0 Professional compiler. This programming tool can compile both C/C++ and Delphi code and is based on an Integrated Development Environment (IDE). Its greatest advantage consists in the remarkable benefits provided by its Visual Component Library (VCL) architecture, which lets the developer use third-part re-usable components. The result is a solid programming environment, which is quite effective in the fast development of 32-bits Windows applications.
In the following, explicit reference to some VCL classes will be made: a description of these classes can be found in the Borland C++ Builder manual (see http://www.borland.com). Examples of VCL class-names are “TForm” and “TTable”.

The application has been designed according to the Object Oriented Programming (OOP) paradigm. All the implemented data and functions belong to objects that interacts each other by following not ambiguous rules.

Basically, the GUI has been build around a Model View Controller (MVC) design pattern, which ensures separation between the data, and the interactions between data and user.

The Model “object” owns all the data (i.e., both model data and analysis parameters). The Model creates and initialises the structure of treated data; furthermore, only the Model can update, append or delete the data, in response to external triggers. No other object has direct access to the Model data, nor has knowledge about the structure of data. Nevertheless, the Model can un-hide parts of its data to other objects for a read-only access.
When a change in data occurs, the Model notifies this event to the Views, which are just “windows” on the data owned by the Model. These Views can be any representation of the data itself, like sheet tables, chart plots, drawings or similar entities. It’s up to a View to filter the event triggered by the Model ad to act consequently, i.e., it’s the View that knows what to represent and how, whereas the Model just triggers the event and lets the View to read relevant data.
The Model does know anything about the Views and how they work, except that it knows which are the Views that do actually exist. In fact, the Model must know at least to which View it is going to notify a data change. This is achieved through a registration mechanism, for which, in order to have the job done, each View must first register itself in a special list owned by the Model. In doing so, the only function required to the Model is to scan its list and to ask each attached View to update itself.
While the Views provide just a feedback to the end-user, the behaviour of the whole architecture is governed by means of the Controllers, which intermediate between the Model and the user. The

Controllers record all the relevant user-driven events (e.g., the insertion of a new record in a table or a mouse double-click) and, according to the actual context, they do ask the Model to update its data (and afterwards, in order to close the loop, to notify the changes to the Views).

There is often a one-to-one correspondence between Views and Controllers, i.e., most Views have their own Controller, and thus in many cases the distinction between one View and its Controller is more academic than practical. A flow-chart of the Model-Views-Controllers (MVC) mechanism is drawn in Fig 2.

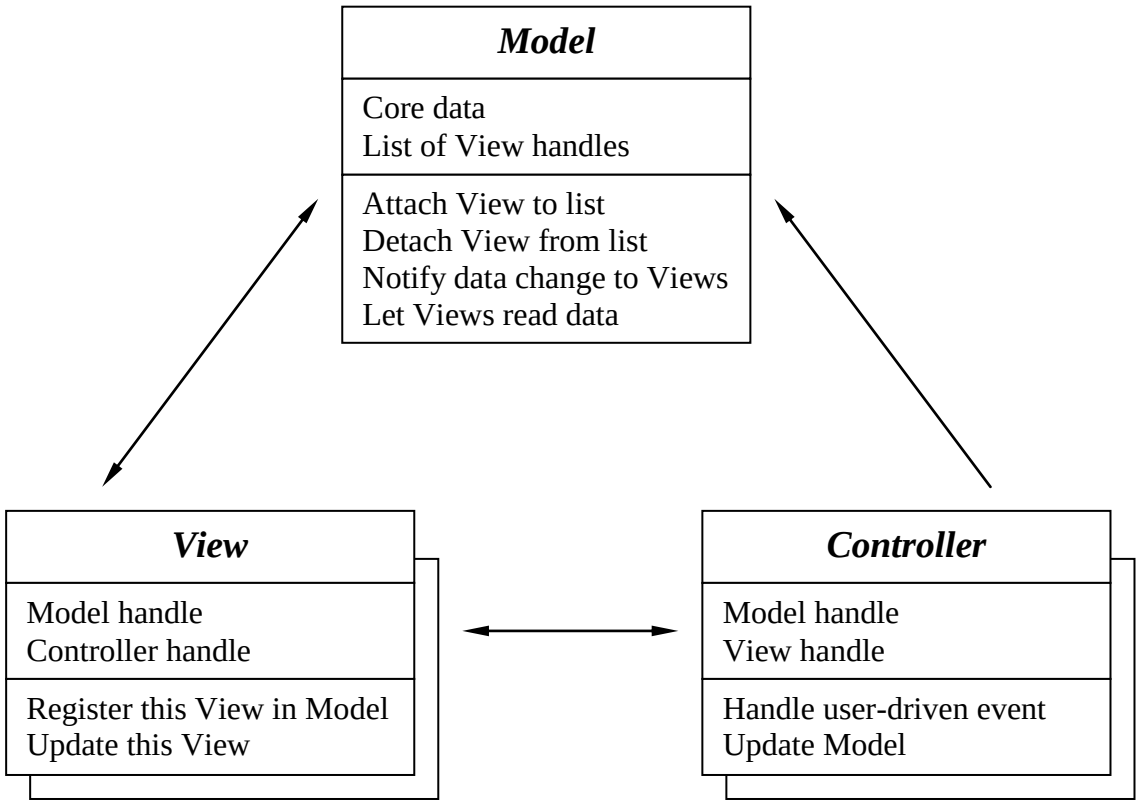


Fig. 2. Flow-chart of the Model-View-Controller pattern.

The MVC pattern ensures that new Views/Controllers can be easily added to the application preserving the Model data structure, and thus without remarkable changes in the code already written. Nevertheless, to achieve this goal, notable care must be taken in the first design of relevant objects, as well as of their interactions. Details are given in the following.

The Graphical User Interface (GUI). The Model data.

All the Model data are stored in a dedicated Paradox database. This database is accessed by the j2d GUI through the Borland Database Engine (BDE), which takes care of all the low-level database implementation details. At a higher level, the BDE, which acts as a collection of shared Application Programming Interfaces (API), can be controlled through specific extensions of the Borland C++ Builder language. This in turn allows obtaining several ways of data access: j2d GUI exploits both direct dataset access (TTable objects) and Structured Query Language (SQL) statements (TQuery objects). Details can be found in the Borland C++ Builder manual.

The GUI database is structured on tables, each one organised as a list of fields. The overall GUI database structure is quite similar to the one built for the j2d solver database. In fact, the GUI database can be viewed as an “extended” replica of the solver one: all the model data and analysis parameters are stored both in solver and GUI, but the latter has some additional fields, to facilitate the user in creating his models, and to allow some referential integrity mechanisms be implemented.
Referential integrity concerns the definition and the application of well-established rules that the user-generated data has to comply with (e.g., no beam or column element is allowed having a never defined material, etc). This has been achieved essentially through two mechanisms: the lookup fields and the referential integrity checks.
Lookup fields are special table fields whose values can be selected by the user only among a list of allowable values, picked from the actual entries of a reference field in another table. These fields, when activated, ensure that no illegal value can be an entry for a given field.
Referential integrity checks, on the other side, prevent further types of data corruption, such as the deletion of a record in a table containing a field value pointed by a lookup field.

The j2d GUI database is made up of the following tables.

1 *Table Name*	*Data set name*
StrTable	Model_Info
ParamTable	Model_Parameters
WireTable	Model_Wires
LevelTable	Model_Levels
MatTable	Model_Materials
SecTable	Model_Sections
BeamTable	Model_Beams
BeamElTable	Model_Beam_elements
ColTable	Model_Columns
ColElTable	Model_Column_elements
DamperTypeTable	Model_Dampers
CableTable	Model_Cables

LevelLoadTable	Model_Loads
GroundAccellTable	Model_Ground_acceleration
StatusTable	Results_Status
LevelShowTable	Results_Levels
BeShowTable	Results_Beam_elements
CeShowTable	Results_Column_elements
CableShowTable	Results_Cable

Therein the table names are coupled with the corresponding data set name.

Details about the structure of each table are given in the following (see Borland C++ Builder user manual for explanations).

1.1 Data set: Model_Info

Field name	***Displayed as***	***Field data type***
id	id	ftAutoInc
name	name	ftString

Index of type [ixPrimary,ixUnique] on field: id

1.1.1 Data set: Model_Parameters

Field name	***Displayed as***	***Field data type***
id	id	ftAutoInc
name	name	ftString
value	value	ftString

Index of type [ixPrimary,ixUnique] on field: id

1.1.2 Data set: Model_Wires

Field name	***Displayed as***	***Field data type***
id	id	ftAutoInc
name	name	ftString

x	x	ftFloat
y	y	ftFloat

Index of type [ixPrimary,ixUnique] on field: id

1.1.3 Data set: Model_Levels

Field name	*Displayed as*	*Field data type*
id	id	ftAutoInc
name	name	ftString
restr	restr	ftBoolean
z	z	ftFloat
mass	mass	ftFloat

Index of type [ixPrimary,ixUnique] on field: id

1.1.4 Data set: Model_Materials

Field name	*Displayed as*	*Field data type*
id	id	ftAutoInc
name	name	ftString
e	e	ftFloat
ni	ni	ftFloat
w	w	ftFloat

Index of type [ixPrimary,ixUnique] on field: id

1.1.5 Data set: Model_Sections

Field name	*Displayed as*	*Field data type*
id	id	ftAutoInc
name	name	ftString
a	a	ftFloat
i11	i11	ftFloat

Index of type [ixPrimary,ixUnique] on field: id

1.1.6 Data set: Model_Beams

Field name	*Displayed as*	*Field data type*
id	id	ftAutoInc
name	name	ftString
wire_id_start	wire id start	ftInteger
wire_id_end	wire id end	ftInteger
level_id	level id	ftInteger
mat_id	mat id	ftInteger
sec_id	sec id	ftInteger
load	load	ftFloat

```
Index of type [ixPrimary,ixUnique] on field: id

Lookup field applied to field: wire_id_start
       data set              : Model_Wires
       key field             : id
       key result field      : name
       field display name    : Start wire
Lookup field applied to field: wire_id_end
       data set              : Model_Wires
       key field             : id
       key result field      : name
       field display name    : End wire
Lookup field applied to field: level_id
       data set              : Model_Levels
       key field             : id
       key result field      : name
       field display name    : Level
Lookup field applied to field: mat_id
       data set              : Model_Materials
       key field             : id
       key result field      : name
       field display name    : Material
Lookup field applied to field: sec_id
       data set              : Model_Sections
       key field             : id
       key result field      : name
       field display name    : Section
```

1.1.7 Data set: Model_Beam_elements

Field name	***Displayed as***	***Field data type***
id	id	ftAutoInc
beam_id	beam id	ftInteger
wire_id_start	wire id start	ftInteger
wire_id_end	wire id end	ftInteger
level_id	level id	ftInteger
mat_id	mat id	ftInteger
sec_id	sec id	ftInteger
load	load	ftFloat

```
Index of type [ixPrimary,ixUnique] on field: id
Index of type [ixCaseInsensitive] on field: beam_id

Lookup field applied to field: wire_id_start
       data set              : Model_Wires
       key field             : id
       key result field      : name
       field display name    : Start wire
Lookup field applied to field: wire_id_end
       data set              : Model_Wires
       key field             : id
       key result field      : name
       field display name    : End wire
Lookup field applied to field: level_id
       data set              : Model_Levels
       key field             : id
       key result field      : name
       field display name    : Level
Lookup field applied to field: mat_id
       data set              : Model_Materials
       key field             : id
       key result field      : name
       field display name    : Material
Lookup field applied to field: sec_id
       data set              : Model_Sections
       key field             : id
       key result field      : name
       field display name    : Section
```

1.1.8 Data set: Model_Columns

Field name	*Displayed as*	*Field data type*
id	id	ftAutoInc
name	name	ftString
wire_id	wire id	ftInteger
level_id_start	level id start	ftInteger
level_id_end	level id end	ftInteger
mat_id	mat id	ftInteger
sec_id	sec id	ftInteger

```
Index of type [ixPrimary,ixUnique] on field: id

Lookup field applied to field: wire_id
       data set              : Model_Wires
       key field             : id
       key result field      : name
       field display name    : Wire
Lookup field applied to field: level_id_start
       data set              : Model_Levels
       key field             : id
       key result field      : name
       field display name    : Start level
Lookup field applied to field: level_id_end
       data set              : Model_Levels
       key field             : id
       key result field      : name
       field display name    : End level
Lookup field applied to field: mat_id
       data set              : Model_Materials
       key field             : id
       key result field      : name
       field display name    : Material
Lookup field applied to field: sec_id
       data set              : Model_Sections
       key field             : id
       key result field      : name
       field display name    : Section
```

1.1.9 Data set: Model_Column_elements

Field name	*Displayed as*	*Field data type*
id	id	ftAutoInc
col_id	col id	ftInteger
wire_id	wire id	ftInteger

```
level_id_start          level id start          ftInteger
level_id_end            level id end            ftInteger
mat_id                  mat id                  ftInteger
sec_id                  sec id                  ftInteger

Index of type [ixPrimary,ixUnique] on field: id
Index of type [ixCaseInsensitive] on field: col_id

Lookup field applied to field: wire_id
       data set              : Model_Wires
       key field             : id
       key result field      : name
       field display name    : Wire
Lookup field applied to field: level_id_start
       data set              : Model_Levels
       key field             : id
       key result field      : name
       field display name    : Start level
Lookup field applied to field: level_id_end
       data set              : Model_Levels
       key field             : id
       key result field      : name
       field display name    : End level
Lookup field applied to field: mat_id
       data set              : Model_Materials
       key field             : id
       key result field      : name
       field display name    : Material
Lookup field applied to field: sec_id
       data set              : Model_Sections
       key field             : id
       key result field      : name
       field display name    : Section
```

1.1.10 Data set: Model_Dampers

Field name	***Displayed as***	***Field data type***
id	id	ftAutoInc
name	name	ftString
c	c	ftFloat
p_damper	p damper	ftFloat
k1	k1	ftFloat
k2	k2	ftFloat
alpha	alpha	ftFloat

Index of type [ixPrimary,ixUnique] on field: id

1.1.11 Data set: Model_Cables

Field name	*Displayed as*	*Field data type*
id	id	ftAutoInc
name	name	ftString
level_id_start	level id start	ftInteger
level_id_end	level id end	ftInteger
mat_id	mat id	ftInteger
sec_id	sec id	ftInteger
damper_type_id	damper type id	ftInteger
p_cable	p cable	ftFloat
x_0	x 0	ftFloat
x_1	x 1	ftFloat
x_2	x 2	ftFloat
x_3	x 3	ftFloat
x_4	x 4	ftFloat
x_5	x 5	ftFloat
x_6	x 6	ftFloat
x_7	x 7	ftFloat
x_8	x 8	ftFloat
x_9	x 9	ftFloat
x_10	x 10	ftFloat
x_11	x 11	ftFloat
x_12	x 12	ftFloat
x_13	x 13	ftFloat
x_14	x 14	ftFloat
x_15	x 15	ftFloat
x_16	x 16	ftFloat
x_17	x 17	ftFloat
x_18	x 18	ftFloat
x_19	x 19	ftFloat

Index of type [ixPrimary,ixUnique] on field: id

Lookup field applied to field: level_id_start
       data set   : Model_Levels
       key field   : id
       key result field   : name
       field display name   : Start level
Lookup field applied to field: level_id_end
       data set   : Model_Levels
       key field   : id
       key result field   : name

```
       field display name    : End level
Lookup field applied to field: mat_id
       data set              : Model_Materials
       key field             : id
       key result field      : name
       field display name    : Material
Lookup field applied to field: sec_id
       data set              : Model_Sections
       key field             : id
       key result field      : name
       field display name    : Section
Lookup field applied to field: damper_type_id
       data set              : Model_Dampers
       key field             : id
       key result field      : name
       field display name    : Damper
```

1.1.12 Data set: Model_Loads

Field name	***Displayed as***	***Field data type***
id	id	ftAutoInc
level_id	level id	ftInteger
load	load	ftFloat
time	time	ftFloat

```
Lookup field applied to field: level_id
       data set              : Model_Levels
       key field             : id
       key result field      : name
       field display name    : Level
```

1.1.13 Data set: Model_Ground_acceleration

Field name	***Displayed as***	***Field data type***
id	id	ftAutoInc
time	time	ftFloat
accell	accell	ftFloat

1.1.14 Data set: Results_Status

Field name	*Displayed as*	*Field data type*
id	id	ftAutoInc
activity_id	activity id	ftInteger
status	status	ftInteger
msg	msg	ftString

1.1.15 Data set: Results_Levels

Field name	*Displayed as*	*Field data type*
analysis_id	analysis id	ftInteger
level_id	level id	ftInteger
cond_id	cond id	ftInteger
time	time	ftFloat
disp	disp	ftFloat
accel	accel	ftFloat
force	force	ftFloat
t_sup	t sup	ftFloat
t_inf	t inf	ftFloat

1.1.16 Data set: Results_Beam_elements

Field name	*Displayed as*	*Field data type*
be_id	be id	ftInteger
analysis_id	analysis id	ftInteger
cond_id	cond id	ftInteger
time	time	ftFloat
m_step_0	m step 0	ftFloat
m_step_1	m step 1	ftFloat
m_step_2	m step 2	ftFloat
m_step_3	m step 3	ftFloat
m_step_4	m step 4	ftFloat
m_step_5	m step 5	ftFloat
m_step_6	m step 6	ftFloat
m_step_7	m step 7	ftFloat
m_step_8	m step 8	ftFloat

m_step_9	m step 9	ftFloat
m_step_10	m step 10	ftFloat
m_step_11	m step 11	ftFloat
m_step_12	m step 12	ftFloat
m_step_13	m step 13	ftFloat
m_step_14	m step 14	ftFloat
t_step_0	t step 0	ftFloat
t_step_1	t step 1	ftFloat
t_step_2	t step 2	ftFloat
t_step_3	t step 3	ftFloat
t_step_4	t step 4	ftFloat
t_step_5	t step 5	ftFloat
t_step_6	t step 6	ftFloat
t_step_7	t step 7	ftFloat
t_step_8	t step 8	ftFloat
t_step_9	t step 9	ftFloat
t_step_10	t step 10	ftFloat
t_step_11	t step 11	ftFloat
t_step_12	t step 12	ftFloat
t_step_13	t step 13	ftFloat
t_step_14	t step 14	ftFloat

1.1.17 Data set: Results_Column_elements

Field name	***Displayed as***	***Field data type***
ce_id	ce id	ftInteger
analysis_id	analysis id	ftInteger
cond_id	cond id	ftInteger
time	time	ftFloat
f_0	f 0	ftFloat
f_1	f 1	ftFloat
f_2	f 2	ftFloat
f_3	f 3	ftFloat
f_4	f 4	ftFloat
f_5	f 5	ftFloat

1.1.18 Data set: Results_Cable

Field name	*Displayed as*	*Field data type*
analysis_id	analysis id	ftInteger
cable_id	cable id	ftInteger

cond_id	cond id	ftInteger
time	time	ftFloat
force	force	ftFloat
damper_disp	damper disp	ftFloat
damper_speed	damper speed	ftFloat
damper_fc	damper fc	ftFloat

Currently, j2d input data and j2d analysis results are stored in the same GUI database. In fact, it's up to the GUI Views to hide or un-hide the data to the user, according to the actual context.

The previously outlined database structure may be subjected to minor adjustments during further developments of the j2d code, but it can be considered as very near to the ultimate one.

All the GUI database tables are created by the Model object at runtime, when the Model object itself is being created. During the initialisation phase, any previous instance of the database is erased by the program, ensuring data congruence during each j2d session.
Once all the tables have been successfully created and their data initialised (to "empty"), j2d can start the Model View Controller loop.

With the Model View Controller loop running, the Model object allows data access to Controllers objects either by means of methods which are native in the VCL components used as Controllers, either through specific methods built by j2d developers for this project.
Basically, each table in the j2d GUI database corresponds to a TTable class object, which in turn is provided also with methods to append, insert, delete or update records. These methods can be called by the application itself or by the user. In the latter case, the TTable class object responds to triggers raised by the visual components (View/Controller) currently active (typically, visual components are sheets like grids).
The triggers are dispatched by means of TDataSource class objects, which acts as links between the non-visual TTable objects and the visual View/Controller objects. These links ensure that different TTable objects can solely refer to the same View/Controller, i.e., the user can use just one visual component to modify more than one table, one at a time.
Other powerful tools exploited in the j2d application in order to manipulate the Model data are the TQuery class objects. These objects are able to perform queries to the database according to SQL statement strings, and are extensively used when loading or saving j2d sessions from/to files; when data must be evaluated and written into the database by the application; and finally, when data must be filtered and extracted from the database itself.

The Graphical User Interface (GUI). Forms

Forms can be referred as the visual windows of the GUI , which contain whatever widget the user needs to control the application (e.g., menus, buttons, grids, charts, etc).

In j2d, any form object derives from the same TForm basic class. Most of them are instantiated once the application starts (i.e., they persist during almost all the lifetime of the application), even if they can appear or can be hidden to the user according to the context.

The content of the forms is currently not completely fixed since some secondary (but helpful) functions have not been implemented yet. Therefore, in the following only an introductory and not exhaustive description is given of these functions.

The main form of the application, which contains the main Controllers, is the first one to be activated at program start-up. Its appearance is shown in Fig. 3, where the lookup mechanism can also be observed.

Basically, it owns a menu, a speed buttons bar, a dataset browser, one or two sheet like grids and a status bar

Through the menu, the user can govern the basic behaviour of the application, that is:

- open a new j2d session;
- open from file a previously saved session;
- save the current session to file;
- export it to file (currently only HMP format is supported);
- define printer page set-up;
- print data;
- launch the solver;
- ask for a sketch of the model and the deformed shapes (not implemented yet);
- ask for an element stress chart plot;
- launch a console session;
- get information/help;
- exit from application.

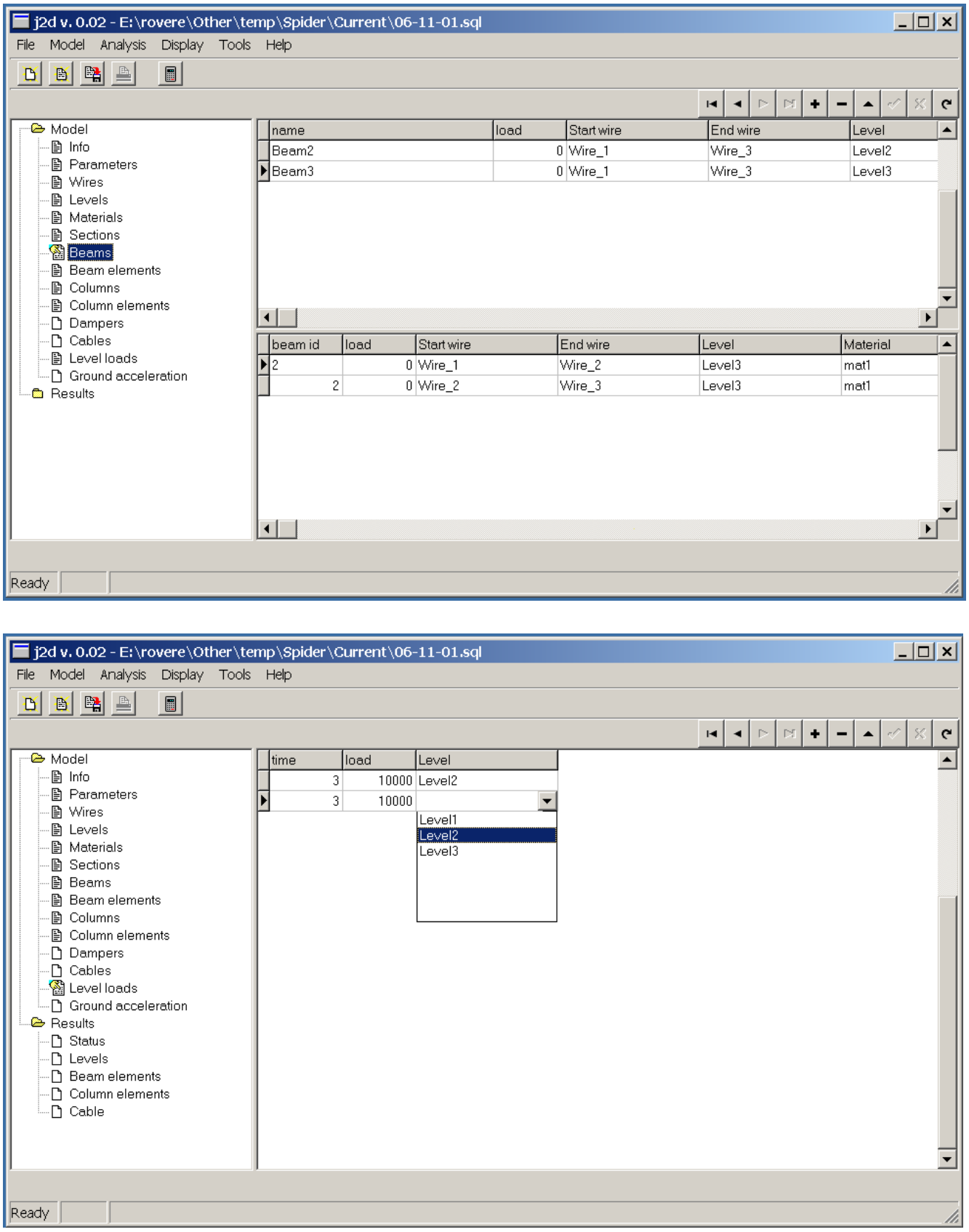


Fig. 3. The main form in action.

The speed buttons just reply the functions of the most used menu options.

The dataset browser (placed at the left of the form) is one of the most important View/Controllers of the application, because it allows the user to select the data set (i.e., the table) that is going to work with. It appears as a tree view widget and the selection is performed via a simple double-click at the icon corresponding to the desired dataset; this event is tracked by the application which re-acts updating the grid View/Controller, that will be finally linked to the chosen table

The grid — placed at the right — is another basic widget of the form, because it is the place where the user can append, insert, update or delete records. Its functions are integrated with the ones of a database navigator widget, which is an helpful toolbar to navigate into the grid sheet.
Just one table can be visible at one time (Fig. 3), except if beams or columns data sets are active. In such cases a master grid and a sleeve grid appear to the user (Fig. 3), The master one contains the beams (columns) definitions, whereas the sleeve one shows the beam elements (column elements) specifications.
The master/sleeve mechanism implemented in j2d ensures that when the cursor on the master grid points to a record, only the entries in the sleeve grid referring to that record are displayed.

The status bar shows the application status, helping the user in understanding the context in which the next operation will be performed.

The analysis general parameters can not be accessed by means of the grid mechanism, but through a dedicated form (Fig. 4).

The main form can be used also in the post-processing phase of an analysis session, letting the user query the solver results in tabular forms.
An improvement to these basic post-processing capabilities is provided by means of complementary forms, among which the plot-form. This can be activated through the main form, and its purpose is to display a diagram plot of the stress states occurring in each element (beam, column, cable) of the model.
This form (Fig. 5) owns a menu, a speed-buttons panel, a TChart VCL component, a control panel and a status bar. Its most important View/Controllers are the chart, able to display formatted plots of one-dimensional functions, and the control panel which specifies what must be displayed.
The latter widget lets the user choice about:

- the analysis performed (static, modal, time-history);
- the element stress (axial force, shear force, bending moment);
- the load state (envelop, mode, time step).

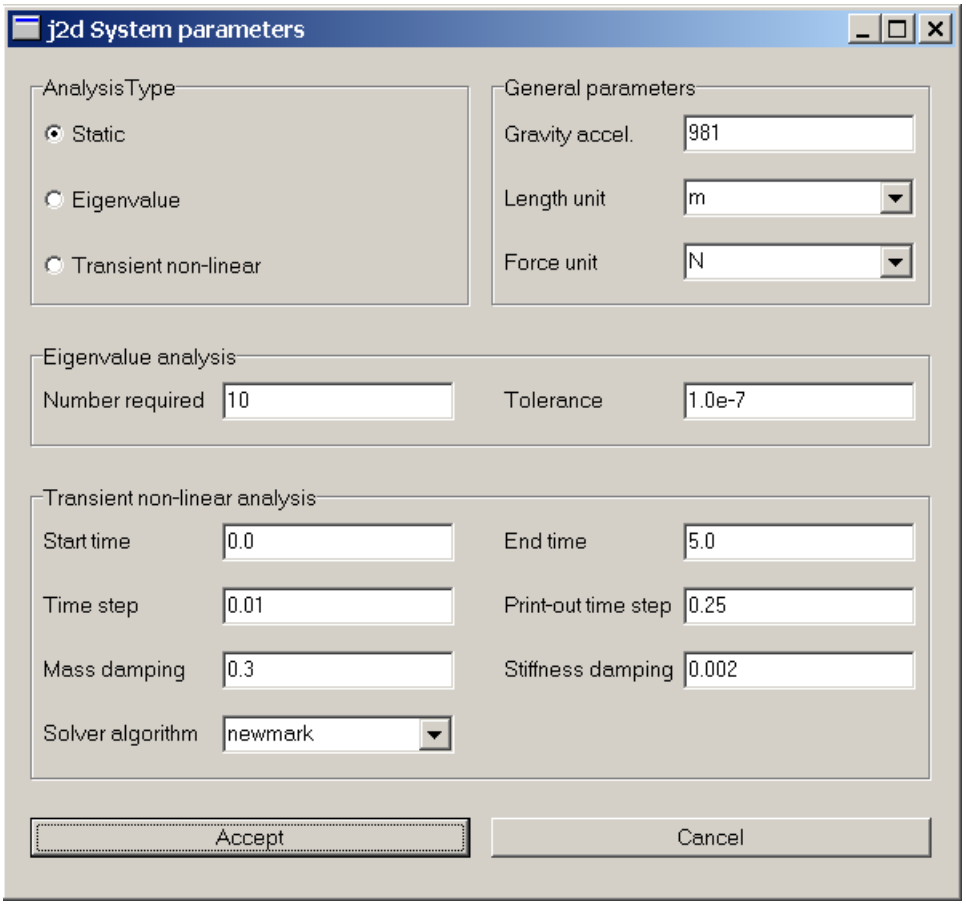


Fig. 4. The form relevant to the definition of the analysis parameters.

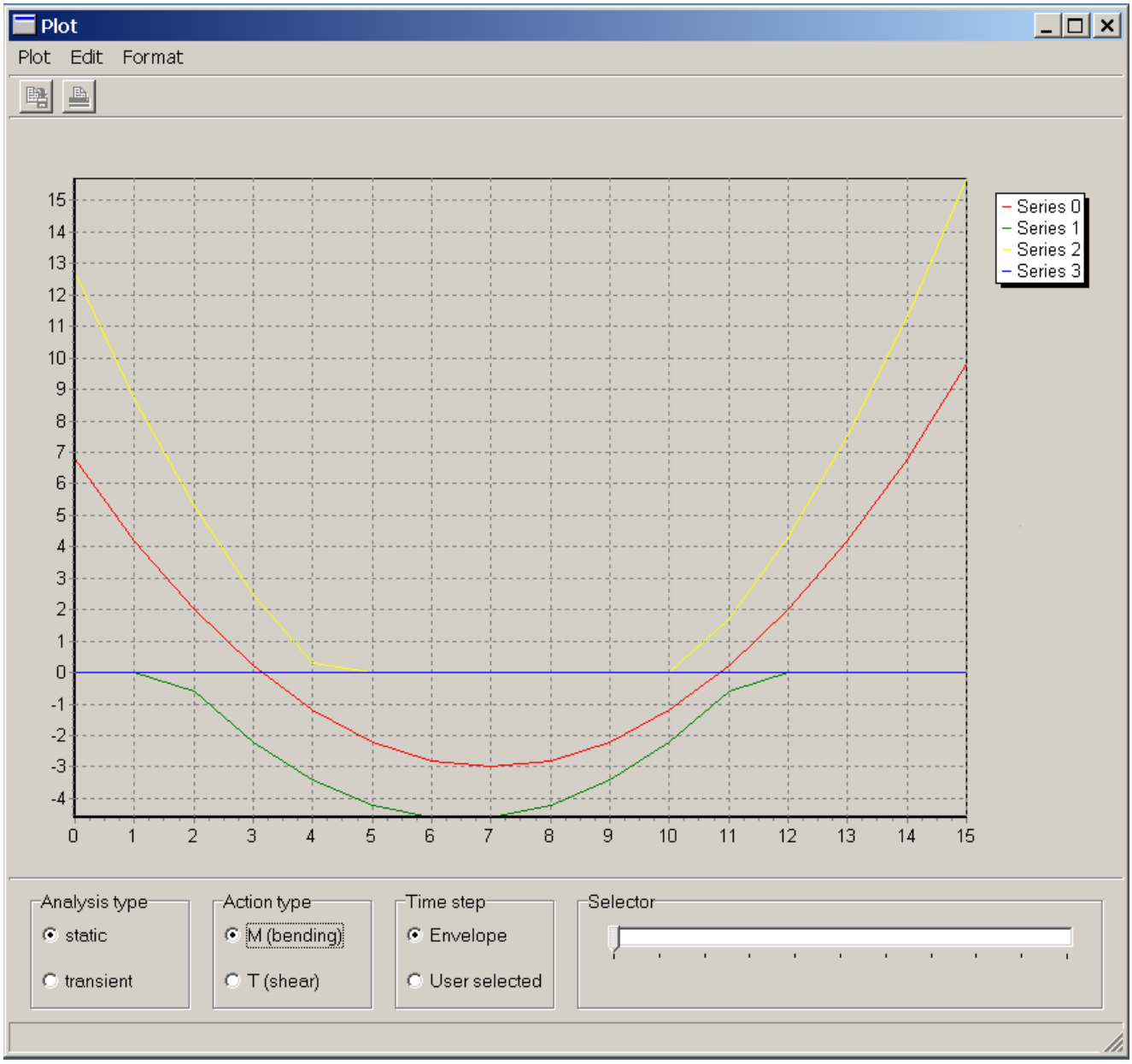


Fig. 5. The plot-form.

The Graphical User Interface (GUI). Development status

Up to date (half of December, 2001), the j2d GUI development state can be summarised as follows:

the Model View Controller implementation is stable and only few features will be added, related to the new Controller/Views that will be attached to the application;

the Model data-management implementation is basically stable and some additional job must still be done only to achieve even more robust Referential Integrity capabilities; minor improvements will be introduced in order to improve formal data representation;

the Main Form implementation is stable and adjustments will be operated only to improve aesthetic requirements and to add secondary features; this form already interfaces effectively to the solver application, both in writing model data and in reading analysis results;

the Plot Form development is under progress but almost completed; major changes may be suggested by user impression's feedback; activity is expected to be completed at the end of January 2001;

the Draw Form, expected to display 2D drawings of the finite element model as well as of the deformed shapes is currently under first development; this module will be based on the OpenGL technology; activity is expected to be completed at the end of February 2001;

the Template Form development will start at the beginning of February, 2002; this form will provide the user with automated model generation tools; this is a low-level risk activity because based on already existing code resources; it is expected to be completed at the end of February 2001;

the help and documentation files development will start at the beginning of the March, 2002; this is a very low-level risk activity and it is expected to be completed at the end of March 2001.

Development activities comprise debugging and testing of the single modules and of the whole application.

Up to date the j2d GUI application already guarantees all the basic visual features needed to create a model; to define the analysis parameters; to run the solver, to read the results; and to save the session to file. At the same time, some job must still be done in order to hide some remaining implementation details of the j2d solver, as well as to enhance the data representation.