A software tool for the analysis, design and monitoring of cable-stayed bridges

Originally published in:
IBK Bericht 286

Dissertation ETH No.15603

A Software Tool for the Analysis, Design and Monitoring of Cable-Stayed Bridges

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH
for the degree of
Doctor of Sciences ETH

presented by

Pietro Pedrozzi
Dipl. Bau-Ing. ETH
born 21 November 1975
citizen of Pregassona TI

accepted on the recommendation of

Prof. Dr. Edoardo Anderheggen, examiner
Prof. Dr. sc. techn. Mario Fontana, co-examiner
Dr. sc. techn. Mike Schlaich, co-examiner

2004

Abstract

Of the newly-built bridges, cable-stayed bridges are today very common worldwide for spans ranging between 200 and 900 meters. Being mostly built by the cantilever method, cable-stayed bridges have to be analyzed separately in every construction stage, taking into account many load cases. In addition, the forces in the cable stays have to be determined and possibly changed during construction so as to obtain the desired deck and mast deflections during service. Some non-linear and execution-dependent effects - such as concrete creep, shrinkage and locked-in bending in composite steel-concrete decks - have to be considered in the case of longer spans.

Conventional structural analysis software cannot automatically handle these specific problems, which today in many cases require the program user to manually process data. The goal of this project is the development of a specially tailored program for the structural analysis of cable-stayed bridges, which automatically handles the aforementioned problems. The program has been designed to be used in all stages of planning and construction: preliminary design, detail planning, construction process and retrofitting.

The program BRIDE is characterized by some newly introduced concepts allowing one to take the aforementioned non-linear effects into account: the chronologically ordered list of model objects accurately representing the complete construction process, which has to be defined in the input by the user, and the stage-by-stage iteration in which all construction stages are simulated in chronological order.

The program's functionality can be summarized as follows: the user first writes a text input file according to the program's input syntax, which allows programming language-like constructs such as variable declarations, for-loops, if-tests, and expression evaluations for input parameterization. The user then opens the input file from the program (this file can also be opened while it is being written to check its correctness). All calculations can then be performed interactively through the user graphic interface. The model and the results for any desired construction stage appear on the screen in three-dimensional graphics or text form. Snapshots with both graphics and text showing model and results can be stored in a highly portable Html-document. Automatic dimensioning of cables and the calculation of required pre-camber in the deck and mast segments and of post-tensioning forces in cables are possible. As such calculations change the model itself, the updated model can be stored in a new input file reflecting the content of the original one and the changes made to it. All the way from preliminary design to the construction phase the bridge designer works on the same input file and, as the planning gets more detailed, refines it either manually or automatically.

As discussed in the conclusions at the end of this thesis the program BRIDE, at least

in its present form, is not to be viewed as a commercial software product. In fact, the main scope of this research work was to study original approaches to an important and non-trivial practical problem.

Zusammenfassung

Unter den neu gebauten Brücken sind Schrägseilbrücken weltweit sehr beliebt für Spannweiten zwischen 200 und 900 Metern. Solche Brücken werden meistens in Frei-Vorbau gebaut und müssen deshalb in allen Bauzuständen unter vielen Lastfällen untersucht werden. Zudem müssen die Kabelvorspannkräfte ermittelt und eventuell während des Baus geändert werden, um die erwünschte verschobene Lage der Decke und des Mastes im Gebrauchszustand zu erhalten. Für längere Spannweiten sollen einige nichtlineare- und ausführungsabhängige Effekte wie zum Beispiel Kriechen, Schwinden und eingefrorene Anfangsverschiebungen in Verbunddecken berücksichtigt werden.

Konventionelle Baustatikprogramme können solche spezifischen Probleme nicht behandeln, und sie zwingen heutzutage den Benützer die Daten manuell zu manipulieren. Das Ziel dieses Projektes ist die Entwicklung eines Programms, das speziell für die statische Analyse von Schrägseilbrücken konzipiert ist und welches die erwähnten Probleme automatisch behandeln kann. Das Programm wurde entworfen um in jeder Planungs- und Bauphase eingesetzt zu werden: Entwurf, Detailplanung, Bau und Sanierung.

Das Programm BRIDE ist gekennzeichnet durch einige neu eingeführte Konzepte welche es erlauben, die erwähnten nichtlineare Effekte zu berücksichtigen: die chronologisch geordnete Liste von Modellobjekte, welche vom Benützer in der Eingabedatei definiert wird, und die schrittweise Iteration über die Bauzustände, die sämtliche Bauzustände in chronologischer Ordnung simuliert.

Die Arbeitsweise des Programms kann wie folgt zusammengefasst werden: der Benützer schreibt zuerst eine Textdatei gemäss der Eingabesyntax des Programms, welche programmiersprachenähnliche Konstrukte wie Variablendeklarationen, for-Schleifen, if-Prüfungen und Ausdruckevaluationen für die Parametrisierung der Eingabe zulässt. Danach öffnet er die Eingabedatei vom Programm aus (die Eingabedatei kann auch geöffnet werden, wenn sie noch nicht fertiggeschrieben wurde, um ihre Korrektheit zu überprüfen). Sämtliche Berechnungen können interaktiv über die graphische Oberfläche ausgeführt werden. Das Modell und seine Resultate erscheinen als dreidimensionales Bild oder als Text für jeden beliebigen Bauzustand und Lastfall auf dem Bildschirm. Schnappschüsse der graphischen oder der Textausgabe können mit dem Modell und seinen Resultaten als portables Html-Dokument gespeichert werden. Eine automatische Kabelbemessung und die Ermittlung sämtlicher benötigten Vorspannkräfte und Überhöhungen können ausgeführt werden. Sofern solche Berechnungen das Modell ändern, kann das aktualisierte Modell als eine neue Eingabedatei abgespeichert werden, welche der ursprünglichen Eingabedatei entspricht plus die vom Programm ermittelten Änderungen. Während des ganzen Prozesses vom Entwurf bis zum Bau arbeitet der Benützer immer

mit der gleichen Eingabedatei und, wenn die Planung detaillierter wird, verfeinert er sie entweder manuell oder automatisch.

Wie in den Schlussfolgerungen dieser Dissertation erwähnt, soll das Programm BRIDE, wenigstens in seiner jetzigen Form, nicht als kommerzielles Produkt angesehen werden. Das Hauptziel dieser Forschungsarbeit war nämlich, originelle Ansätze für ein wichtiges und nicht triviales praktisches Problem zu erforschen.

Contents

1. Introduction

1.1. The cable-stayed bridge design problem

In cable-stayed bridges the deck is supported at more or less regular distances by cables which are fixed to the top or along a mast protruding from the deck plane. In most cases cable-stayed bridges are self-anchored, i.e. the normal force introduced in the deck by the cables on one side of a mast is compensated by the normal force introduced on the other side.

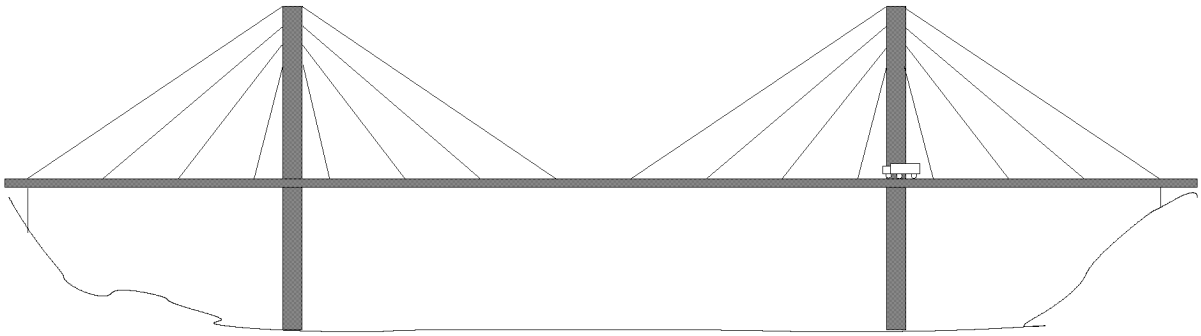


Figure 1.1.: A typical cable-stayed bridge.

The main advantages of cable-stayed bridges are that they can be built with very large spans (today with a central span of up to 900 meters) by free cantilevering (see figure 1.2), provide a large stiffness, need little material and can look quite elegant. Despite these advantages, cable-stayed bridges only became more widespread in the nineteen-fifties because it was impossible to analyze them with a reasonable effort and a satisfactory accuracy using the manual methods of pre-computer days.

The reason why the design and erection of cable-stayed bridges is so computationally intensive is that they are statically indeterminate to a degree which is approximatively as large as the number of cables and, at the same time, because their displacements and section forces are adjustable through the post-tensioning of each single cable. In addition, long span cable-stayed bridges are subject to non-linear phenomena such as cable sag and concrete creep.

The goal of adjusting the post-tensioning force in each cable and of mounting pre-cambered mast and deck segments is to have a finished structure which reflects the planned geometry. Even though cable-stayed bridges are globally very stiff, single members can be quite flexible and produce large displacements, especially during construction.

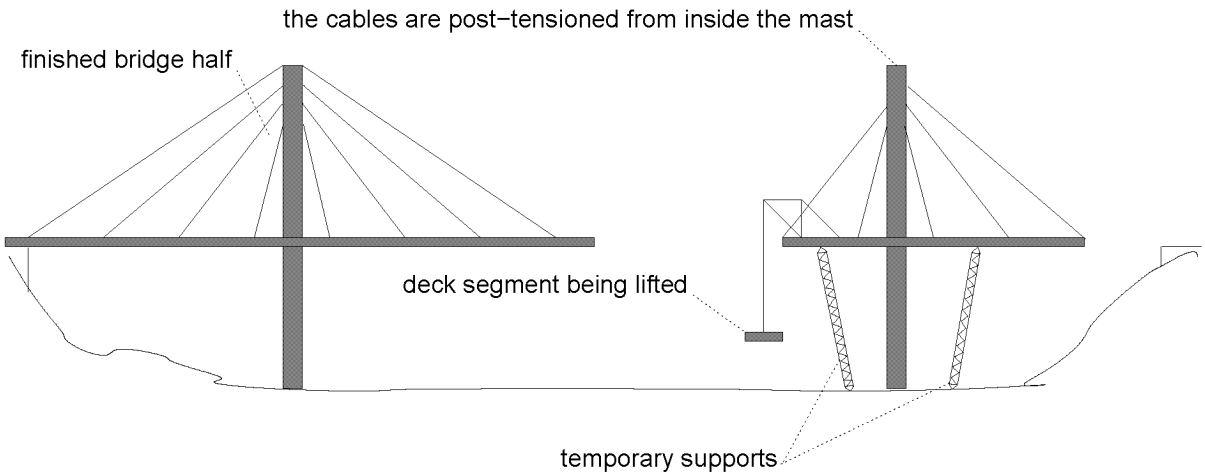


Figure 1.2.: Bridge erection by free cantilevering (analogous to the Uddevalla bridge on the cover page).

Once statical analysis for several construction stages and for the final service state has been carried out, computations might still be needed during erection in combination with measurements in situ in order to correct alignment errors and to keep the rather unpredictable behaviour of concrete and supports under close control.

1.2. Use of standard structural analysis software

Cable-stayed bridges can, in principle, be successfully analyzed with standard structural analysis software. This however is not straightforward. Standard software, in general, cannot automatically determine the post-tensioning forces needed in the cable-stays or the pre-camber values for the mast and deck segments. In many cases these still have to be found by the bridge designer by means of time-consuming and error-prone manual computation. Here are a few possible strategies:

- Insertion of fictitious hinges in the mast so as to find cable forces for which the mast is not subjected to bending moments.
- Use of very high artificial stiffnesses for the normal force, to avoid axial displacements in the deck and the pylons.
- Use of supports instead of cables to obtain the post-tensioning forces from the support reactions.
- Determination of the bridge's displaced shape caused by a unitary shortening of every cable and a unitary pre-camber value for every deck and mast segment. Formulation of an equation system (to be solved in an external program) representing as many compatibility conditions as unitary shortenings and pre-camber values, allowing one to find the actually needed cable shortenings and pre-camber values

(this is the method adopted in our program BRIDE but implemented in a totally automatic way).

These computations, when performed either manually or using additional external programs, are very time-consuming and inhibit the investigation of several project variants.

1.3. Technical progress of computers

The performance of today's computer technology is impressive: not only the hardware has become faster, also the available programming techniques have progressed considerably, allowing one to describe complex systems in a much more compact and expressive way. Nowadays solution algorithms can be developed which, relying on the computational power available on everybody's desk, model the physical phenomena in a more natural and accurate way than algorithms developed when computational power was limited.

This opens up new research paths which were technically not conceivable before, allowing the researcher to be innovative even in an seemingly well-explored field such as structural analysis.

1.4. The program BRIDE and its special features

The difficulties encountered in the analysis of cable-stayed bridges with standard software and the opportunity offered by the technical progress in computer hard- and software appeared to be two good reasons to start the development of a computer program specially tailored to the requirements of cable-stayed bridge designers.

The fact that a growing number of programs for the analysis of cable- stayed bridges are now available shows that the time is ripe for fundamental research on this topic: this can provide a knowledge base for further research in technically newly accessible fields, assess commercially available programs or as a seed development for new commercial products.

The program has been called "BRIDE", an acronym for BRIdge DEsigner (it is a tradition of our institute to give programs romantic names), and its goal is to help the designer in the following tasks:

- model all kinds of cable-stayed bridges, taking all main non-linear effects into account in a consistent way,
- easily allow the analysis of each construction stage,
- find all needed post-tensioning forces for the cables and all pre-camber values for the mast and deck segments in a fully automatic way and
- correct deviations in the geometry measured in situ by changing the cable forces.

Five main ideas determined the design of the program BRIDE and distinguish it from others programs:

- As the displacements are assumed to be small, which is acceptable since cable-stayed bridges are very stiff, the behaviour of every segment of a structure modelled with a finite element can be fully described by the element's local stiffness matrix plus a set of "initial" displacements. Such an element does not depend on the global structure and, fully representing the physics of the modelled segment of the structure, can be added to or removed from the numerical model just as the corresponding segment can be added or removed in real life.
- Post-tensioning forces or pre-camber values which have to be found automatically by the program can be formulated as initial element displacements whose magnitudes are found through conditions to be fulfilled.
- The erection of a cable-stayed bridge is an evolutionary process and the simplest way to describe it with accuracy is with a list of the changes occurring in the system in chronological order.
- In order to take displacements locked in concrete and creep into account it is necessary to know the displacements of the structure in the stages prior to the stage being analyzed.
- The program BRIDE should be sufficiently user-friendly to be applicable both in the bridge construction industry and for teaching purposes.

1.5. User friendliness: BRIDE vs. commercial software

With programs developed in a research context in computational mechanics it is usual to concentrate more on the implementation of the computational core than of the user interface. In this project, however, the goal was set to achieve an acceptable user-friendliness. For this reason much effort has been invested in the development of a graphic user interface with a three-dimensional representation of the numerical models.

During the design of the program BRIDE a rigorous transparency has been actively attempted in the way the program generates the numeric models from the user specifications and in the way computations are performed. This transparency is a kind of user-friendliness which many commercial programs do not offer and which, apart from bridge designers, can be of pedagogical advantage for both teachers and students.

1.6. Structure of this dissertation

Chapter 2 gives an overview of the main features of the program BRIDE. Chapter 3 explains the composite beam element. Chapters 4, 5 and 6 deal in detail with the contributions to the initial displacements $\mathbf{a}_{initial}$ discussed in the section 2.1 due to locked-in

displacements (chapter 4), creep (chapter 5) and shrinkage (chapter 6). Chapter 7 deals with the computation of the intensities of conditional loads (see subsection 2.2). Chapter 8 explains the way cable stays are automatically dimensioned. In chapter 9 a practical example is given. Chapter 10 presents the conclusions. The appendices contain the user manual of the program BRIDE: in appendix A the input syntax is defined, in appendix B the graphic user interface is explained, appendix C presents the information technology tools used to implement the program BRIDE and in appendix D the complete input files for the model shown in chapter 9 are listed and commented on.

2. A tour of the program BRIDE

The approach adopted in the development of the algorithms for the program BRIDE was to keep as close as possible to the physics of cable-stayed bridge construction, so as to avoid inconsistencies. This lead to an "object-oriented" solution in which the model consists of a list of data objects representing the different components of the real bridge ordered with the same chronological order with which their physical counterparts are added to the real bridge. All needed computations are performed by the program BRIDE on this chronologically consistent numerical model. This chapter gives an overview of them.

2.1. Element description through its local stiffness matrix and a set of initial displacements

Cable-stayed bridges are modelled in the program BRIDE as space frames where 6 displacement parameters (3 displacements and 3 rotations) are introduced in each node.

The geometry of the model is described through the Cartesian coordinates of its nodes and through the element's incidence nodes. In addition, every element requires section and material properties. Once the coordinates of the incidence nodes, the material and the section of the element are defined, its elastic response can be formulated by means of a local 12x12 elastic stiffness matrix.

If we were to consider an existing bridge already modelled in a simulation program (see figure 2.1), take a concrete cutter, cut out a segment of the structure modelled with

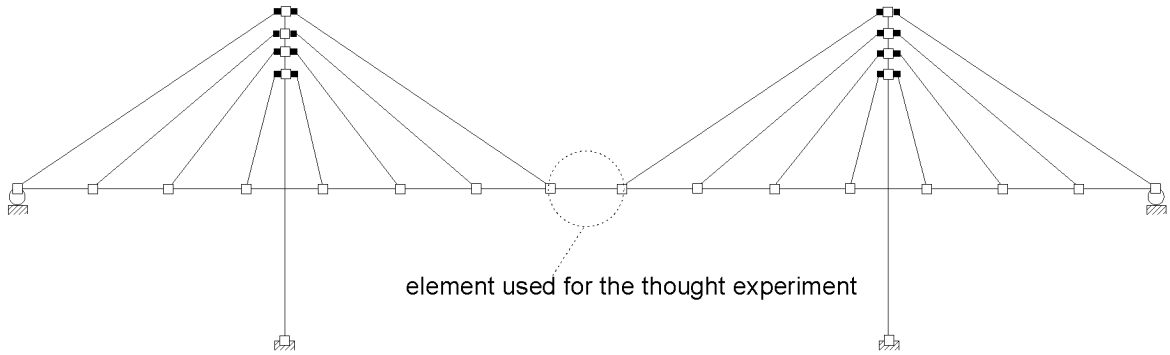


Figure 2.1.: Step 1. Choose one element in the numerical model of an existing bridge.

a single finite element (see figure 2.2) and determine its geometry in the unloaded state,

we would notice that its geometry does not correspond to the geometry of the finite element. In fact, displacements locked-in during the hardening of the concrete, creep, shrinkage and the pre-camber would have slightly changed the shape of the cut out and thus unloaded segment (see figure 2.3).

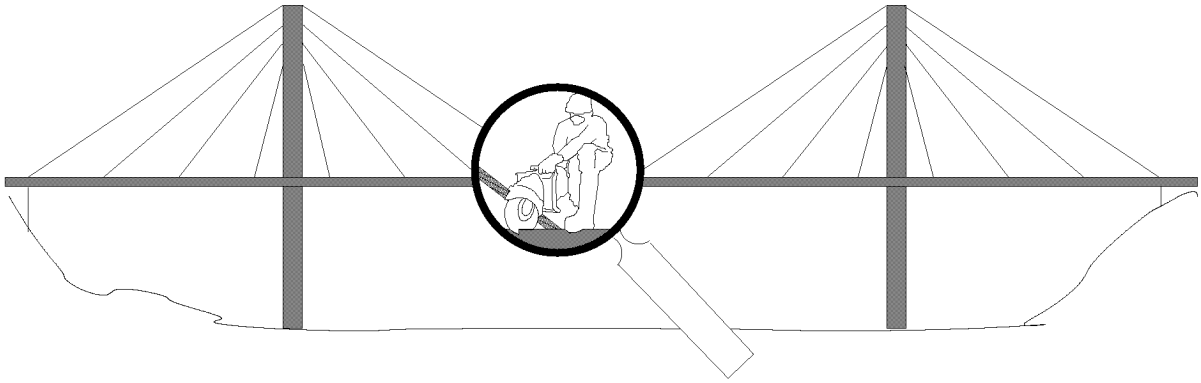


Figure 2.2.: Step 2. Cut out the structural segment corresponding to the chosen element.

The vector $\mathbf{a}_{initial}$ of size 12 (3 translation and 3 rotation components for each incidence node) containing the difference between the cut out segment's and the finite element's geometry can be formulated as follows:

$$\mathbf{a}_{initial} = \mathbf{a}_h + \mathbf{a}_{cr} + \mathbf{a}_{sh} \tag{2.1}$$

where $\mathbf{a}_h$ is the contribution of the locked-in concrete displacements, $\mathbf{a}_{cr}$ that of creep and $\mathbf{a}_{sh}$ is that of shrinkage (the pre-camber is not taken into account here because it is considered as a "conditional load", see section 2.2).

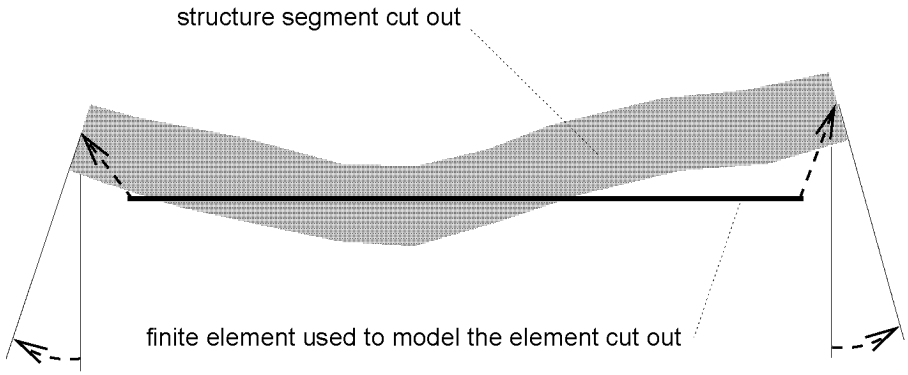


Figure 2.3.: Step 3. The geometry of the cut out structural segment and the geometry of the finite element used to model it differ by the initial displacements $\mathbf{a}_{initial}$.

One way to model the cut out segment correctly is to reproduce its displacements by subjecting the finite element to a set of initial self-equilibrating forces $\mathbf{f}$ (see figure 2.4)

producing the initial displacements $\mathbf{a}_{initial}$. The forces $\mathbf{f}$ are found by multiplying the element local stiffness matrix $\mathbf{k}$ by $\mathbf{a}_{initial}$:

$$\mathbf{f} = \mathbf{k} \cdot \mathbf{a}_{initial} \tag{2.2}$$

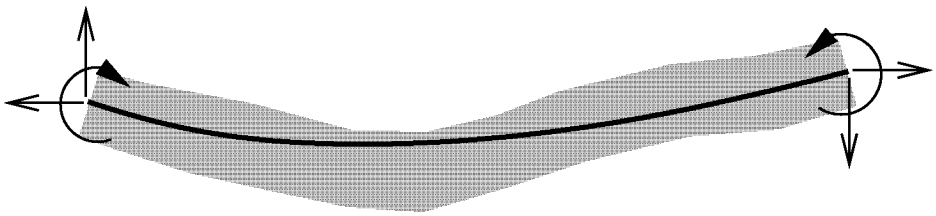


Figure 2.4.: Step 4. The finite element subjected to the initial self-equilibrating force $\mathbf{f}$ assumes the same geometry as the cut out structural segment.

The components of the vector $\mathbf{f}$, as for any kind of element load, have to be assembled in the global load vector every time the element is part of the model, its local stiffness matrix $\mathbf{k}$ being assembled in the global stiffness matrix. Being initial, i.e. corresponding to a deformed but stress-free state, the contribution from $\mathbf{f}$ to the section forces has to be ignored during post-processing, i.e. the section forces induced by $\mathbf{f}$ in the element have to be subtracted from the section forces found from the nodal displacements of the global solution. The subtraction does not affect the equilibrium of the structure because $\mathbf{f}$ is self-equilibrating: this is a property of local stiffness matrices which, being singular, always deliver a self-equilibrating forces group if multiplied by any nodal displacement vector.

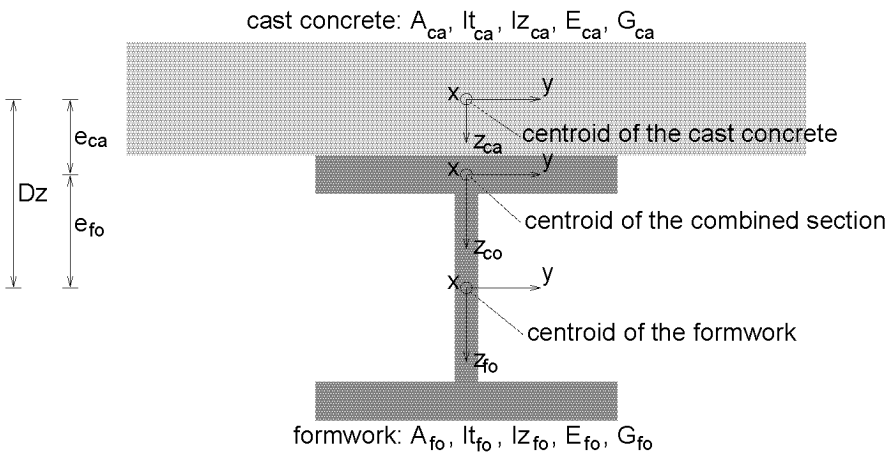


Figure 2.5.: Section idealization used for cast and composite beam elements.

Initial displacements have to be taken into account in a slightly different way for the composite beam element specially developed for the program BRIDE, which allows one

to take the bond between concrete and its underlying steel profile into account (see figure 2.5). The composite beam element, as explained in chapter 3, takes bonding into account by merging into a single combined section the section of the cast concrete and the section of the steel profile, which is modelled by a so-called formwork beam (see section 4.2). For composite beam elements $\mathbf{f}$ is found as follows:

$$\mathbf{f} = \mathbf{k}_{co} \cdot \mathbf{a}_h^{co} - \mathbf{q}_{co}^{fo} + \mathbf{k}_{ca} \cdot (\mathbf{a}_{cr}^{ca} + \mathbf{a}_{sh}^{ca}) + \mathbf{k}_{fo} \cdot (\mathbf{a}_{cr}^{fo} + \mathbf{a}_{sh}^{fo}) \tag{2.3}$$

where $\mathbf{k}_{co}$ is the stiffness matrix computed with the combined section, $\mathbf{a}_h^{co}$ and $\mathbf{q}_{co}^{fo}$ are the displacements and the nodal forces of the formwork beam during the casting stage, $\mathbf{k}_{ca}$ and $\mathbf{k}_{fo}$ are the stiffness matrices of the cast concrete part and of the formwork beam, respectively, with no bond between them. $\mathbf{a}_{cr}^{ca}$, $\mathbf{a}_{sh}^{ca}$, $\mathbf{a}_{cr}^{fo}$ and $\mathbf{a}_{sh}^{fo}$ are the initial displacements of the formwork beam and of the cast concrete beam (it is justified to have in addition initial displacements due to creep and shrinkage for the formwork since this might be the case in pre-cast concrete). All these forces, stiffness matrices and displacements are expressed relative to the centroid of the combined section. The element is then excentrically connected to its nodes.

The fact of not taking bonding into account for $\mathbf{k}_{ca}$ and $\mathbf{k}_{fo}$ is a obvious simplification which is acceptable because the self-equilibrating load $\mathbf{f}$ is applied to the undisplaced composite element (whose stiffness takes bonding into account) hence differences between the initial displacements of the cast concrete $\mathbf{a}_{cr}^{ca} + \mathbf{a}_{sh}^{ca}$ and those of the formwork $\mathbf{a}_{cr}^{fo} + \mathbf{a}_{sh}^{fo}$ only lead to differences in the section forces but not to bond-producing discontinuities in the total strain (the elastic and the initial strain summed together). The elastic strain must compensate any initial strain because in the finite element model the element is forced into its undisplaced shape for the formulation of global equilibrium conditions.

2.2. Conditional loads

The so-called conditional loads represent the post-tensioning of the cable-stays and the pre-camber of mast and deck segments. Their intensities are automatically computed by the program BRIDE.

Both conditional loads and regular loads are model objects to be inserted in the chronological objects list (see section 2.3). The only differences is that in regular loads the load intensity is explicitly defined while in conditional loads it is defined through a condition to be fulfilled, and that conditional loads always belong to the standard load case *load_history* (see section 2.4). Their required intensities are found automatically by the program BRIDE by fulfilling appropriate conditions specified by the program user (e.g. "post-tension a cable with a force such that its anchoring point on the deck exhibits no vertical displacement in the final construction stage, taking into account the dead load and all other conditional loads", see figure 2.6).

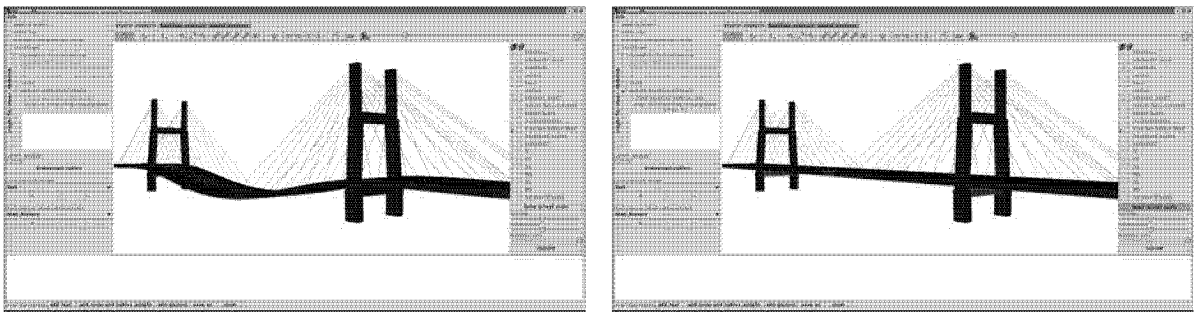


Figure 2.6.: Deflected shapes of the same model without and with the automatically computed pre-camber of the deck and mast segments and the post-tensioning of the cables.

2.3. Chronological objects list: a full description of the erection process

After some attempts it became clear that the most convenient way to specify the evolving model of a cable-stayed bridge during construction is a list of all the model's objects (nodes, elements, supports, forces, ...) ordered in the same chronological order in which their real counterparts are added to the bridge being constructed. The object "stage" has been introduced to allow the program user to define a new construction stage whenever he feels that enough model objects have been added to the model since the last stage object. The stage objects define the states of the bridge where analysis is possible.

Once the program BRIDE has read the objects list defined by the user, it constructs a list of data objects representing it.

To build the numerical model of any construction stage, the (object-oriented) program BRIDE goes through the list of data objects starting from the beginning, sends to each data object the signal to assemble itself [1] to the actual model and stops when it encounters the data object corresponding to the desired construction stage (see figure 2.7).

To analyze the model for a given load case the program BRIDE proceeds in an analogous way as for the assembling of the finite element model: it goes through the chronological objects list from the beginning, takes into account the load objects belonging to the chosen load case and stops when the stage object corresponding to the stage being analyzed is encountered (see figure 2.8).

[1] The fact of viewing units of data as active "objects" rather than passive units distinguishes the *object oriented* programming paradigm supported by programming languages like Java and C++ from the traditional imperative paradigm of programming languages such as Fortran, Pascal and C, in which the units of data are passive and manipulated through procedures not included in their definition.

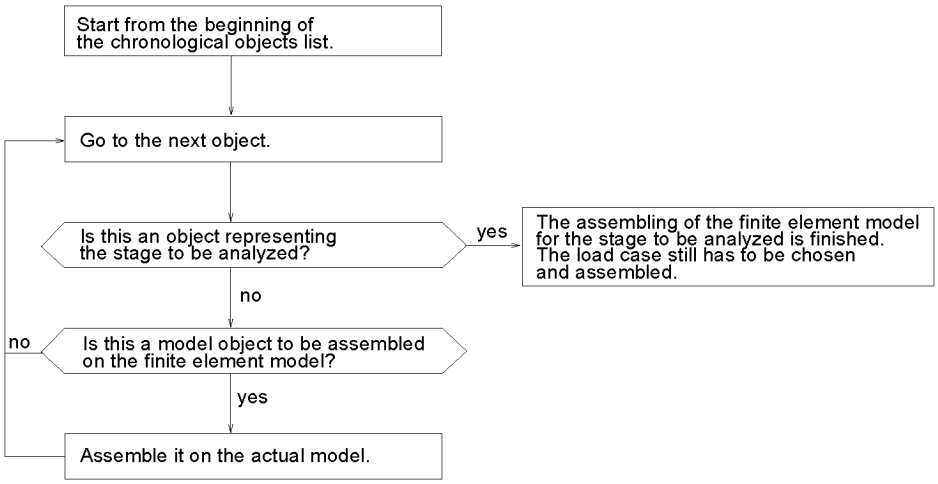


Figure 2.7.: Flow diagram of the assembling of a construction stage.

2.4. The stage-by-stage iteration: an automatic analysis of each construction stage in chronological order

To take creep and locked-in concrete displacements in a given construction stage into account it is necessary to know the initial displacements $\mathbf{a}_{cr}$ and $\mathbf{a}_h$ for every element (see section 2.1), which, in turn, depend on the displaced shapes of the bridge in all previous construction stages. Since it is important to allow the user to view the results of any construction stage immediately, without having to wait for the analysis of all former stages, the nodal displacements of every construction stage due to all relevant loads are stored in the corresponding stage data object.

Such nodal displacements are found during what is called *stage-by-stage iteration*, in which every construction stage is automatically assembled and analyzed in chronological order. Figure 2.9 shows the flow diagram of such an iteration.

To denote the load objects which should be taken into account during the stage-by-stage iteration the standard load case called *load_history* has been introduced. The conditional loads are automatically assigned to this standard load case because they are to be computed during the stage-by-stage iteration. The user, however, is still free to assign them to other load cases or load combinations.

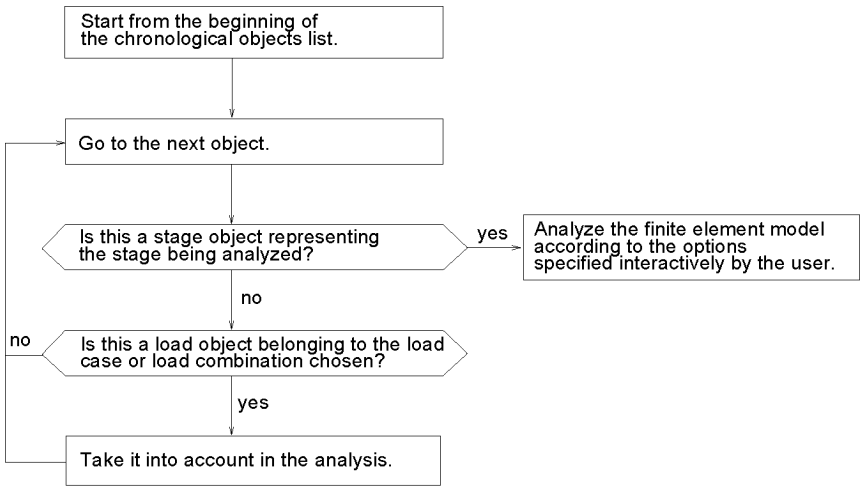


Figure 2.8.: Flow diagram of the selection of the load objects which have to be taken into account during the analysis.

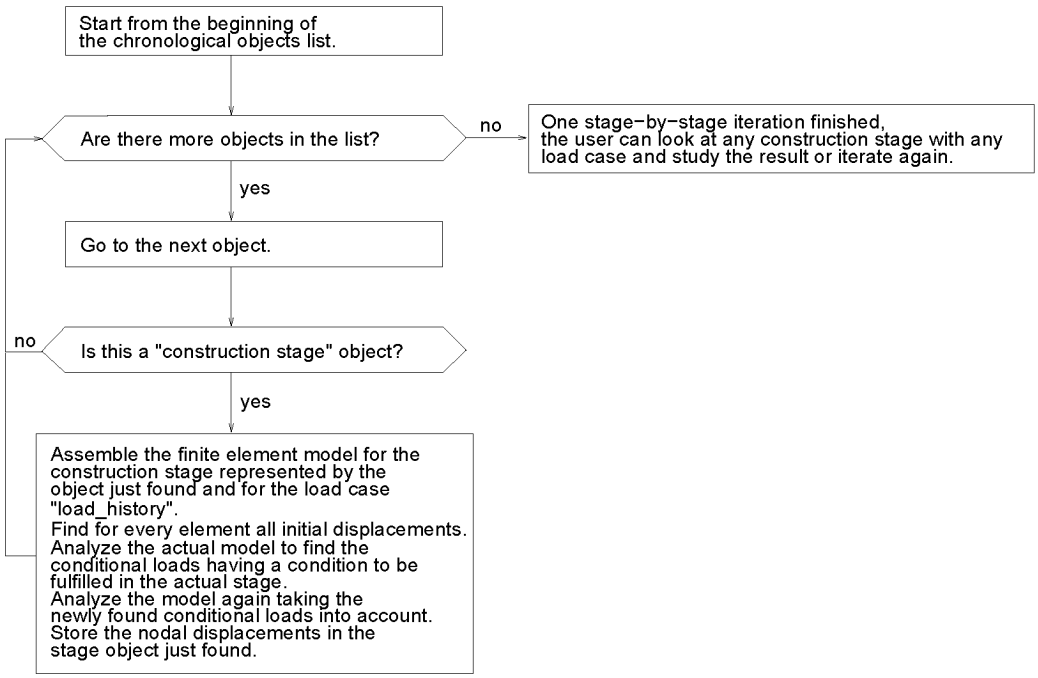


Figure 2.9.: Flow diagram of one stage-by-stage iteration.

3. Steel-concrete composite beam elements

Composite beam elements have been specially developed for the program BRIDE to model steel-concrete composite decks with a shear connection assumed to be rigid taking locked-in displacement into account. This is generally more important for composite decks than for plain cast decks (see chapter 4), because the stiffness of the steel profiles is generally lower than the stiffness of the temporary formworks used for the plain cast segments.

The main difference between a cast deck and a composite deck is that here the formwork, i.g. the steel profile, is never removed from the cast section but remains rigidly bonded to it. A combined model for composite decks has been adopted, which is based on the assumption that for the force range usually allowed for bridges the behaviour of composite deck segments remains elastic.

When, during stage-by-stage iteration, a stage is selected in which a new composite beam is defined, its *formwork beam* (the beam object used to simulate the steel profile, see chapter 4) is removed from the model and the properties of the *combined section* resulting from the merging of the cast concrete section (defined in the composite beam object) and the formwork beam section are automatically found by the program BRIDE and assigned to the composite beam element.

The section properties for the combined section are calculated as follows:

$$(AE)_{co} = A_{fo}E_{fo} + A_{ca}E_{ca} \tag{3.1}$$

$$e_{fo} = \frac{A_{ca}E_{ca} \cdot Dz}{(AE)_{co}} \tag{3.2}$$

$$e_{ca} = \frac{A_{fo}E_{fo} \cdot Dz}{(AE)_{co}} = Dz - e_{fo} \tag{3.3}$$

$$(I_tG)_{co} = I_{t,fo}G_{fo} + I_{t,ca}G_{ca} \tag{3.4}$$

$$(I_yE)_{co} = (I_{y,fo} + A_{fo}e_{fo}^2)E_{fo} + (I_{y,ca} + A_{ca}e_{ca}^2)E_{ca} \tag{3.5}$$

$$(I_zE)_{co} = I_{z,fo}E_{fo} + I_{z,ca}E_{ca} \tag{3.6}$$

where the subscript $_{co}$ denotes the combined section, $_{fo}$ denotes the formwork beam section, $_{ca}$ denotes the cast concrete section, A is the section's area, e is the excentricity of the centroid of either cast concrete and formwork section from the centroid of the combined section, E the Young-modulus, G the shear-modulus, I_t the the torsional

rigidity, I_y and I_z are the moments of inertia with respect to the local y and z axes (see figure 2.5).

The normal stress distribution along the local z axis in the combined beam is found by superposing four components (see figure 3.1):

- the normal stresses during the casting stage, where the composite beam section is unstressed because the concrete is still wet and the formwork beam has its elastic normal stress
- the elastic normal stresses of the combined beam from the actual global solution and from the actual local load
- the stresses obtained by subtracting the elastic normal stresses the combined beam would have if it was subjected to the hardening displacements $\mathbf{a}_h^{co}$ (see chapter 4) and to the local load acting on the formwork beam during the casting stage
- the stresses due to creep and shrinkage.

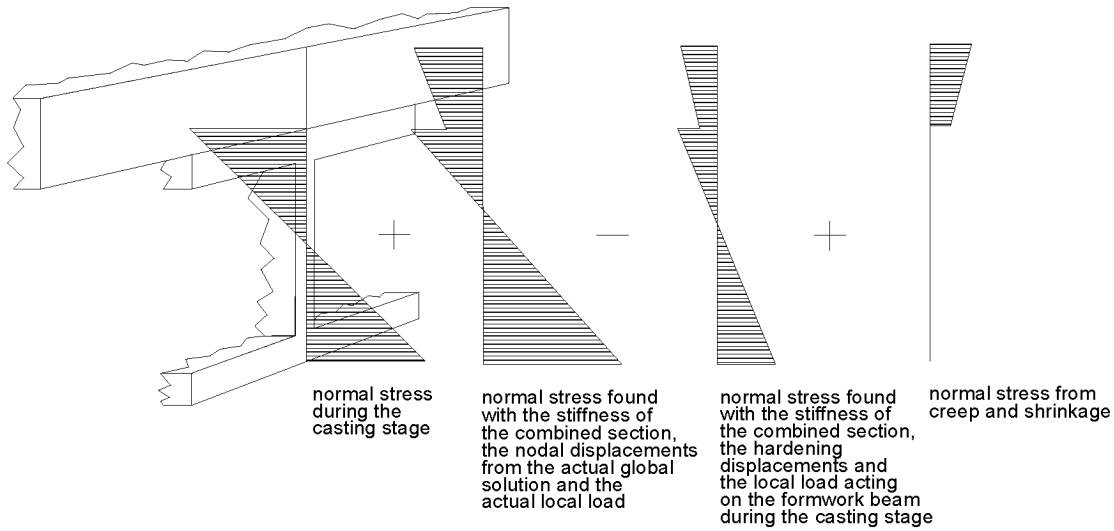


Figure 3.1.: Superposition of the different normal stress components in a concrete-steel composite beam.

3.1. The excentric local stiffness matrices k_{ca} and k_{fo}

As mentioned in section 2.1 to compute the self-equilibrating force $\mathbf{f}$ needed to take the initial displacements into account for composite beam elements it is necessary to have the local stiffness matrix $\mathbf{k}_{ca}$ for a hypothetical beam having the cast concrete as section and the stiffness matrix of the formwork element $\mathbf{k}_{fo}$, both as if there was no bond between formwork and cast concrete and expressed relative to the combined section's centroid, i.e. excentrically (see eq. 2.3).

$\mathbf{k}_{ca}$ can be derived as follows: assuming a rigid connection between the formwork beam and the cast concrete, the displacements $\mathbf{a}^{ca}$ in the centroid of the cast concrete section can be expressed as a function of the displacement parameters $\mathbf{a}$ of the combined beam:

$$\mathbf{a}^{ca} = \mathbf{C}_{ca} \cdot \mathbf{a} \tag{3.7}$$

or the nodal forces from the centroid of the cast concrete section $\mathbf{q}^{ca}$ can be transformed to $\mathbf{q}$, the same forces set but applied to the combined beam:

$$\mathbf{q} = \mathbf{C}_{ca}^{\mathbf{T}} \cdot \mathbf{q}^{ca} \tag{3.8}$$

With 3 displacement parameters in global X-, Y-, and Z-directions and 3 rotations with respect to the corresponding coordinate axis introduced (in that order) at each node, the 12x12 transformation matrix $\mathbf{C}_{ca}$ is defined as follows:

$$\mathbf{C}_{ca} = \begin{matrix} \mathbf{T} & 0 \\ 0 & \mathbf{T} \end{matrix}$$

$\mathbf{T}$ is the following 6x6 transformation matrix:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 & dz & -dy \\ 0 & 1 & 0 & -dz & 0 & dx \\ 0 & 0 & 1 & dy & -dx & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where dx, dy and dz are the components of the excentricity vector $[0, 0, e_{ca}]$ expressed in the global coordinates system.

Since the local stiffness matrix $\mathbf{k}_{ca,centric}$ of a beam having the cast concrete section is known and since:

$$\mathbf{q}^{ca} = \mathbf{k}_{ca,centric} \cdot \mathbf{a}^{ca} \tag{3.9}$$

by inserting eq. 3.7 in eq. 3.9 and eq. 3.9 in eq. 3.8 the following relation applies:

$$\mathbf{q} = \mathbf{C}_{ca}^{\mathbf{T}} \cdot \mathbf{k}_{ca,centric} \cdot \mathbf{C}_{ca} \cdot \mathbf{a} \tag{3.10}$$

The resulting stiffness matrix is the excentric stiffness matrix for the cast concrete section $\mathbf{k}_{ca}$:

$$\mathbf{k}_{ca} = \mathbf{C}_{ca}^{\mathbf{T}} \cdot \mathbf{k}_{ca,centric} \cdot \mathbf{C}_{ca} \tag{3.11}$$

$\mathbf{k}_{fo}$ can be derived in a completely analogous way.

4. Locked-in displacements

Any deformation of the formwork remains "stored" in the cast concrete as it hardens. In contrast to the usual case, where the formwork is stiff enough to avoid excessive deflections, this has to be taken into account in the case of cable-stayed bridges (especially those with a composite steel-concrete deck) because the repetitive, incremental construction by free-cantilevering can lead to a systematic accumulation of error causing an important curvature to be locked-in in the deck.

To simulate locked-in displacements two different types of elements have been implemented: a *cast beam* for concrete having no shear connection with its formwork and a *composite beam* for concrete having a rigid shear connection with a steel profile which plays the role of non-removable formwork (see chapter 3). In the program BRIDE the formwork is modelled by a regular beam element.

4.1. Cast beams to simulate locked-in displacements in concrete cast on an elastic formwork without shear connection

In a cast beam element, instead of defining the element's incidence nodes the user defines the beam element representing the formwork and a value Dz (see figure 2.5) representing the distance between the cast beam and the formwork beam centroids. The cast beam element retrieves its incidence nodes and orientation from the formwork element and the initial hardening displacements the formwork beam had during the *casting stage* - the stage coming before the one in which the new cast beam is added to the chronological objects list - accordingly transformed, contribute to each cast beam's initial displacements $\mathbf{a}_{initial}$.

This is how cast beams are treated numerically:

- The user defines a beam element modeling the formwork (the *formwork beam*) and a distributed element force object simulating the weight of the wet concrete.
- The user defines the *cast beam* object for the construction stage right after the casting stage.
- While performing the stage-by-stage iteration the program BRIDE checks if in the stage following the currently selected one new cast beams are introduced in the model. If this is the case it transforms the displacement parameters of the corresponding formwork beams to the displacements of the (not yet assembled)

centroids of the cast beams assuming a rigid connection (analogous to eq. 3.7) and stores them within the cast beam objects. These transformed displacement parameters are the *locked-in displacements* $\mathbf{a}_h$. They represent a contribution to the cast beam's $\mathbf{a}_{initial}$ (see eq. 2.1).

4.2. Composite beams to simulate locked-in displacements in concrete cast on an elastic formwork with shear connection

The locked-in displacements for composite beams can be found in the same way as for cast beams up to a point: the nodal forces of the formwork, which is removed from the model after the casting stage (since its stiffness is already included in the combined section's stiffness), have to be subtracted from the initial self-equilibrating forces $\mathbf{f}$ after having been transformed for the centroid of the combined section (see eq. 2.3):

- The hardening displacements $\mathbf{a}_h$ which the formwork beam had during the casting stage have to be transformed analogously as in eq. 3.7 for the centroid of the combined section. The transformed hardening displacements are denoted by $\mathbf{a}_h^{co}$.
- The nodal forces $\mathbf{q}_{fo}$ applied by the formwork beam to its incidence nodes during the casting stage as a result of its elastic deformation are found as follows:

$$\mathbf{q}_{fo} = \mathbf{k}_{fo} \cdot \mathbf{a}_h \tag{4.1}$$

  where $\mathbf{k}_{fo}$ is the formwork's (centric) stiffness matrix.
- analogous to eq. 3.8, $\mathbf{q}_{fo}$ is transformed for the centroid of the combined section and denoted by $\mathbf{q}_{fo}^{co}$.
- the initial self-equilibrating force $\mathbf{f}_{h,co}$ needed to take the locked-in displacements in the composite beam, which is the first term of eq. 2.3, is found as follows:

$$\mathbf{f}_{h,co} = \mathbf{k}_{co} \cdot \mathbf{a}_h^{co} - \mathbf{q}_{fo}^{co} \tag{4.2}$$

5. Creep

Concrete creep often has to be taken into account in the analysis of wide-span cable-stayed bridges. According to [CEB/FIP, p. 46] in the range of service stresses, i.e. $\sigma \leq 0.4 f_{cm}$[1], concrete may be considered as an ageing linear viscoelastic material. In the program BRIDE only the linear viscoelastic behaviour has been taken into account, while ageing (i.e. changing of material characteristics over the time) has been ignored[2].

The creep behaviour of materials is usually described by the creep factor $\phi(t)$ which is the ratio between the elastic component ε_{el} and the creep component ε_{cr} of strain in an uniaxial stress-strain state. In the program BRIDE $\phi(t)$ is approximated by the following hyperbolic function:

$$\phi(t) = \phi_\infty \cdot tanh(t \cdot \frac{arctanh(0.5)}{t_2}) \tag{5.1}$$

where ϕ_∞ is the maximum asymptotic value of ϕ and t_2 is the number of days in which half the maximum asymptotic value ϕ_∞ is reached. Both ϕ_∞ and t_2 have to be specified as input parameters in the material object (see subsection A.1.7). The graph for $\phi(t)$ is shown in figure 5.1.

The values for ϕ_∞ and t_2 should be obtained from the corresponding design code in which the age of concrete when it is loaded can also be taken into account. Typically ϕ_∞ has a value around 2 and t_2 a value around 40 days. The time t (see figure 5.1) is measured from the day the concrete first comes under stress.

5.1. Hereditary creep behaviour of concrete

According to [Wittmann, IV.9] and [Shaw and Whiteman, p.185] hereditary creep behaviour for viscoelastic materials such as concrete can be taken into account by assuming that a Boltzmann superposition of stress can be applied, where these stress increments are related by Hooke's law to corresponding strain increments. If a uniaxial stress-strain state is considered, the stress tensor σ_{ij} and the strain tensor ε_{ij} are reduced to a scalar and Hooke's law can be expressed as follows:

$$\sigma(t) = D(t) \cdot \varepsilon(t_0) \tag{5.2}$$

[1] f_{cm} is the mean value of the cylinder compressive strength of concrete f_c at an age of 28 days, σ is the normal tension.

[2] This is a constraint due to the adoption of Boltzmann's superposition principle explained in section 5.1.

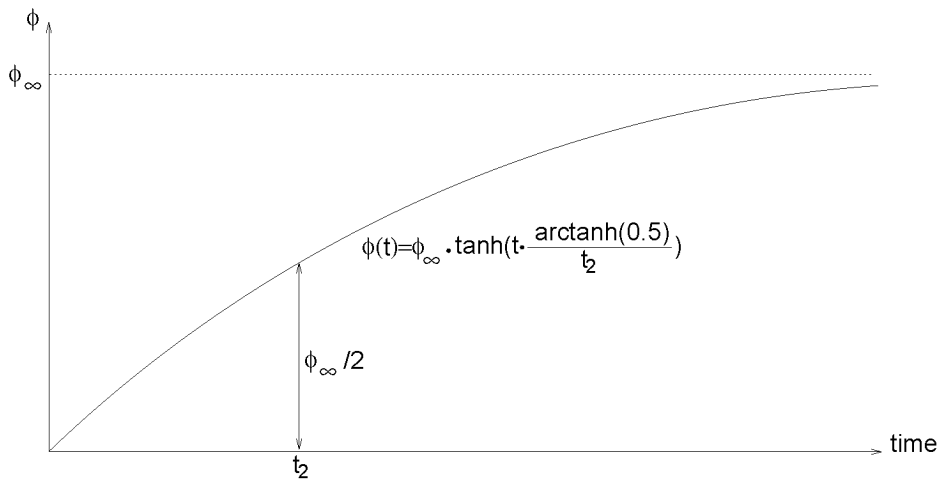


Figure 5.1.: Function $\phi(t)$ used to approximate the creep factor.

where a time dependence has been introduced in $D(t)$ which is a smooth monotonically decreasing function of t. The loss of tension over time due to the decreasing of $D(t)$ with a constant $\varepsilon(t) = \varepsilon(t_0)$ is known as *stress relaxation*.

Let us denote by an increasing index i all the construction stages since the assembling of a new element, starting with $i = 0$ for the stage in which the element has been assembled. If we assume that the nodal displacements and thus the strain ε remain constant between one stage and the next, the strain $\varepsilon(t)$ in the element can be approximated by the step function $\widetilde{\varepsilon}(t)$:

$$\widetilde{\varepsilon}(t) = \varepsilon(t_i) \qquad in\,[t_i, t_{i+1}[\,for\, i = 0, 1, 2, ..., \tag{5.3}$$

Each strain increment

$$\Delta\varepsilon(t_{i+1}) = \varepsilon(t_{i+1}) - \varepsilon(t_i) \tag{5.4}$$

induces a stress increment according to Hooke's law:

$$\Delta\sigma(t_i, t_j) = D(t_i - t_j) \cdot \Delta\varepsilon(t_j) \qquad for\, 0 \leq j \leq i \tag{5.5}$$

where $\Delta\sigma(t_i, t_0)$ and $\Delta\varepsilon(t_0)$ represent the stress and the strain increments at time t_0 when the element passes from not being assembled (a stress and strain free state) to being assembled. Notice that each of these stress increments also relaxes according to the time dependence of $D(t)$. The total stress at time t_i is now given by superposition:

$$\sigma(t_i) = \sum_{j=0}^{i} \Delta\sigma_j(t_i) = \sum_{j=0}^{i} D(t_i - t_j)\Delta\varepsilon(t_j) \tag{5.6}$$

In a material subject to the constant strain $\varepsilon(t_0)$ the sum of the elastic and the creep components of strain $\varepsilon_{el}(t)$ and $\varepsilon_{cr}(t)$ is constant and must equal $\varepsilon(t_0)$:

$$\varepsilon(t_0) = \varepsilon_{el}(t) + \varepsilon_{cr}(t) = constant \tag{5.7}$$

since $\phi(t)$ is the ratio between $\varepsilon_{el}(t)$ and $\varepsilon_{cr}(t)$, $\varepsilon_{cr}(t)$ can be eliminated from eq. 5.7:

$$\varepsilon(t_0) = \varepsilon_{el}(t) \cdot (1 + \phi(t)) = constant \tag{5.8}$$

the elastic strain $\varepsilon_{el}(t)$ can be formulated as a function of the elastic stress $\sigma(t)$ and the Young's modulus E:

$$\varepsilon_{el}(t) = \frac{\sigma(t)}{E} \tag{5.9}$$

hence for t_0, when the relaxation has not yet taken place, the elastic stress $\sigma(t_0)$ can be formulated as follows:

$$\sigma(t_0) = \varepsilon(t_0) \cdot E \tag{5.10}$$

and as follows for $t > 0$:

$$\sigma(t) = \frac{E \cdot \varepsilon(t_0)}{1 + \phi(t)} \tag{5.11}$$

the ratio between $\sigma(t)$ and $\sigma(t_0)$ is the relaxation factor $\psi(t)$ (see figure 5.2):

$$\psi(t) = \frac{1}{1 + \phi(t)} \tag{5.12}$$

With help of eq. 5.12 for uniaxial stress-strain states $D(t)$ can be determined as follows:

$$D(t) = E \cdot \psi(t) \tag{5.13}$$

Eq. 5.6 can be written as follows:

$$\sigma(t_i) = \sum_{j=0}^{i} E \cdot \psi(t_i - t_j) \Delta\varepsilon(t_j) \tag{5.14}$$

Ignoring the contribution to creep from the element's local load, the relationship of eq. 5.14 can be generalized for the nodal forces $\mathbf{q}$, since $\Delta\varepsilon(t_j)$ is proportional to the difference between the nodal displacements $\Delta\mathbf{a}(t_j)$ of two subsequent stages numbered $j-1$ and j:

$$\mathbf{q}(t_i) = \sum_{j=0}^{i} \mathbf{k} \cdot \psi(t_i - t_j) \cdot \Delta\mathbf{a}(t_j) \tag{5.15}$$

where $\mathbf{k}$ is the element's local stiffness matrix and the difference $\Delta\mathbf{a}(t_j)$ is found as follows:

$$\Delta\mathbf{a}(t_j) = (\mathbf{a}(t_j) - \mathbf{a}_{sh}(t_j) - \mathbf{a}_{user}(t_j)) - (\mathbf{a}(t_{j-1}) - \mathbf{a}_{sh}(t_{j-1}) - \mathbf{a}_{user}(t_{j-1})) \tag{5.16}$$

where $\mathbf{a}(t)$ is the vector of the nodal displacements from the global solution, $\mathbf{a}_{sh}(t)$ is the vector of the initial displacements due to shrinkage and $\mathbf{a}_{user}(t)$ is the vector of the initial displacement defined by the user (e.g. to model a pre-camber or a cable post-tensioning). $\mathbf{a}(t_{-1})$, $\mathbf{a}_{sh}(t_{-1})$ and $\mathbf{a}_{user}(t_{-1})$ needed to find $\Delta\mathbf{a}(t_0)$ are zero vectors representing the unloaded state of the element before it is assembled in the structure.

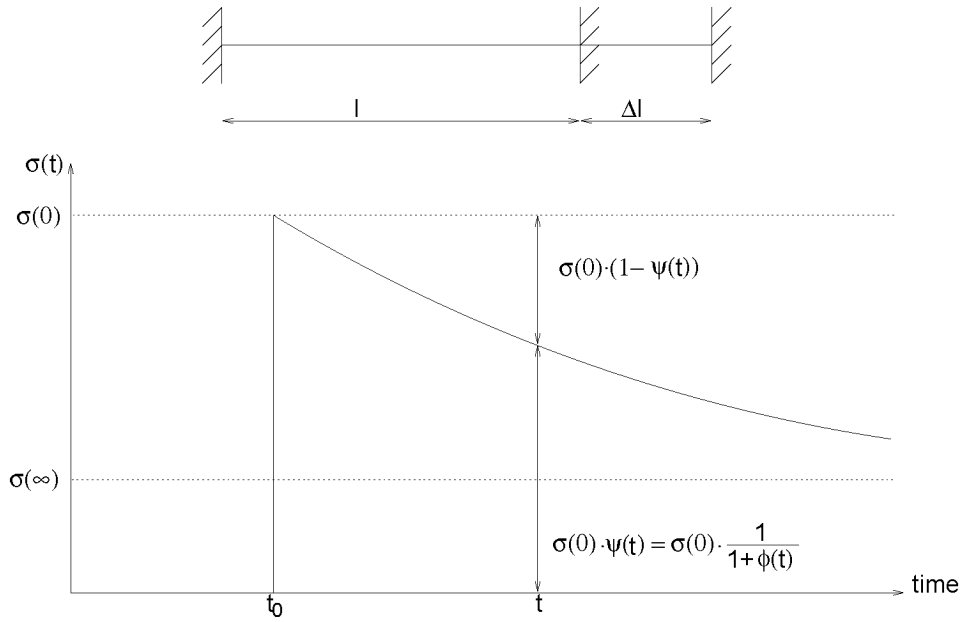


Figure 5.2.: Relaxation in a bar lengthened by Δl at the time t_0.

Every geometry and load change in the structure occurring between the stages $j-1$ and j induces a change $\Delta\mathbf{a}(t_j)$ which is found by solving the global equation system. The increment of the nodal forces $\Delta\mathbf{q}(t_j)$ related to the increment in the nodal displacements $\Delta\mathbf{a}(t_j)$ is computed as follows:

$$\Delta\mathbf{q}(t_j) = \mathbf{k} \cdot \Delta\mathbf{a}(t_j) \tag{5.17}$$

From eq. 5.17 and eq. 5.15 we can show that the loss of magnitude of $\Delta\mathbf{q}(t_j)$ due to relaxation can be expressed as follows:

$$\Delta\mathbf{q}_{cr}(t_j) = \mathbf{k} \cdot \Delta\mathbf{a}(t_j) \cdot (1 - \psi(t_i - t_j))$$

which corresponds to an initial displacement

$$\Delta\mathbf{a}_{cr}(t_i, \Delta\mathbf{q}(t_j)) = (1 - \psi(t_i - t_j)) \cdot \Delta\mathbf{a}(t_j) \tag{5.18}$$

the total initial displacement $\mathbf{a}_{cr}$ (see eq. 2.2 and 2.3) needed to take creep into account is a superposition of all $\Delta\mathbf{a}_{cr}(t_i, \Delta\mathbf{q}(t_j))$:

$$\mathbf{a}_{cr}(t_i) = \sum_{j=0}^{i} \Delta\mathbf{a}_{cr}(t_i, \Delta\mathbf{q}(t_j)) = \sum_{j=0}^{i} \Delta\mathbf{a}(t_j) \cdot (1 - \psi(t_i - t_j)) \tag{5.19}$$

5.2. Approximation in the computation of creep in structures with a varying strain

While creep is considered exactly with eq. 5.19 for systems having a constant strain, for system having a varying strain an error is introduced because the contributions to $\mathbf{a}_{cr}$ from the nodal displacements due to creep between subsequent stages are ignored. However this approximation has proved to be good and to improve rapidly if the time is subdivided in more steps through the definition of more possibly fictitious construction stages.

The increase of the displacements due to creep in a statically determinate model (the extreme case for varying strains) having a single material and in which all load is applied at once at the time t_0 can be summarized with the factor $\phi_B(t_i) = \frac{a(t_i)-a(t_0)}{a(t_0)}$, where a is any component of the displacements vector $\mathbf{a}$.

The factor $\phi_B(t_i)$ implicitly found by the program BRIDE during the stage-by-stage iteration can be formulated as follows:

$$\phi_B(t_i) = \left(\sum_{a=0}^{i} s_{i,a}\right) - 1 \tag{5.20}$$

where:

$$s_{i,0} = 1 \qquad s_{1,1} = 1 - \psi(t_1 - t_0) \tag{5.21}$$

$$s_{i,j=i} = \left(\sum_{a=0}^{i-1} s_{i-1,a} - \sum_{a=0}^{i-2} s_{i-2,a}\right) \cdot (1 - \psi(t_i - t_{i-1}) \tag{5.22}$$

$$s_{i,j<i} = \frac{s_{j,j}}{1 - \psi(t_j - t_{j-1})} \cdot (1 - \psi(t_i - t_{j-1}) \tag{5.23}$$

This complicated formulation, which exactly expresses what happens during the stage-by-stage iteration, can be visualized for four construction stages by means of the following table:

$s_{i,j}$	$j=0$	$j=1$	$j=2$	$j=3$
$i=0$	1			
$i=1$	1	$1-\psi(t_1-t_0)$		
$i=2$	1	$1-\psi(t_2-t_0)$	$(1-\psi(t_1-t_0))(1-\psi(t_2-t_1))$	
$i=3$	1	$1-\psi(t_3-t_0)$	$(1-\psi(t_1-t_0))(1-\psi(t_3-t_1))$	$((1-\psi(t_2-t_0))+(1-\psi(t_1-t_0))(1-\psi(t_2-t_1))-(1-\psi(t_1-t_0))\cdot(1-\psi(t_3-t_2))$

The relative error $r(t_i)$ which occurs between the function $\phi(t)$ adopted to approximate the creep function and the actual $\phi_B(t)$ can be expressed as follows:

$$r(t_i) = \frac{\phi_B(t_i) - \phi(t_i)}{\phi(t_i)} \tag{5.24}$$

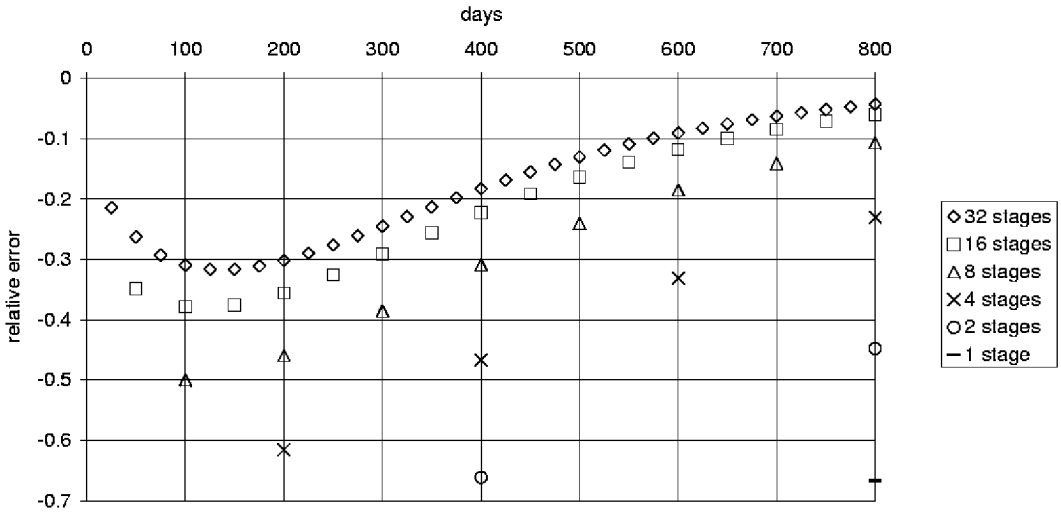


Figure 5.3.: Relative error between $\phi_B(t)$ and $\phi(t_i)$ with different subdivisions of time and for a material having a $\phi_\infty = 2$ and a $t_2 = 100$ days.

and its magnitude can be controlled numerically. Figure 5.3 shows a chart of $r(t_i)$ for different subdivisions of time: with an increasing subdivision, $\phi(t)$ converges towards the exact solution and for subdivision into 32 steps $r(t_i)$ has a magnitude comparable to the approximation in the estimation of ϕ_∞ and t_2 in the design codes.

6. Concrete shrinkage

Concrete shrinkage is difficult to predict but has to be taken into account for wide span cable-stayed bridges. The program BRIDE allows one to introduce empirical values in the model in a simple and effective way exploiting the advantage of knowing when each element has been added to the structure thanks to the chronological object list.

For our purposes this complex phenomenon can be simplified to a time-dependent initial strain approximated with the same function as for creep:

$$\varepsilon_{sh}(t) = \varepsilon_{sh\infty} \cdot tanh(t \cdot \frac{arctanh(0.5)}{t_2})$$

(see figure 6.1) defined in the material object (see A.1.7) by two values: the maximum asymptotic value $\varepsilon_{sh\infty}$ and the number of days t_2 in which it reaches half the maximum asymptotic value. The time t, however, has to be measured from the day the concrete hardens (even if it is not stressed, which is the case when prefabricated panels are used).

The values $\varepsilon_{sh\infty}$ and t_2 should be obtained from the relevant design code. Typically the value for $\varepsilon_{sh\infty}$ is around 0.0005 and the value for t_2 is between 30 days and 2 years.

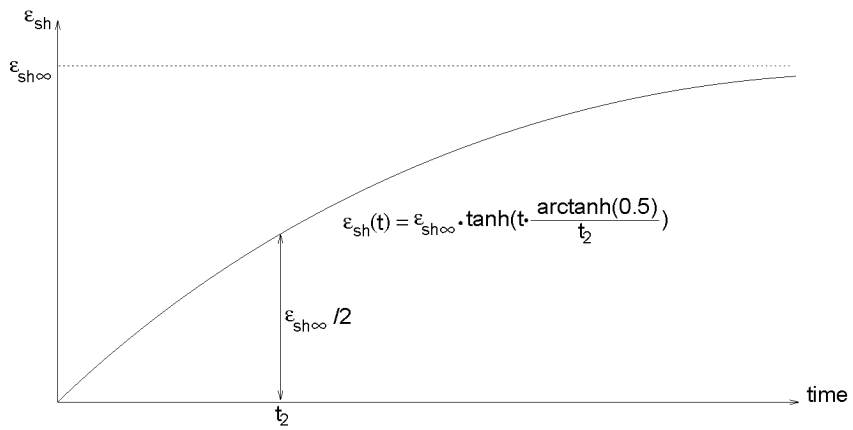


Figure 6.1.: Function $\varepsilon_{sh}(t)$ used to approximate the initial strain due to shrinkage.

6.1. Initial displacements a_{sh} needed to take concrete shrinkage into account

In contrast to locked-in displacements and creep, shrinkage does not need the stage-by-stage iteration to be computed because it depends only on the time t_0 the concrete was cast. In the case of cast and composite beams the time t_0 corresponds to the day number of the casting stage, otherwise it corresponds to the day number of the construction stage in which the material (a model object as any other) has been defined. However, it is necessary to take shrinkage into account during the stage-by-stage iteration because it affects the displaced shape which, in turn, affects locked-in displacements, creep and conditional loads.

The initial displacements $\mathbf{a}_{sh}$ needed to take shrinkage into account (see eq. 2.1 and 2.3) are obtained from the nodal displacements the shrinkage-affected element would have if it was fully clamped in the first node and free to move in the second. For this reason $\mathbf{a}_{sh}$ is a vector in which the only non-zero component is the component

$$a_{sh,tx,2} = \varepsilon_{sh}(t - t_0) \cdot l$$

corresponding to the translation in the longitudinal direction of the second incidence node, where t_0 is the time at which creep begins, t is the analysis time and l is the element length.

7. Conditional loads

In most structures the displacements are so small that they do not cause an excessive deviation from the planned geometry. Quite contrary to this rule, displacements play a major role in cable-stayed bridges and are usually corrected by post-tensioning the cables and by mounting mast and deck segments with a pre-camber. Such adjustments in geometry can be simulated with initial displacements whose magnitude is found automatically by the program BRIDE and which are denoted as *conditional loads*.

Conditional loads are load objects (see subsections A.1.12 and A.1.13) to which a condition to be fulfilled has been appended (see subsection A.1.14). These are a few examples of the statements which can be formulated in the program input with such conditions:

- The initial displacement of the cable specified in the load object to which the condition is appended should have a value such that the vertical displacement of the anchoring point of the cable equals zero in the "last" construction stage, under the load case "load_history" and taking all other conditional loads into account whose intensity are also found in the "last" construction stage.

- The initial displacement of the cable specified in the load object to which the condition is appended should have a value, such that the rotation of the deflected shape of the formwork beam to which the loaded cable is anchored is equal at the formwork beam's beginning and ending during the formwork's casting stage[1], under the load case "load_history" and taking all other conditional loads into account whose intensity are also found in the formwork's casting stage.

- The initial force acting on the deck segment specified in the load object to which the condition is appended and modelling the deviation of the real creep from the assumptions made at input should have a value such that the vertical displacement of the deck at the measurement point equals the measured value in a given construction stage (the stage corresponding to the measurement day), under the load case "load_history" and taking all other conditional loads into account whose intensity are also found in that construction stage.

The imposition of absolute displacements and section forces are not the only conditions which can be formulated. The program BRIDE also allows relative and "same factor as" conditions (see section A.1.14).

[1]This condition can be used to avoid the accumulation of locked-in bending in composite decks, as explained in [Schlaich].

A typical case in which a "same factor as" condition should be used is the one in which two symmetric cables, both anchored to the ground on the same anchor block but one on the right and the other on the left of the carriageway, and both reaching the top of the same mast, have to be post-tensioned to avoid a longitudinal displacement of the mast's top. This should be formulated using an absolute condition for one cable (i.e. "post-tension the first cable in such a way that the top of the mast does not move longitudinally") and a same factor condition for the second cable (i.e. "post-tension the second cable so as to have the same conditional factor as for the first cable").

For the definition of a conditional load the following is needed:

- A load object whose correct intensity is not yet known and thus still arbitrary: the *arbitrary intensity load object.* During the stage-by-stage iteration the correct intensity is automatically found by the program BRIDE in the form of a *conditional factor* with which the load represented by the arbitrary intensity load object is multiplied. Actually after such multiplication the arbitrary intensity load object does not deserve the adjective "arbitrary " anymore, because its intensity has been computed; however we keep calling them an arbitrary intensity load object because this computed intensity is used to perform the influence matrix algorithm (see section 7.1) in the following stage-by-stage iterations in the same way it was used in the first stage-by-stage iteration, when it really was arbitrary.

- A *condition* to be fulfilled whose definition is appended to the definition of the arbitrary intensity load object (see subsections A.1.12 and A.1.13) and contains three bits of information (see subsection A.1.14):

  - a *condition parameter* specifying which displacement or section force has to assume the condition value at a given point of the structure
  - a *condition value* specifying the absolute or relative value the condition parameter has to assume
  - a *condition stage* in which the condition has to be fulfilled.

  From an algebraic point of view it would not be strictly necessary to couple the conditions to an arbitrary intensity load. This coupling however is reasonable because it makes sure that in the input there are always as many conditions as arbitrary intensity loads to be updated in a given stage. This forces the user to think about the influence of the arbitrary intensity load on the condition parameter (reducing the risk that a non-solvable equation system is formulated) and allows the program to check that all arbitrary intensity loads are activated before or during, but not after, the condition stage.

- A *fixed load* consisting of all loads active during the condition stage belonging to the load case *load_history* excluding those whose intensity is being computed in the same condition stage.

7.1. The influence matrix algorithm

The linearity of first order static models allows the superposition of displacements and section forces: knowing the displacements and the section forces caused by given loads one can find the displacements and the section forces caused by any linear combination of those loads just by combining the respective displacements and section forces with the same linear factors used to combine the loads.

Conversely, knowing the displacements and the section forces caused by n loads of arbitrary intensity and having n conditions to fulfil in the form of prescribed displacements and section forces, one may find the linear factors needed for a conditions-fulfilling load combination. These linear factors are called *conditional factors* and they are found automatically by the program during the stage-by-stage iteration with the *influence matrix algorithm* developed in this project.

This is how the influence matrix algorithm works if all condition values are absolute:

1. During the stage-by-stage iteration, given n arbitrary intensity loads associated to n conditions having as a common condition stage, the stage actually selected by the stage-by-stage iteration, the nxn influence matrix $\mathbf{N}$ has to be built, where N_{ij} is the value assumed by the condition parameter defined in the condition i under the arbitrary intensity load j.

2. The nx1 vector $\mathbf{L}$ has to be built, where L_i is the value assumed by the condition parameter of condition i under the fixed loads.

3. The nx1 vector $\mathbf{C}$ has to be built, where C_i is the required condition value of condition i.

4. The following equation system has to be built and solved to find the vector $\mathbf{F}$ containing all the conditional factors:

$$\mathbf{NF} + \mathbf{L} = \mathbf{C} \Rightarrow \mathbf{F} = \mathbf{N}^{-1}(\mathbf{C} - \mathbf{L}) \tag{7.1}$$

Relative conditions state that the conditional load i has to induce a value of the condition parameter which is not absolute, but a portion of a section force or displacement at another point of the structure (e.g. "post-tension the cable with an intensity, such that the bending moment here is half the bending moment there"). If the condition is relative the corresponding factors for the matrices of eq. 7.1 are found as follows:

- N_{ij} is the difference between the value assumed by the condition parameter associated to the conditional load i (i.e. "the bending moment here") and the other section force or displacement multiplied by the proportion factor (i.e. "half of the bending moment there") under the arbitrary intensity load j.

- L_i is the difference between the value assumed by the condition parameter associated to the conditional load i (i.e. "the bending moment here") and the other section force or displacement multiplied by the proportion factor (i.e. "half of the bending moment there") under the fixed load.

- C_i is set to *0*.

The "same factor as" condition is used to avoid more conditional loads having the same condition parameter by stating that the conditional load i must have the same conditional factor as another conditional load j ($F_i = F_j$). Such conditions are expressed as follows in the influence matrix:

- on the i-th row of $\mathbf{N}$, N_{ii} and N_{ij} are set to 1 and -1 respectively,
- all other factors on the row i inclusive of L_i and C_i are set to *0*.

7.2. The need to perform the stage-by-stage iteration repeatedly to find the conditional loads

As explained in the introduction to this chapter a conditional load is constituted by an arbitrary intensity load coupled with a condition. The conditional factor needed to multiply this load is found with the influence matrix algorithm when, during stage-by-stage iteration, the condition stage specified in the condition is selected (see section 7.1).

Frequently the condition stage does not correspond to the activation stage of the conditional load (i.e. the stage in which the conditional load is added to the chronological objects list and begins to act on the structure) but it is a posterior one, e.g. often a cable has to be tensioned during construction to avoid a vertical displacement of its anchoring point on the deck in the final stage. If the condition does not correspond to the activation stage there are stages in between (activation stage included) in which, during the stage-by-stage iteration, the conditional load is already acting on the structure but with its still unmodified arbitrary intensity.

If the conditions of all conditional loads have to be fulfilled in the same stage or if new conditional loads are inserted in the chronological objects list after the stage objects representing the condition stages of all conditional loads formerly defined, the conditional factors found with the first stage-by-stage iteration are the correct ones because no conditional load of unmodified arbitrary intensity was considered as part of the fixed load.

By contrast, if conditions have to be fulfilled for an earlier stage in which other conditional loads whose condition has to be fulfilled in a later stage are already active, the factors of the conditional loads computed in the earlier stage are incorrect after the first stage-by-stage iteration because they have been computed with a fixed load containing also the still arbitrary intensity of the conditional loads determined at a later stage.

When during the stage-by-stage iteration conditional factors are found, they are immediately used to correct the arbitrary intensity of the arbitrary intensity loads they are associated to; hence the influence matrix algorithm can converge if the whole stage-by-stage iteration is performed repeatedly (leaving out the case in which locked-in displacements are taken into account explained in subsection 7.3.2) because the errors manifest themselves as wrong values of the vector $\mathbf{L}$ and are compensated in the next stage-by-stage iteration (see equation 7.1).

A practical example of such a situation is given by bridges with composite decks in which the cables are post-tensioned twice: the first time to induce a given deflected shape in the formwork element during the casting stage and a second time for the final construction stage (see [Schlaich] for more details). In this case the arbitrary intensity of the conditional loads with the condition to be fulfilled in the final stage, being considered as part of the fixed load, induce an error in the conditional factors of the conditional loads with a condition to be fulfilled during the casting stage and it is necessary to repeat the stage-by-stage iteration two or more times.

7.3. Convergence of the influence matrix algorithm with non-linear models

As shown in the practical example of chapter 9 the influence matrix algorithm does not need to converge for non-linear models to be applicable, since the conditional load intensities found with a linear model are exact enough and lead to acceptable values of the conditional parameters even if non-linear effects are taken into account.

However, the convergence behaviour of the influence matrix algorithm combined with the repeatedly performed stage-by-stage iteration has been studied and the following sub-sections explain the results of this investigation.

7.3.1. Creep and shrinkage

The influence matrix combined with the stage-by-stage iteration converges quickly also if concrete creep and shrinkage are being taken into account, because these phenomena do not change the material stiffness. The errors they induce in the values assumed by the condition parameters appear in the vector $\mathbf{L}$ and are also automatically corrected by the influence matrix algorithm when performed during the next stage-by-stage iteration.

7.3.2. Locked-in displacements

During the stage-by-stage iteration it is possible to take locked-in displacements into account *and* to compute the conditional factors, and it may happen that the condition stage of every conditional load is not always the same as its activation stage but a later one. In this case (once the stage-by-stage iteration reaches the condition stage) a new intensity is assigned to the conditional load which would have affected the locked-in stresses found in all stages between the activation stage and the condition stage. The consequence of this is that at the end of such a stage-by-stage iteration the model is in a physically inconsistent state, because the locked-in displacements do not reflect the newly found conditional load intensities.

This physical inconsistency leads to the necessity of always performing the stage-by-stage iteration twice (which is automatically done by the program BRIDE) when computing the conditional factors taking locked-in displacements into account:

1. During the first stage-by-stage iteration locked-in displacements are considered *and* the intensities of the conditional loads are computed.

2. During the second stage-by-stage iteration just locked-in displacements are considered while the intensities of the conditional loads are kept unchanged. In this way the locked-in displacements are updated to the new intensities of the conditional loads.

There is another curious effect which impedes the convergence of the influence matrix algorithm when repeatedly performing the stage-by-stage iteration.

Locked-in displacements amplify the effect on the deflected shape caused by the changes of the conditional load intensities between subsequent stage-by-stage iterations in such a way that the self-correcting behaviour of the influence matrix algorithm explained in section 7.2 cannot take place. In other words, when taking locked-in displacements into account the structure "over-reacts" to the changes in the conditional load intensities which occur when the stage-by-stage iteration is performed repeatedly and in each stage-by-stage iteration the conditional factors are computed again: instead of converging to the condition values the values of the condition parameters oscillate around the condition values with growing distance. This fact can be observed by the user with the graphic representation of the deflected shape which is updated every time a stage-by-stage iteration is completed.

The *error feed-back factor* α ranging from 0 to 1 has been introduced to avoid such oscillation and has to be set interactively by the user when the stage-by-stage iteration is repeatedly performed on the basis of the observed variation of the deflected shape. Let's call $\mathbf{R}$ the residuum affecting the condition parameters after a stage-by-stage iteration, the residuum $\mathbf{R}$ multiplied with the feed-back factor α is added as follows to the right side of eq. 7.1 in the next stage-by-stage iteration:

$$\mathbf{NF} + \mathbf{L} = \mathbf{C} + \mathbf{R} \cdot \alpha \tag{7.2}$$

If α is set to 0 eq. 7.2 equals the self-correcting eq. 7.1. If α is set to 1 $\mathbf{F}$ cannot change, hence the (error-affected) solution is stabilized. Any value of α between 0 and 1 makes the influence matrix algorithm converge more or less quickly. While repeatedly performing the stage-by-stage iteration computing both conditional loads and locked-in displacements the user should keep α as close as possible to 0 but big enough to avoid the oscillation of the values of the condition parameters about the condition values.

7.3.3. Second order effects and cable sag

Second order effects and cable sag[2] are stiffness changing phenomena which transform eq. 7.1 as follows:

[2] The cable sag is taken into account in the program BRIDE using the parabola-based formula for the tangent Young's modulus E_{tan} derived in [Gimsing, p. 141]:

$$E_{tan} = \frac{E}{1 + \frac{\gamma^2 l^2 E}{12\sigma^3}}$$

$$(\mathbf{N} + \triangle\mathbf{N})\mathbf{F} + \mathbf{L} = \mathbf{C} \tag{7.3}$$

where $\triangle\mathbf{N}$ is the change in the influence matrix due to a change in the stiffness of the structure.

This stiffness change can be taken into account by computing the influence matrix using the updated stiffness of the structure.

If locked-in displacements are not taken into account (hence abrupt changes of the conditional factors between subsequent stage-by-stage iterations do not cause the over-reaction of the structure explained in section 7.3.2) the result $\mathbf{F}$ can be improved by performing a final stage-by-stage iteration in which the error $\mathbf{R}$ is subtracted from the condition values (in eq. 7.2 this would correspond to $\alpha = -1$). With this last iteration the error in the newly computed conditional factors is reduced dramatically. This is what happens numerically:

1. In the last stage-by-stage iteration, once the error has stabilized itself, instead of explicitly finding $\mathbf{N} + \triangle\mathbf{N}$ we can assume that $\triangle\mathbf{NF}$ equals the residuum $\mathbf{R}$, being $\triangle\mathbf{NF}$ the only term in eq. 7.3 differing from eq. 7.1 .

2. The correct $\mathbf{F}$ can be found by re-formulating equation 7.3 as follows:

$$\mathbf{NF} + \mathbf{L} = \mathbf{C} - \mathbf{R} \tag{7.4}$$

   which is what is done in the last stage-by-stage iteration if the user has chosen this option.

7.4. Conclusions regarding the conditional loads

Not every conditional load leads to a solvable equation system: if the user tries to induce a displacement with an arbitrary intensity load j having no influence on the condition parameter i (i.e. $N_{ij} = 0$), it is quite possible that the matrix $\mathbf{N}$ is going to be singular. Another unsolvable situation is given if the same condition parameter is used more than once, as this leads to two identical rows in $\mathbf{N}$ (which should be avoided using a "same factor as" condition). More generally it can be said that the major problem in the formulation of conditional loads is that the user has to have a good understanding of the behaviour of the structure.

The important aspect to be considered when formulating an arbitrary intensity load is not its absolute intensity which can be corrected in a few linear stage-by-stage iterations even if it is very different from the appropriate value, but the way the arbitrary intensity load influences the static system and specially the condition parameter.

Another problematic aspect of the influence matrix algorithm is that condition parameters representing section forces (and not displacements) can lead to near-singular

where E is the Young's modulus and γ is the weight per unit volume of the cable's material, l is the cable's length and σ the stress in the cable.

influence matrices not delivering the conditional factors exact condition fulfilment. Also in this case a good understanding of the structure's behaviour is required from the user, who, in some cases, might be forced to choose alternative conditions prescribing displacements.

8. Automatic cable dimensioning

Automatic cable dimensioning has been implemented in the program BRIDE. In fact, cable dimensioning is a tedious and time-consuming task if performed manually.

To perform an automatic cable dimensioning the user has to specify the cable dimensioning objects (see subsection A.1.15) in the chronological objects list and simply click on a button of the graphic user interface.

8.1. Cable dimensioning algorithm

When the user clicks on the cable dimensioning button the program finds for every cable to be dimensioned the smallest sections which resist all normal forces experienced by the cables in every construction stage and load combination pair specified in the cable dimensioning objects.

To dimension a cable it is necessary to know its normal force which can be found only by means of structural analysis. In turn, structural analysis can be performed only if a section has already been assigned to the cable. Thus the cable sections cannot be dimensioned directly but only iteratively by first adopting an approximative section. If this is very different from the adequate one it may be necessary to iterate (by simply clicking the cable dimensioning button) two or three times. In addition sections must be chosen from an appropriate pool of predefined cable types.

This is how the section re-dimensioning problem has been solved:

1. Every section object is assigned to a section group with an identifier (see subsection A.1.6).

2. As for all other elements the user has to define a section for every cable, even if they are going to be re-dimensioned (see subsection A.1.5).

3. While performing the automatic cable dimensioning the program uses the section specified by the user as the initial approximative section and looks for the smallest section under the sections belonging to the same section group.

9. Example of an input file for the program BRIDE

In this chapter an example of a computation performed with the program BRIDE is discussed. The model used for the computation has been freely obtained by merging different models used to test the program during its development so that it can show all implemented features.

Our imaginary bridge is symmetric and has been defined using two files. At the beginning of the main file (see section D.1 and figure 9.1) all materials and sections needed are defined. After the material and section definitions, a second input file (see section D.2 and figure 9.2) in which one half of the model is defined keeping the signs of the X and Y coordinates and all identifiers dependent on a single variable "s" (to be defined before inclusion, as mentioned in figure 9.1) is included twice (once for each model side). Between the first and the second file inclusion instruction a restart object is inserted. After this double inclusion all objects needed to join together the two halves and to dimension the cables are defined.

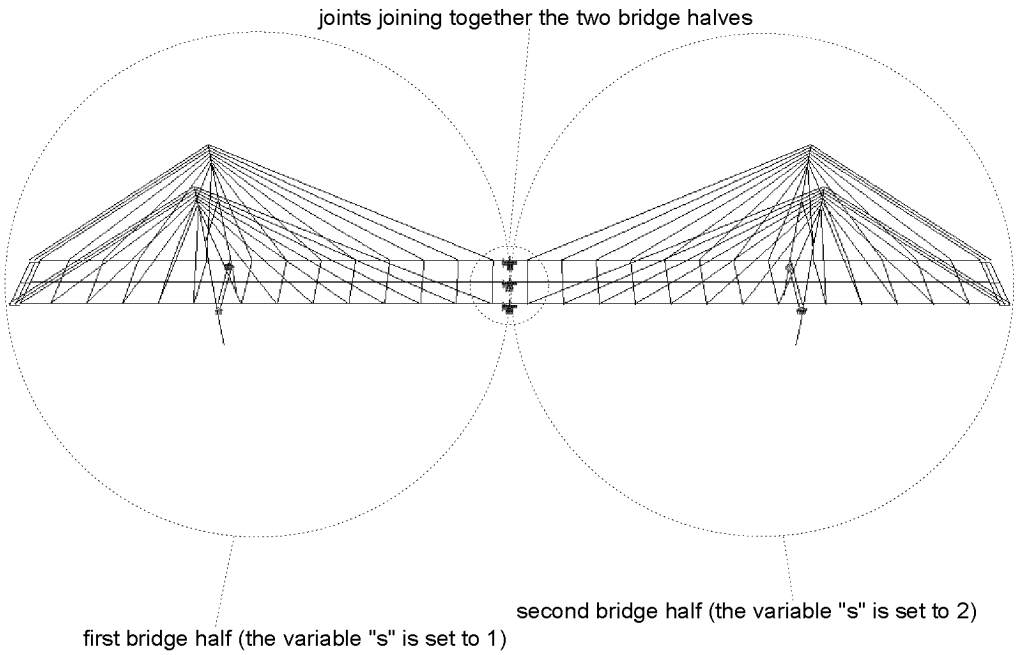


Figure 9.1.: The two bridge halves joined together in the main file.

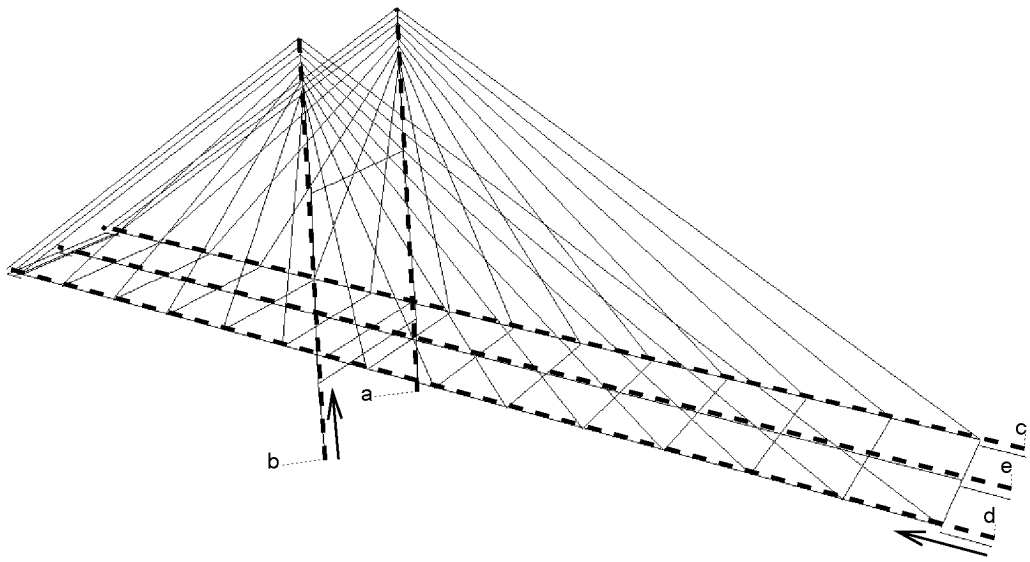


Figure 9.2.: Half bridge defined in the included file with the convention used to define the identifiers: the objects on a dashed line have the letter assigned to that line and a number growing along the line in the direction of the arrow in the identifier. Notice that the formwork beams in the middle of the mid-span (where the letters "c", "d" and "e" are) have not been merged yet with the composite beams which are defined in the outer file, hence they appear excentric relatively to the other's composite beams.

This definition with two files has the advantage of allowing an easy formulation of the model for a single side (which is useful during construction, see section 9.2) and to keep a better overview of the way the two sides are joined together.

The model has a composite deck modelled with composite beams (see figure 9.3). Another example with cast beams instead of composite beams is not shown because the use of composite and cast beams is fully analogous. In fact a model with cast beams instead of composite beams can be obtained from the model by simply replacing all occurrences of the keyword "CompositeBeam" with the keyword "CastBeam" (see figure 9.4).

9.1. Analysis strategy

After having completed the writing of the input file(s) the user has to open it with the program BRIDE (actually it is advisable to open the input file also when it is not yet completed, so as to track all errors as soon as possible using the error messages from the program).

As mentioned in chapter 7, the stage-by-stage iteration does not always converge when taking all non-linear effects into account. However, it has been observed that such convergence is actually not necessary since the intensities of the conditional loads found

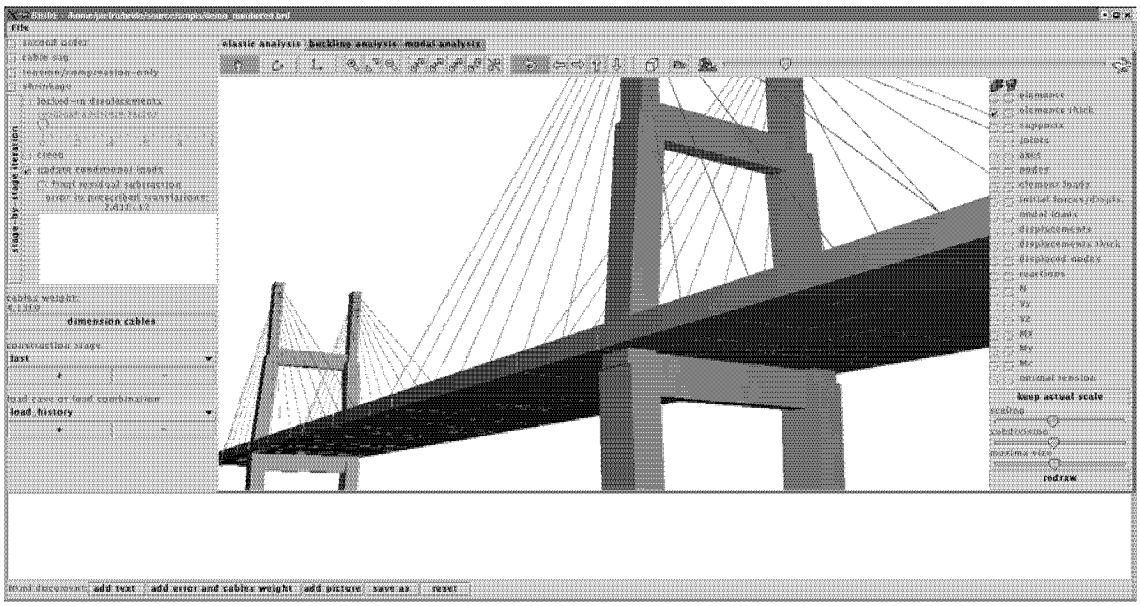


Figure 9.3.: The model listed and commented in appendix D with its composite deck. The box shape of the element only shows the maxima and the minima of the element's section and has nothing to do either with its stiffness or its self-weight.

with a linear model lead to a satisfactory precision in the conditional parameters also when, after having found the conditional loads linearly, non-linearities are taken into account in the model. This is plausible since otherwise it would be incomprehensible that cable-stayed bridges could be successfully built with the manual methods listed in section 1.2. For this reason a two step strategy has been used:

- All model-changing computations such as the computation of the conditional load intensities and the cable dimensioning are first performed using a linear model. The model modified by the computation of the conditional loads and of the cable sections is then stored as an expanded input file (i.e. an input file without pre-processor instructions and in which the intensities of the conditional loads and the cable sections have been updated, see figures from 9.5 to 9.10).

- The stored model is opened with the program BRIDE. All non-linear phenomena[1] such as second order, cable sag, shrinkage, locked-in displacements and creep are taken into account with a stage-by-stage iteration, during which the intensities of the conditional loads are left unchanged (see figures from 9.11 to 9.14).

[1]Tension/compression-only behaviour is not taken into account at this stage of the analysis since the engineer is asssumed to have designed the structure in such a way that an unforseen construction-history- relevant loss of contact of a support and the loss of tension in a cable, respectively, does not occur.

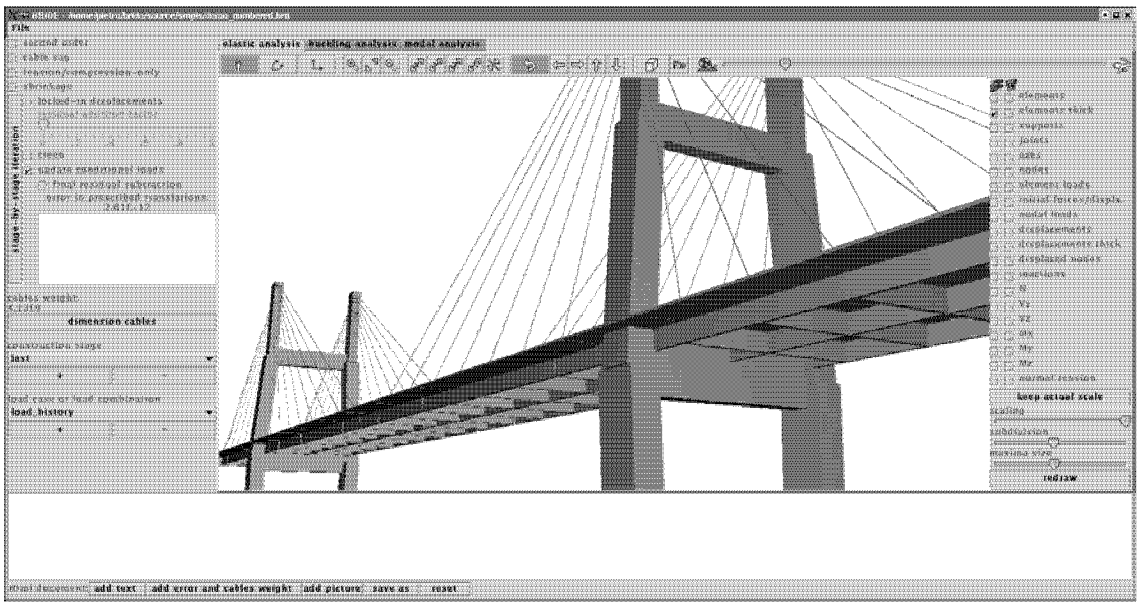


Figure 9.4.: The model listed and commented in appendix D where its composite beams are substituted by cast beams simply by replacing all occurrences of the keyword "CompositeBeam" with the keyword "CastBeam" in the input files.

9.2. Monitoring the bridge erection by keeping the input file up-to-date

During the construction of the bridge the input file has to be kept up-to-date with the events in situ: if a cable has been post-tensioned with its intensity found using a conditional load, this conditional load definition in the input file has to be transformed into a regular load by removing the condition and by substituting the computed intensity with the intensity measured while post-tensioning the cable.

If a deviation from the planned geometry is observed, e.g. due to an unforseen creep behaviour of concrete, the engineer on site should update the input file by producing this deviation in the model by means of a conditional load. Once the deviation has been introduced into the model the engineer can investigate which adjustments have to be undertaken, e.g. introducing additional conditional loads.

At the end of construction such an input file kept up-to-date also represents a detailed report of all relevant events that occurred during the erection of the bridge.

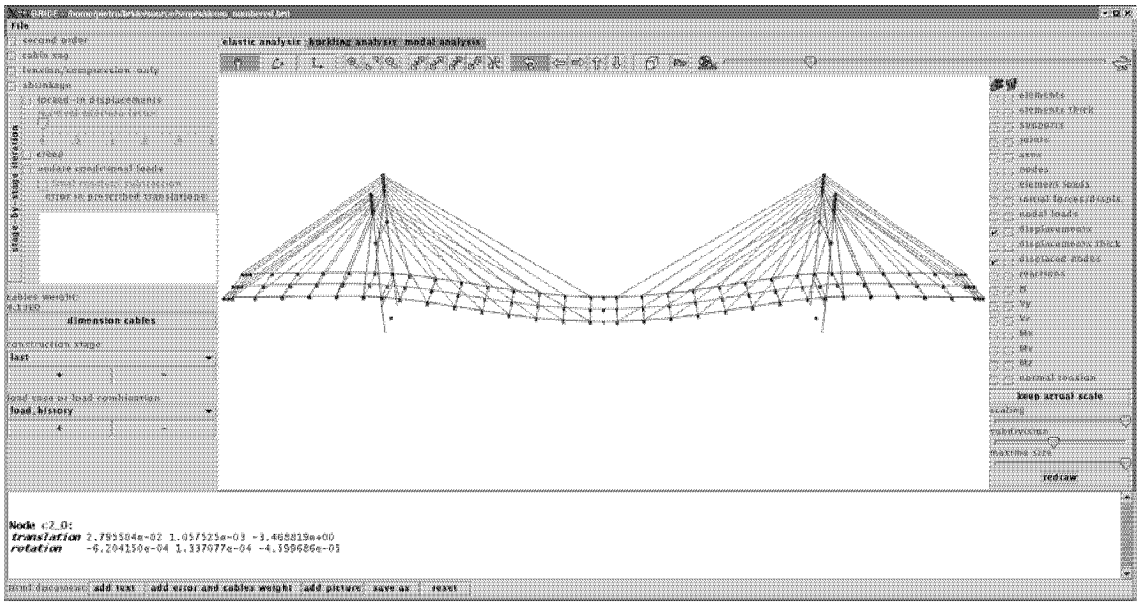


Figure 9.5.: Step 1. The file listed in appendix D is opened and its displaced shape in the *last* construction stage and under the *load_history* load case is presented. A displaced node in the middle of the mid-deck has been selected and its displacement values are displayed in the text panel (the units are meters and radians). The model has no pre-camber, no post-tensioning and the cables are too flexible thus the displacements are obviously too big.

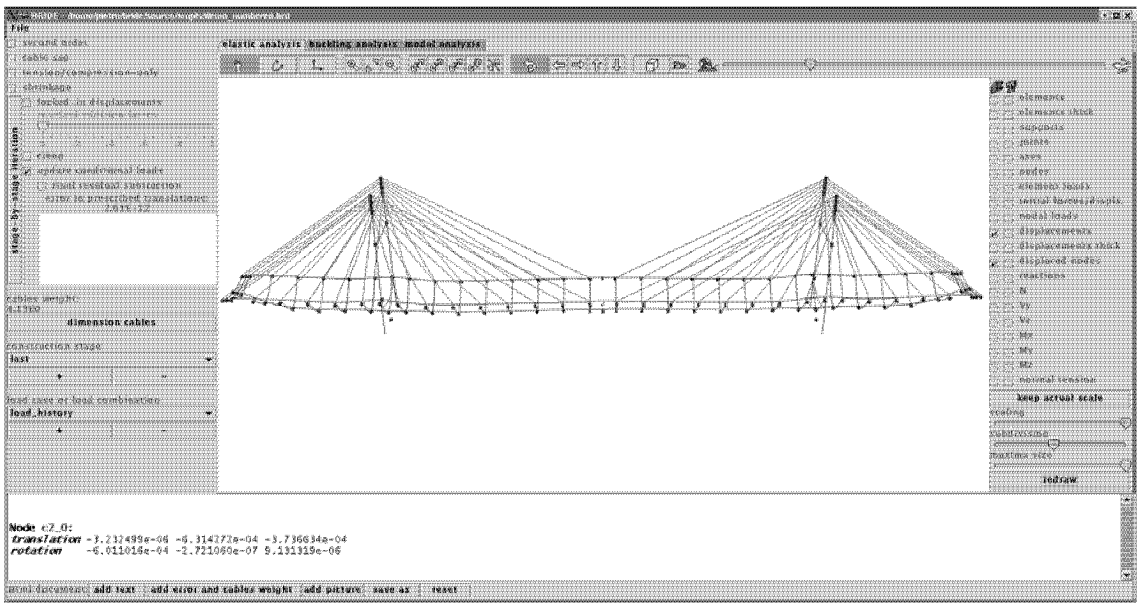


Figure 9.6.: Step 2. One linear stage-by-stage iteration (i.e. not taking non-linearities into account) is performed in which the conditional loads are found.

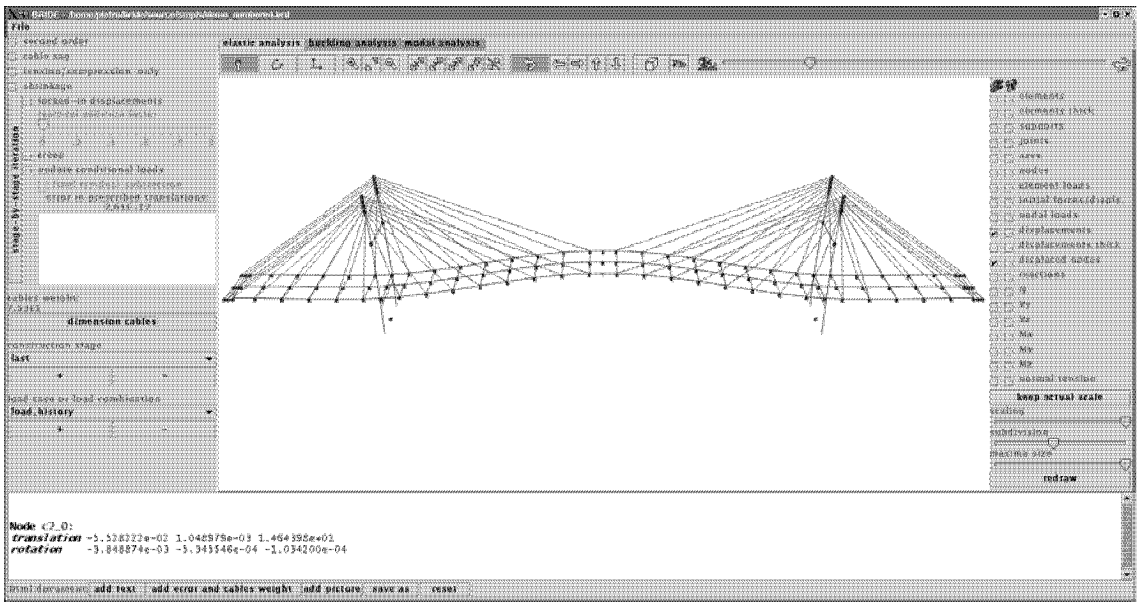


Figure 9.7.: Step 3. The cables are dimensioned. The change of stiffness of the cables makes the conditional loads out of date.

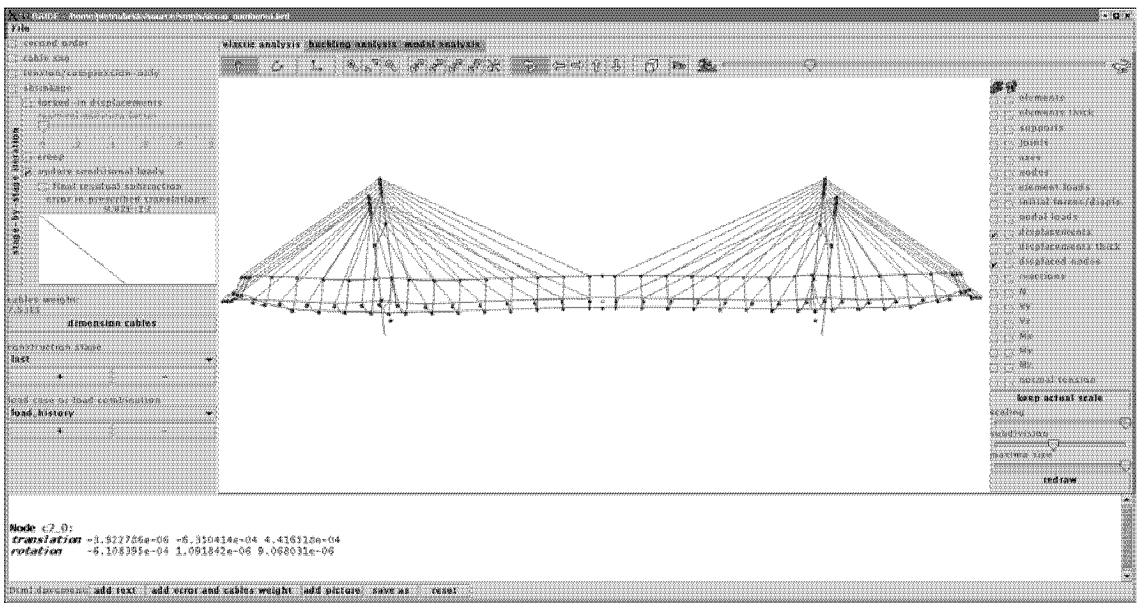


Figure 9.8.: Step 4. Another linear stage-by-stage iteration is performed in which the conditional loads are updated to the new stiffness of the cables.

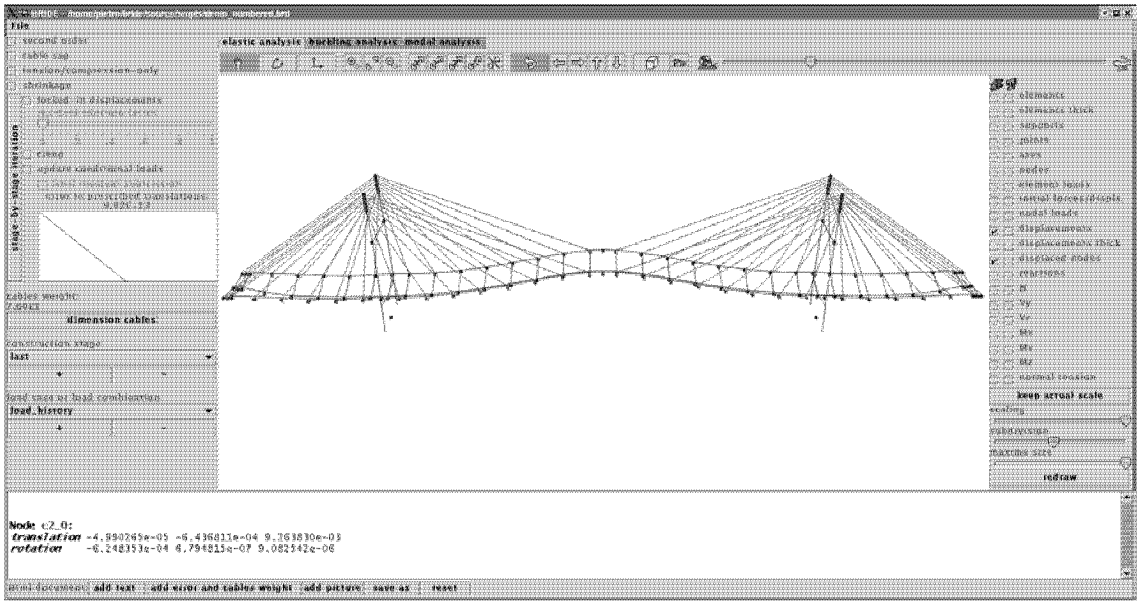


Figure 9.9.: Step 5. With this model, after having updated the conditional loads the cables need to be dimensioned another time, which makes the conditional loads being out of date again.

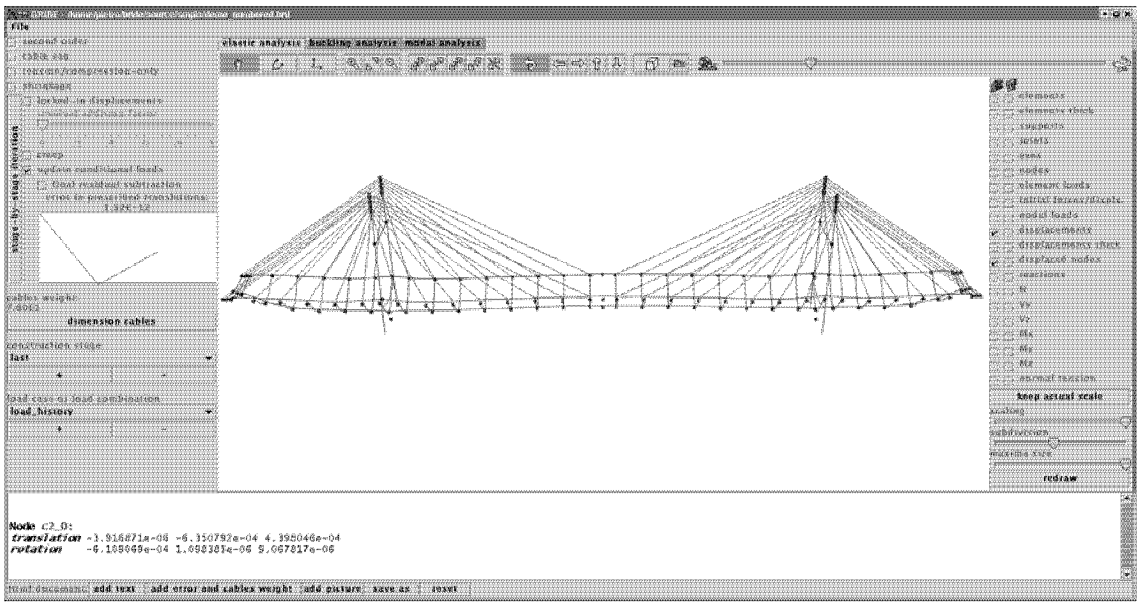


Figure 9.10.: Step 6. A last linear stage-by-stage iteration is performed in which the conditional loads are updated to the new cable stiffnesses. Now the cable sections are the right ones and are no longer changed by the program even if the button for the cables dimensioning is pressed again.

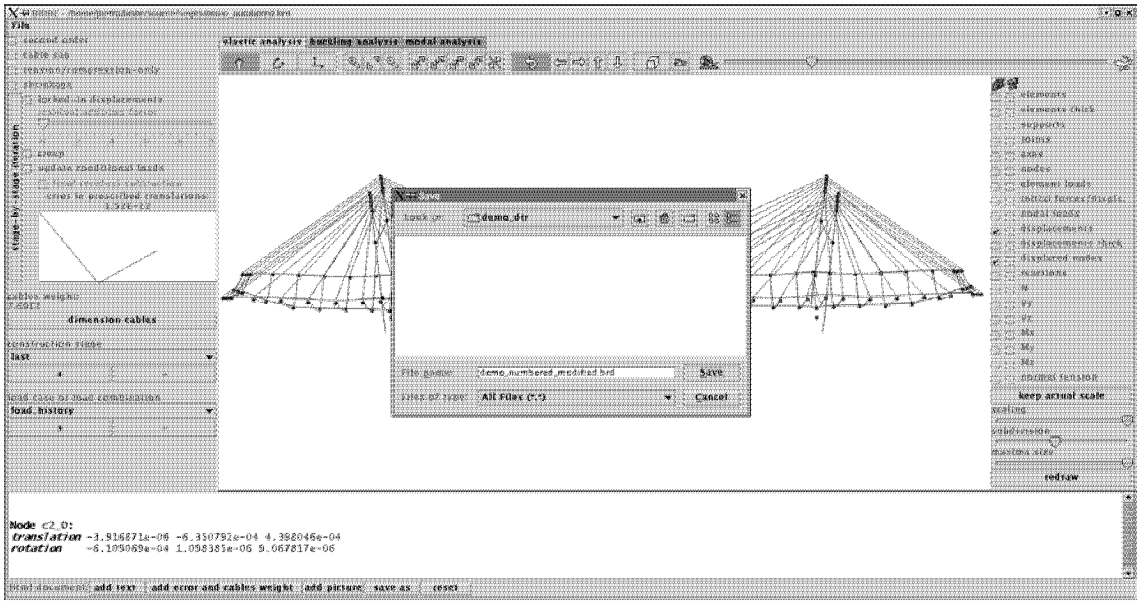


Figure 9.11.: Step 7. The model with the dimensioned cables and the updated conditional loads is saved as a new input file.

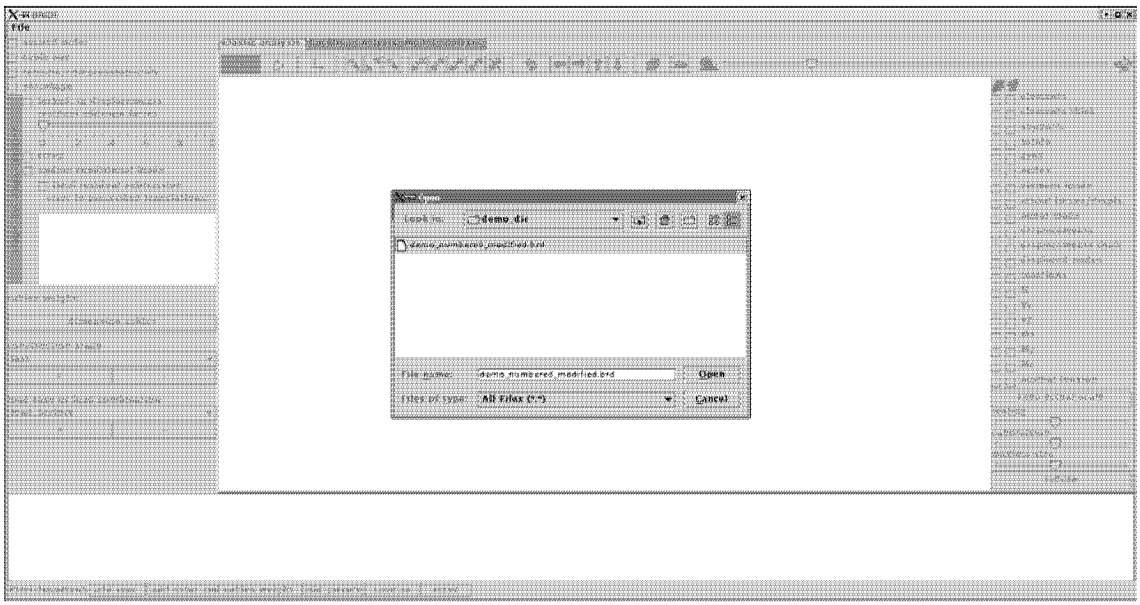


Figure 9.12.: Step 8. The original file is closed with no changes and the file stored in step 7 is opened instead.

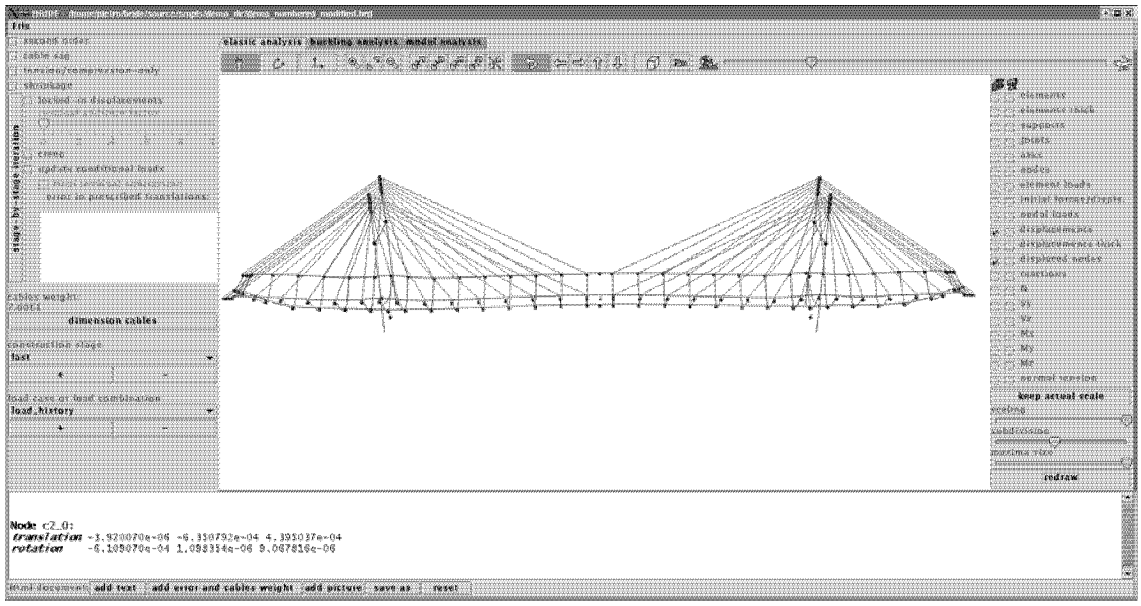


Figure 9.13.: Step 9. The slight differences with respect to the nodal displacements in step 6 are due to the rounding of numeric values when writing the stored file to disk.

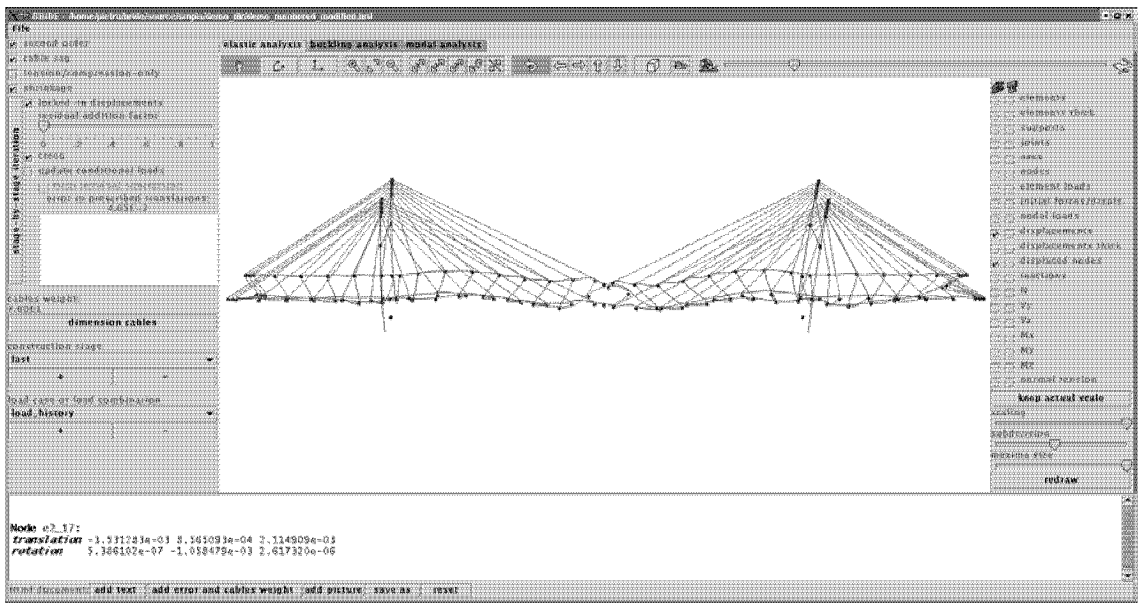


Figure 9.14.: Step 10. Finally, a stage-by-stage iteration taking all non-linearities into account and not updating the conditional loads can be performed. The biggest displacements can be checked. In fact, they should be in an acceptable range (here a node at the centre of the deck close to the right anchor blocks has been selected).

10. Conclusions

In its present form the program BRIDE has not yet reached the maturity required for commercial use. Nevertheless, it represents a prototype with which original approaches to the important and non-trivial practical problem of the design, erection and retrofitting of cable-stayed bridges could be developed.

Three concepts have been introduced and developed which allow one to model a cable-stayed bridge in a very effective way:

Chronological objects list: when describing the construction of a cable-stayed bridge not only the information about which parts are added to the structure, but also the information about when they are added to the structure is important. The chronological object list concept allows one to include this information in the input file, i.e. in a natural way.

Stage-by-stage iteration: creep and locked-in displacements at a given time depend on the structure's previous deflected shape. Or, for the same reason, having an influence on the deflected shape, they are also going to have an influence on all subsequent deflected shapes. The stage-by-stage iteration allows one to take all these phenomena into account in a simple but effective way by modelling all construction stages in chronological order and by retrieving the effects of creep and locked-in displacements from the deflected shapes of previous construction stages. The adoption of the stage-by-stage iteration also allowed the development of quite innovative ways of taking creep and locked-in displacements into account.

Conditional loads: pre-camber and post-tensioning can be modelled as initial displacements, but how large should be such initial displacements? The newly introduced conditional loads are an elegant solution to this problem, in which the user does not specify the absolute intensity of loads explicitly but through a condition to be fulfilled.

Since the program BRIDE allows one to accurately plan and optimize the post-tensioning schedule, the added value deriving from its use may extend not only to the time saving for the civil engineer responsible for design and construction but possibly also in a reduced number of times the forces in the cables need to be readjusted.

The rather interdisciplinary nature of this project (structural engineering, materials, information technology) required a very summarizing approach from the author. Now that the project's skeletal structure has been established, it would be desirable that further research and validation work on the individual aspects and assumptions of the developed solution algorithms were performed. Such research would also be helpful for

the consolidation of the approach developed in this project which, despite its promising effectiveness, still needs from the user a good understanding for the static system's inner logic.

The program BRIDE has been implemented for the simulation of three-dimensional models. However, for preliminary design a two-dimensional analysis is often sufficient. This restriction would allow to simplify the graphic user interface and the computation in such a way that a much higher degree of user interaction could be reached than with a three-dimensional program. It is the opinion of the author that the development of a two-dimensional but highly interactive program implementing the original approach to the analysis of cable-stayed bridges adopted for the program BRIDE would be welcome to many users.

Bibliography

[Anderheggen] E. Anderheggen, Lineare Finite-Element-Methoden: eine Einführung für Ingenieure, Institut für Baustatik und Konstruktion, ETH Zürich, 1993

[CEB/FIP] Structural Concrete, Textbook on Behaviour, Design and Performance, Updated knowledge of the CEB/FIP Model Code 1990, Volume 1, International Federation for Structural Concrete (fib), 1999

[FAQs] M. Cline, G. Lomow, M. Girou, C++ FAQs, Second Edition, Addison-Wesley, 1998

[Gimsing] N. J. Gimsing, Cable supported bridges concept and design, Second edition, John Wiley &Sons, 1997

[Java 3D] Java 3D API Tutorial, on-line document, URL: http://java.sun.com/developer/onlineTraining/java3d/

[Java Tutorial] The Java Tutorial, on-line document, URL: http://java.sun.com/docs/books/tutorial

[JNI] R. Gordon, Essential JNI: Java Native Interface, Prentice Hall, 1998

[Patterns] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

[Refactoring] M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999

[Schlaich] M. Schlaich, Erection of Cable-Stayed Bridges Having Composite Decks with Precast Concrete Slabs, Journal of Bridge Engineering, September/October 2001

[Shaw and Whiteman] S. Shaw and J. R. Whiteman, Some Partial Differential Volterra Equation Problems Arising in Viscoelasticity, Brunel University, Uxbridge, England, 1997, URL: www.hindawi.dk/books/9775945054/B9775945054000125.pdf

[Stroustrup] B. Stroustrup, The C++ Programming Language, Third Edition, Addison-Wesley, 1998

[Wittmann] F. H. Wittmann, Werkstoffe im Bauwesen, ETH Zürich, 1989

A. How to define a numerical model for the program BRIDE

In the former chapters it has often been mentioned that the numerical models in the program BRIDE consist of a chronological list of model objects specified by the user, but it was not explained how the user is supposed to specify them. This appendix closes this gap.

Every numerical simulation program needs an input produced by the user for describing the model to be analyzed. Depending on the program, this may be specified interactively through a graphical user interface or in written form as a text file using a problem-specific input language. Both alternatives have advantages and disadvantages. A graphical input is more intuitive for the inexperienced user but less transparent than a text input which allows the experienced user to have full control of the model.

There are also solutions which are a combination of graphical and text input. Such solutions offer both user friendliness and transparency and consist mostly of a graphical preprocessor which generates the text input to be read by the main program. Even if the user inputs graphically, the generated text input remains accessible for correctness checks and for the re-editing of some details.

For the program BRIDE the text input solution has been chosen because of its transparency. In addition, a text input allows the implementation of programming-language-like constructs by means of a preprocessor (see section A.2).

In the program BRIDE to every model object there corresponds a text instruction in the input file defining it. These are called *model instructions*. The type of model objects defined in the input file can alternately be free, provided they all belong to the same construction stage. This has two important advantages:

- It allows to group together the information belonging together (for example by putting a new node used in a new element right before the new element) so that it is not necessary to search the whole input file for already defined information while writing it. This principle has been borrowed from modern programming languages like C++ or Java, which for the same reason allow one to declare a variable where it is needed in the code and not, like in older programming languages, at the beginning of a block.

- It allows one to write an input file which contains the information about the construction chronology just by inserting the different model objects in the input file in the same sequence in which their physical counterpart are built in the real world. Such an input file appears like a construction schedule, which is the clearest and simplest way to specify the complex chronology of the construction process.

A. How to define a numerical model for the program BRIDE

A finite element model needs the following information:

1. Nodes: each node is defined by an identifier and by its spacial coordinates. In each node six displacement parameters are introduced.

2. Elements: each element is defined by an identifier, by its physical attributes (like section and material) and by the identifiers of the two nodes to which the element is connected ("incidence nodes").

3. Supports: supports are similar to elements, but they have just one incidence node.

4. Loads: typically loads can act on elements or directly on nodes.

The syntax for the model instructions is constituted following this schema:

1. A keyword uniquely defining the type of the object (e.g ***Node***, ***Support***, ...).

2. An identifier uniquely defining the object consisting of letters, digits and the underscore character (e.g. *deck_node_1*, *east_support* ,...). The first character must be a letter. The identifier is case- sensitive and may not conflict with the keywords of the text preprocessor (see section A.2).

3. The information building the content of the object[1].

4. A semicolon ends the input object.

Blanks and end of lines are treated as separators.

In the following sub-sections the explanation of the syntax for the model instructions is limited to those parts of the railroad diagrams[2] which are not self-explanatory.

In this project a further formalism has been introduced for the railroad diagrams:

[1]Internally the program does not transform the values found in the input file in order to change their units. The only constant used in the program with predefined physical units is the gravity acceleration of $9.81\frac{m}{s^2}$ acting in the negative global Z-direction. The set of metric units which has been taken into account during program development is the following:

length	meter	m
force	kilo newton	kN
mass	ton	t
stress	kilo Pascal	kPa

The user is assumed to know the physical meaning of the values he defines in the input file, and thus to transform them accordingly whenever needed.

[2]Railroad diagrams are used to graphically represent a syntax and consist of a line (the track) interrupted by stations representing the so-called "tokens" and the switches representing alternatives routes. To define an object it is necessary to read its diagram from the upper left corner, to copy to the input file the tokens encountered and to insert the numeric values required along the track. When a switch is encountered, the user can choose one direction to continue following the diagram.

- bold is used for keywords with a predefined semantic meaning (they do not need to be written completely: one or more characters from the keyword's beginning are enough as long as they unequivocally define it),
- italic is used for user-defined tokens (in most cases identifiers or numbers in integer or floating-point notation), the italic text explains which information is required,
- boxes refer to other railroad diagrams which, ideally, have to be inserted at the position of the box.

A.1. Syntax for the definition of model objects

A.1.1. Stage object

In the chronological objects list the stage object signalizes when a new stage has been reached and gives it an identifier and a day number. The day number is necessary for the calculation of time-dependent effects and for the merging of many object lists using the *Restart* object (see subsection A.1.2).

The syntax for a stage object is:

Stage *stage_identifier* **DayNumber** *day_number* **;**

The day number must be positive and bigger than the day number of the former stage (unless it is the first stage coming after a restart object). It specifies the day when the construction stage is reached.

A.1.2. Restart object

The *Restart* object allows one to merge many objects lists and belongs at the beginning of a chronological objects list, being appended to an already defined one. This should be used for structures consisting of different sub-structures which stand alone before being joined together at the end of construction (typically the cantilever portions of a cable-stayed bridge attached to different pylons). It allows one to keep the input files for the sub-structures in different smaller files first - as long the sub-structures are not joined together - and then to easily merge them.

The syntax for a *Restart* object is:

Restart ;

The merging of different input files is best done by writing an additional short input file containing just the *Restart* object(s), the input files for each sub-structure being inserted using the preprocessor statement *Include* (see subsection A.2.6) and the additional objects needed to specify the junction elements between the sub-structures.

A.1.3. Remove object

The *Remove* object allows one to detach single model objects so as to simulate the removal from the real structure of temporary members (e.g. formwork), supports or loads. This is its self-explanatory syntax:

```
Remove ┬ Element element_identifier ──────────┐
       ├ Support support_identifier ──────────┤
       ├ Joint joint_identifier ──────────────┤ ;
       ├ NodeLoad node_load_identifier ───────┤
       └ ElementLoad element_load_identifier ─┘
```

A.1.4. Nodes

Each node is defined by an identifier and its spacial coordinates. Syntax for a node object:

```
Node node_identifier X_coordinate Y_coordinate Z_coordinate ;
```

- The three numeric values *X_coordinate*, *Y_coordinate*, and *Z_coordinate* are the absolute coordinates of the node's location in the program's arbitrary Cartesian system with the Z axis pointing upward (see figure A.3).

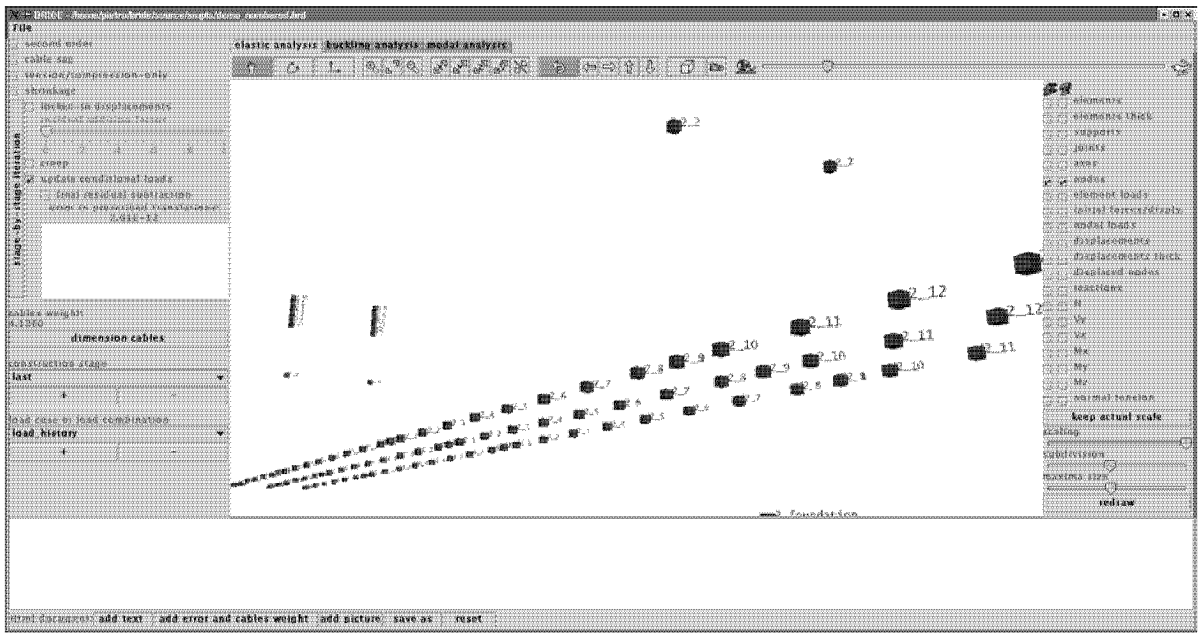


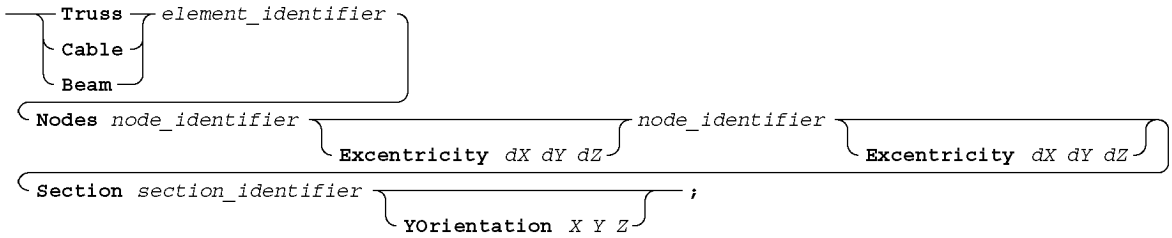
Figure A.1.: A snapshot of the program BRIDE showing the nodes of a cable-stayed bridge model with their identifiers.

A.1.5. Elements

Elements are three dimensional straight bar elements connecting two nodes. Each element is defined by an identifier, its physical attributes and the identifiers of the two nodes to which the element is connected. Possible element types are:

- Truss elements: transmit only a normal force along the element's axis, with no bending stiffness
- Cable elements: same as truss elements, but they can be automatically re-dimensioned by the program. They take tension-only response and sag into account.
- Beam elements: transmit bending and shear forces in two directions, torsion and a normal force. They correspond to the classical model for slender beams ignoring shear deformations.
- Cast beam elements: same as beam elements but allow the consideration of locked-in deformations of cast concrete.
- Composite elements: same as cast beam elements, but allow the consideration of the bond between the concrete and the steel profile.

The following syntax is used to define a truss, a cable, or a beam element:



- The two node identifiers define the nodal incidences.
- The three numeric values after the optional "Excentricity" keyword are the components of a vector starting from the incidence node and pointing to the element start- or end- point defining an excentric, rigid connection of the element relative to the incidence node.
- The section identifier after the "Section" keyword defines the section properties which are assumed to be constant over the length of the prismatic element.
- The optional three numeric values after the "YOrientation" keyword allow one to prescribe explicitly the orientation of the element's local coordinates system. Their numeric values give the direction of a vector co-planar to the local x axis, which is always on the element's axis, and the local x-y plane (see figure A.3). If the user does not prescribe the orientation of the element's local coordinates system

explicitly the program assumes that the local y axis is parallel to the global X-Y plane and, if the element is vertical, that it is parallel to the global Y axis.

The following syntax is used to define a cast or a composite beam, whose idealization is represented in figure 2.5:

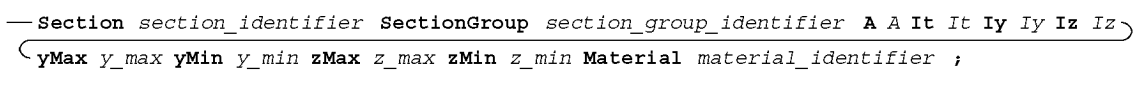
- Cast and composite beams must be coupled to a previously defined beam element (*formwork_beam_identifier*) which is intended to model the structural behaviour of the formwork or of the steel profile of a composite section. The incidence nodes and the orientation of the local coordinates system are the same for both formwork and cast/composite beams, and so do not have to be specified.
- *Dz* is the distance between the axis of the formwork beam and the axis of the cast/composite beam in the local z coordinate.
- The section specified is that of the cast/composite beam alone, regardless of the fact that the section of composite beams is automatically merged by the program with their formwork section into a single combined section.

If a cast beam is defined, the cast concrete and formwork sections are kept separated using two different beams with no shear connection between them but a rigid excentric connection of their incidence nodes. If a composite beam is defined, its cast concrete section (the section specified in the definition of the composite beam) is merged together with the formwork section in a single combined section carrying the identifier of the composite beam, while the formwork beam is removed from the model after the activation of the composite beam. This allows one to take into account the shear connection between cast concrete and steel, which is always assumed to be rigid.

A.1.6. Sections

Several elements often have the same section, so for this reason the section definition is decoupled from the element definition. A complete section specification is needed for all the implemented element types even if it makes little sense to define e.g. a non-vanishing bending stiffness for a cable as the program would ignore it in the calculation.

The syntax to define an element section is:



The *section_group_identifier* is used to define sets of sections belonging to the same section group. This is needed when re-dimensioning the cable sections as the program

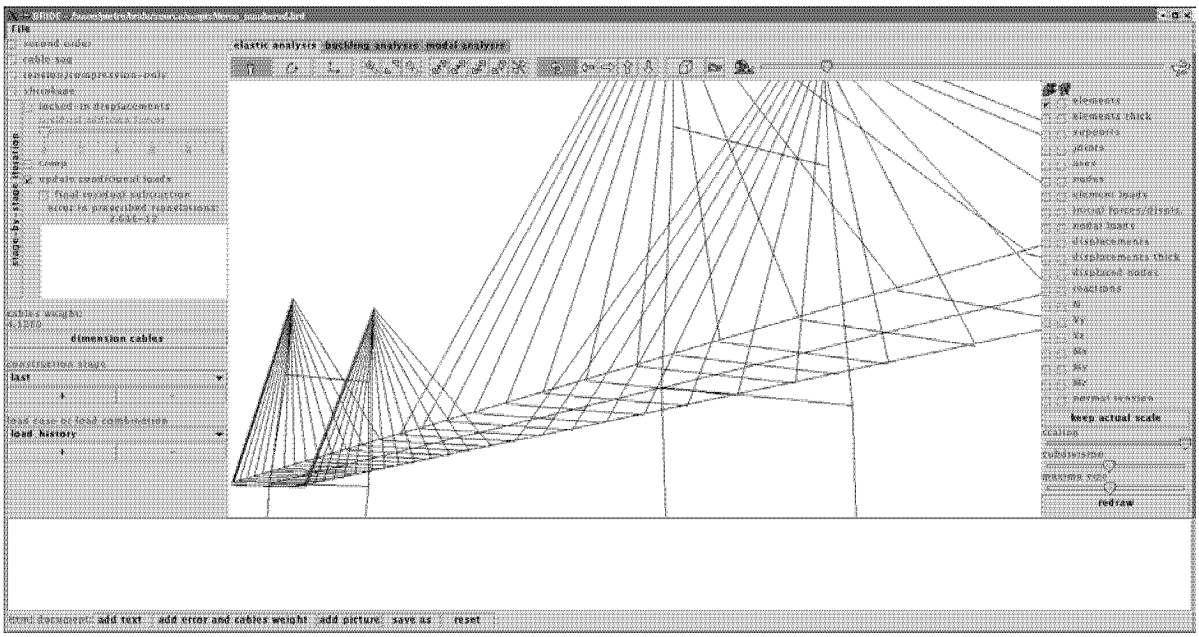


Figure A.2.: A snapshot of the program BRIDE showing the elements of a cable-stayed bridge model.

only searches among the sections belonging to the same group as the section of the cable being substituted. The local y- and z-maxima and minima are needed for the volumetric element representation and to find the normal stress distribution along the local z-axis.

A.1.7. Materials

As many sections usually have the same material, the material definition was separated from the section definition. The syntax to define a material is:

```
—Material material_identifier E E SigMax sigma_max G G Rho Rho
 MaxCreepFactor max_creep_factor DaysForHalfCreep days_for_half_creep
 MaxShrinkageStrain max_shrinkage_strain DaysForHalfShrinkage days_for_half_shrinkage ;
```

E is the Young's modulus, *SigMax* is the value of the maximum tension which is used for cable re-dimensioning, G is the shear modulus used to determine the torsional stiffness, *Rho* is the density (mass per unit volume). The development of both creep and shrinkage over time, being approximated by a hyperbolic tangent, have to be specified with two values each: the maximum asymptotic value for $t = \infty$ and the number of days needed to reach half the maximum asymptotic value (see the chapters 5 and 6).

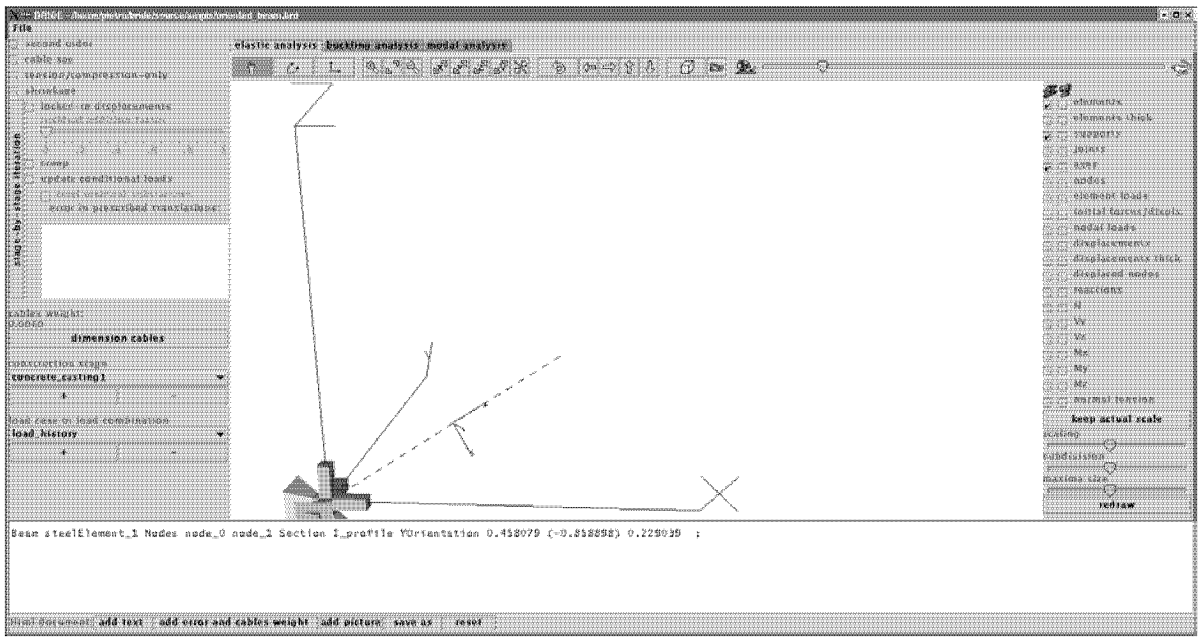
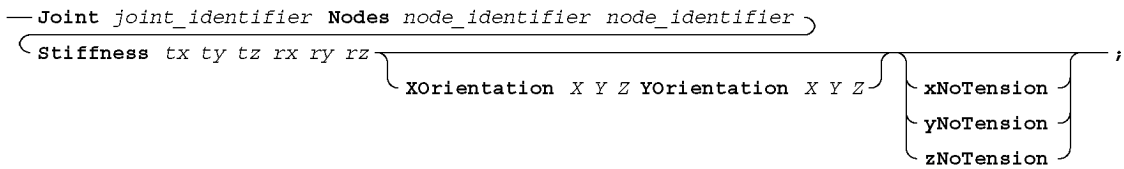


Figure A.3.: A snapshot of the program BRIDE showing an element modelling a cantilever (represented by a dashed line) with its support, its local coordinates system and the global coordinates system.

A.1.8. Joints

Joints are zero-length elements used to connect two nodes which are in the same location (element incidence nodes in the same location are not allowed because they would lead to an element of zero length and infinite stiffness).

The syntax for the definition of a joint is:



- The two node identifiers define the joint's incidences.
- *tx*, *ty*, and *tz* are the stiffnesses of the springs connecting the translational degrees of freedom of the two nodes, expressed in local coordinates.
- *rx*, *ry*, and *rz* are the stiffnesses of the springs connecting the rotational degrees of freedom of the two nodes, expressed in local coordinates.
- An orientation of the local coordinates system differing from the global one has to be defined explicitly. The three numeric values after the keyword *YOrientation*

give the direction of a vector co-planar to the local x axis, defined immediately beforehand, and the local x-y plane.

- The keywords *xNoTension*, *yNoTension*, and *zNoTension* allow one to define a contact connection in one of the three local directions, in which the nodes can separate freely. This represents another non-linear effect the program can deal with.

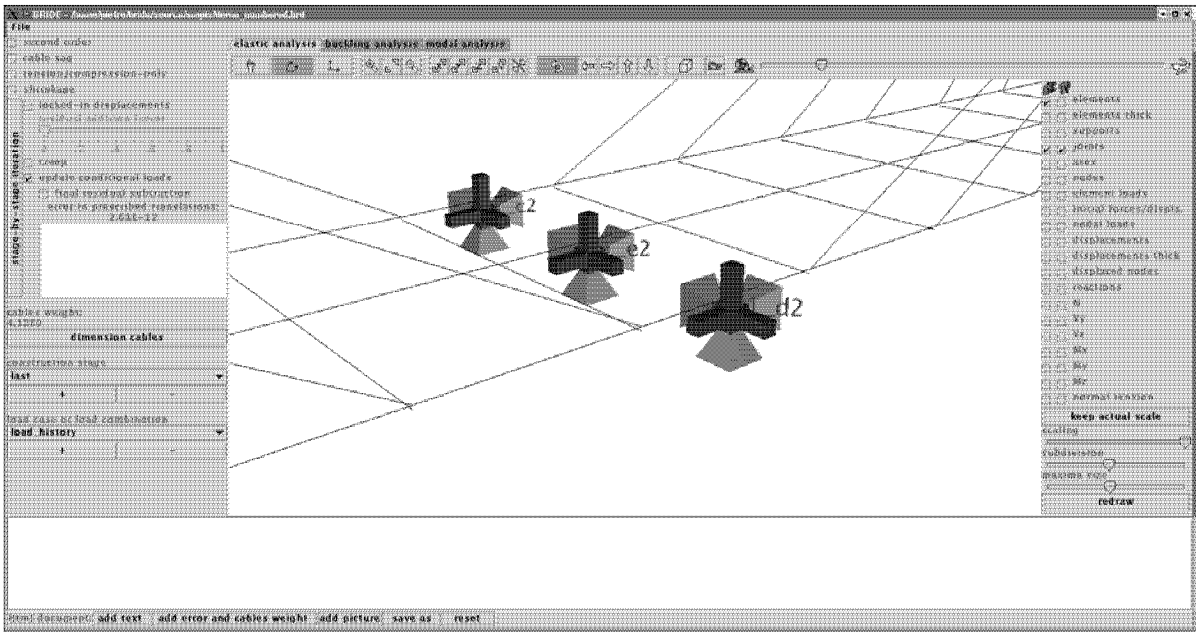
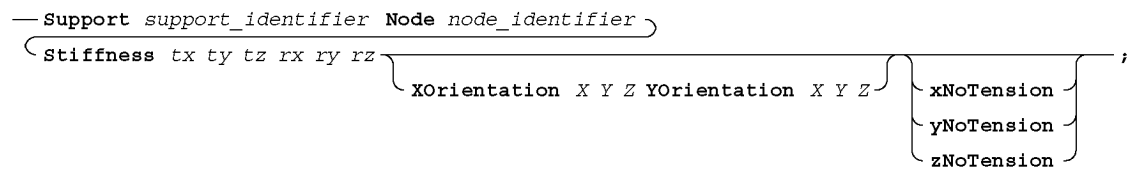


Figure A.4.: A snapshot of the program BRIDE showing the joints of a cable-stayed bridge model with their identifiers and the elements in a wire-frame representation.

A.1.9. Supports

Supports are similar to joints but with a single incidence node: they represent a connection between this single incidence node and the ground. Supports are necessary in a static model because they anchor it in space making it stable.

The syntax for the definition of a support is:



- The single node identifier defines the support's single incidence node.
- *tx*, *ty*, and *tz* are the stiffnesses of the springs connecting the translational degrees of freedom of the single incidence node to the ground, expressed in local coordinates.
- *rx*, *ry*, and *rz* are the stiffnesses of the springs connecting the rotational degrees of freedom of the single incidence node to the ground, expressed in local coordinates.
- An orientation of the local coordinates system differing from the global one has to be defined explicitly. The three numeric values after the keyword *YOrientation* give the direction of a vector co-planar to the local x axis, defined immediately beforehand, and the local x-y plane.
- The keywords *xNoTension*, *yNoTension*, and *zNoTension* allow one to define a contact support in one of the three local directions in which the node can lift off from the ground. This represents one of the non-linear effects the program BRIDE can deal with.

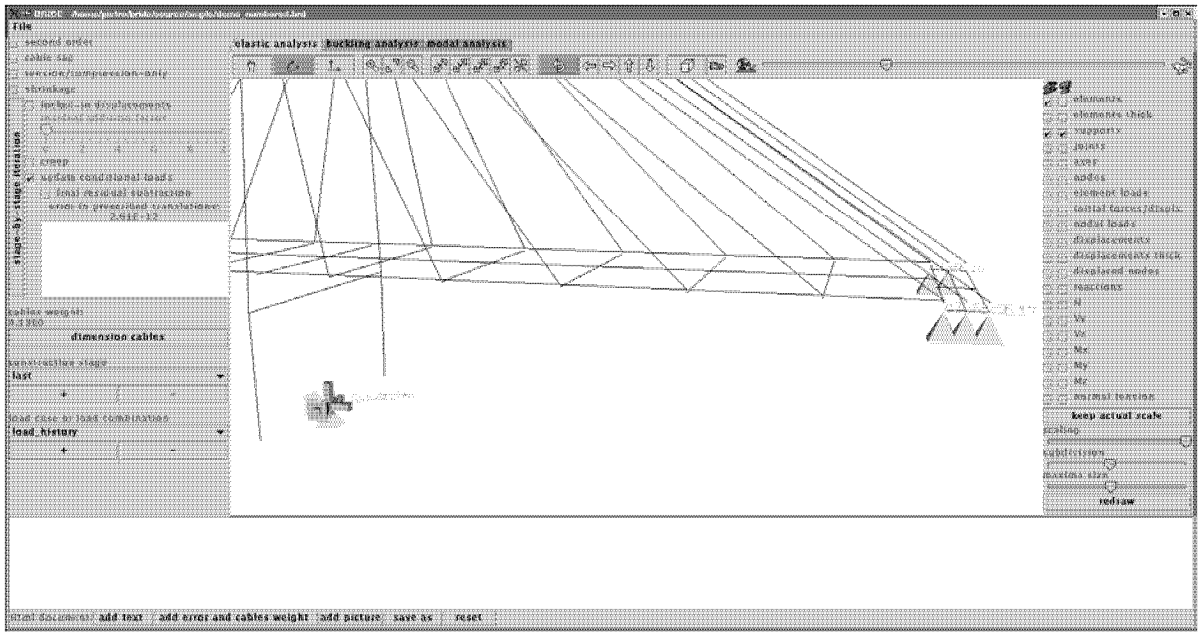


Figure A.5.: A snapshot of the program BRIDE showing the supports of a cable-stayed bridge model with their identifiers and the elements in a wire-frame representation.

A.1.10. Load cases and load combinations

In structural engineering it is usual to investigate all possible hazards which could involve a structure and to simulate them with sets of forces acting on the structure called *load*

cases. Loading forces are defined as any other object. A loading force can affect just one element or node. To attribute load forces to one or more load cases, load case identifiers have to be appended at the end of the object (see subsections A.1.12 and A.1.13).

Load cases can be multiplied by different factors and superposed according with the rules of the national codes in force. Those load case superpositions are called *load combinations*.

This is the syntax for the definition of load combinations:

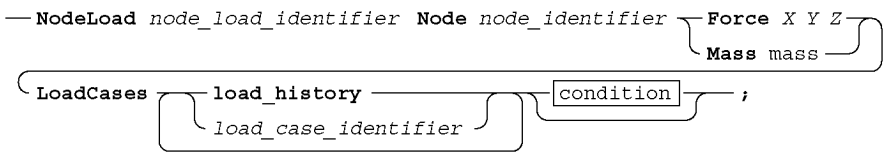
Any number of load cases given in the load combination are multiplied by the adjacent load factor and superposed.

A.1.11. The load_history load case

While the initial self-equilibrating forces **f** simulating locked-in displacements, creep and shrinkage are automatically applied to the model during stage-by-stage iteration, other user defined loads such as the self-weight or conditional loads (see chapter 7) are not. To state that a user defined load should be taken into account during the stage-by-stage iteration it has to be assigned to the *load_history* load case (see subsections A.1.12 and A.1.13).

A.1.12. Node loads

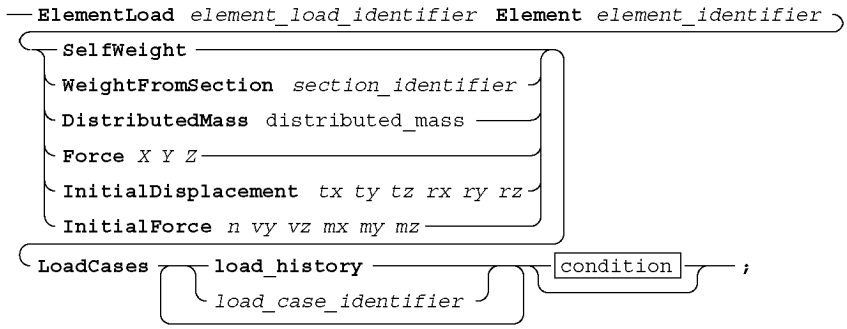
The syntax for a node load instruction is:



- The node identifier defines the node in which the force acts.
- The three numeric values after the keyword *Force* define the vector of a force acting on the node.
- The numeric value *mass* defines a concentrated mass attached to the node which is used for the modal analysis, the static analysis, and the buckling analysis: when a static or a buckling analysis is performed the concentrated mass is automatically multiplied by the acceleration due to gravity of $9.81\frac{m}{s^2}$ resulting in a load in the negative Z-direction. Rotational nodal masses cannot be taken into account.
- A conditional node load can be defined by adding a condition (see subsection A.1.14). If a condition is defined the node load has also to be assigned to the load case *load_history*.

A.1.13. Element loads

The syntax for the plane element load instruction is:



- The element identifier defines the element on which the force acts.
- The element forces defined by selecting the keywords *SelfWeight*, *WeightFromSection*, and *DistributedMass*, are defined by a distributed mass which is used for modal analysis and for defining the weight for static analysis and buckling analysis.
- The distributed mass is automatically multiplied by the acceleration due to gravity acting in the -Z direction when a static or a buckling analysis is performed.
- The self-weight element force does not need any further specification because element section and material are known.
- The weight-from-section element force is the same as the self-weight element force but the section used is not the element's own but another one specified with the identifier. This has been introduced to define the weight of liquid concrete on the formwork element of a cast or composite beam to take locked-in displacements into account.
- The numeric value *distributed_mass* defines the value of the uniformly distributed mass over the length of the element.
- The three-dimensional vector after the keyword *Force* defines the uniformly distributed force's vector (force components per unit length) acting on the element's axis.
- The six values after the keyword *InitialDisplacement* are, respectively, the three translation and the three rotation components of the initial displacement of the second incidence node in the element's local coordinates system, if it were free to move in any direction and if the first incidence node were clamped.
- With the keyword *InitialForce* an initial self-equilibrating group of forces acting in the element's incidence nodes can be defined. The six numeric values after the

keyword *InitialForce* are, respectively, the normal force, the shear forces in local y and z directions, the torsion and the bending moments in local y and z directions acting on the first incidence node. The normal forces and moments acting on the second incidence node are the same as the specified ones but with opposite sign. If shear forces are defined (i.e. they are not equal to zero) the related bending moments acting on the second incidence node are changed according with the well known equilibrium relations between bending moments and shear forces.

- The element forces defined by selecting the keywords *Force, InitialForce* and *InitialDisplacement* are only used for static and buckling analysis.

- A conditional element load can be defined by adding a condition (see subsection A.1.14). If a condition is defined the element load has to be assigned to the load case *load_history*.

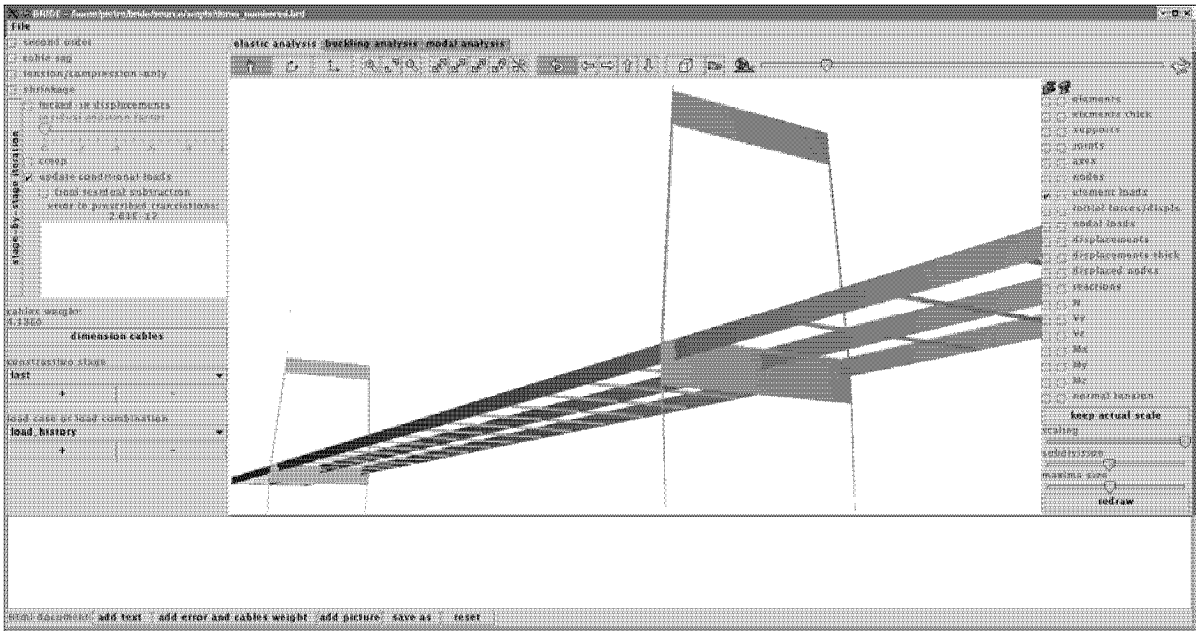
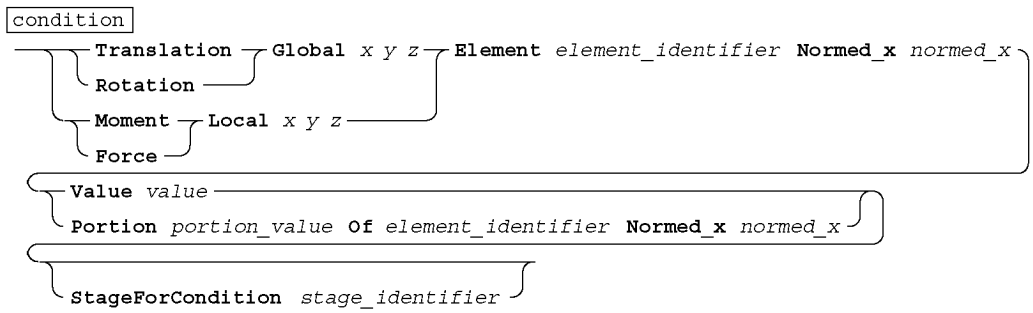


Figure A.6.: A snapshot of the program BRIDE showing element loads acting on a cable-stayed bridge model.

A.1.14. Conditions

The syntax for conditions prescribing a displacement or a force is:



- The arbitrary intensity load is defined in the load object the condition has been assigned to and the activation stage is implicitly defined by the position of the load object in the chronological objects list. The fixed load is defined by all load objects belonging to the load case *load_history* active in the condition stage.

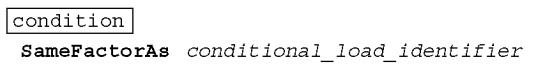
- The first line of the rail-road diagram allows one to define the condition parameter which can be a translation, a rotation, a combination of inner bending moments or a combination of section forces. The element identifier and a normed value for the x coordinate (i.e. going from 0 at the element's beginning to 1 at the element's end) allow one to define the location of the condition parameter.

- The second line allows one to define the condition value which can be expressed absolutely or relatively (as a portion of the value assumed by the same kind of parameter in another point of the structure). If the geometry of the model defined in the program input reflects the service geometry of the cable-stayed bridge, the values for an absolute displacement condition are in most cases set to 0.

- The condition stage may be the same as the activation stage or a subsequent one. If it is a subsequent one its identifier has to be given after the keyword *StageForCondition* otherwise it is assumed to be the same as the activation stage.

- Conditional factors are computed during the stage-by-stage iteration, hence the conditional loads must be attributed to the standard load case *load_history* and their conditions are also fulfilled only for this load case.

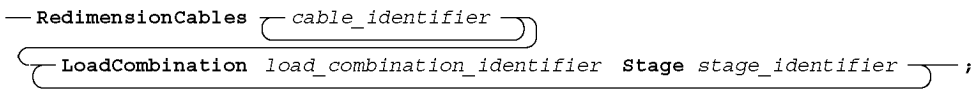
The syntax used to define a "same factor as" condition:



The condition is defined by specifying the identifier of the conditional load which must have the same conditional factor, the condition stage is retrieved from that conditional load.

A.1.15. Cable dimensioning object

The syntax to define a cable dimensioning object is:



The object consists of a list of cables to be dimensioned and a list of stage/load-combination pairs which the cables have to resist.

A.1.16. Notes

Notes can be inserted at any point of the objects list using this syntax:

```
Note note text ;
```

such notes -unlikely to preprocessor comments (see subsection A.2.7)- are stored and appear in the input files stored by the program.

A.2. Text-preprocessor

A powerful text-preprocessor developed by Dr. Guido Sartoris has been integrated in the syntax checking system. This allows the use - in the input file - of parameters, for-loops, if-tests, mathematical expressions, and standard mathematical functions.

The text-preprocessor has its own keywords and constructs for text manipulations which are disseminated all along the input file. When they are found by BRIDE's syntax checking system, the preprocessor evaluates them on-line and substitutes them with the result of their evaluation.

The text-preprocessor features combined with the loose sequence of input objects allow a powerful input generation, e.g. objects can be generated in a for-loop and the loop's control variable can be used for both the definition of the numeric values through mathematical expressions and the definition of the object identifier which may contain a preprocessor variable value in its string.

A.2.1. Variables

The preprocessor variables are similar to the variables of a programming language[3] and can be used in innumerable ways, e.g. as parameters in expressions or as a control variable in loops for the generation of coordinates and identifiers. Their name gives a

[3] the main differences are that the preprocessor variables have just a floating point number type, that after being declared they remain visible to the end of the input and that even if they are declared more than once there is always only one instance associated to a variable name.

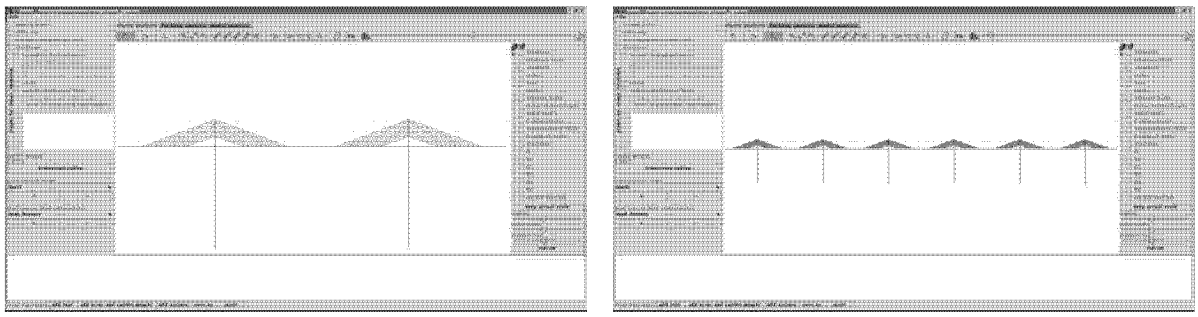


Figure A.7.: Two different models both obtained with the same input file by setting its variable "number_of_masts" first to 2 (left) and then to 6 (right).

meaning to the value they store and, depending on how the input file has been written, changing their initialization can change the whole input file (see figure A.7).

Variables can be declared and defined in two ways, either with the following object-like definition:

```
Define variable_name = variable_value ;
```

or by using and initializing the new variable directly where it is needed with a "=", as in the following example where the variable *Susanna* is declared, initialized with the value 4 and used to express the X coordinate:

```
...
Node mast_1 Susanna = 4 0 0 ;
Node mast_2 Susanna 0 3 ;
...
```

The former objects would be transformed as follows by the preprocessor:

```
...
Node mast_1 4 0 0 ;
Node mast_2 4 0 3 ;
...
```

Variable names (e.g. *Susanna*) can be constituted by any sequence of letters, digits and the underscore character beginning with a letter.

A.2.2. Expressions

Expressions can be used at any place of the input file in which a numeric value is required. The syntax is the standard expression syntax also used in programs such as spreadsheet programs. Parentheses can be used to force evaluation precedences. Expressions may not begin with a minus (in such a case the whole expression has to be put in parentheses) and can contain variables which can be defined and initialized previously or in the expression itself, e.g. the example in the former paragraph (there the expressions are constituted by just one variable delivering the x coordinate). The operators allowed in expressions

Operator	Associativity	Description
?:	Right To Left	Conditional
||	Left to Right	Logical OR
&&	Left to Right	Logical AND
!=	Left to Right	Inequality
==	Left to Right	Equality
<	Left to Right	Less than
<=	Left to Right	Less than or equal to
>	Left to Right	Greater than
>=	Left to Right	Greater than or equal to
+	Left to Right	Addition
-	Left to Right	Subtraction
*	Left to Right	Multiplication
/	Left to Right	Division
%	Left to Right	Modulo Division

Table A.1.: The operator's precedence in increasing order. Operators with the same precedence are grouped together.

are summarized in the table A.1.

The preprocessor also allows the use of the following standard mathematical functions needing one parameter:
`exp(...)`, `log(...)`, `sin(...)`, `asin(...)`, `cos(...)`, `acos(...)`, `tan(...)`, `atan(...)`, `abs(...)`, `int(...)`, `sqrt(...)`
or two parameters:
`min(...,...)`, `max(...,...)`, `pow(...,...)`, `atan2(...,...)`.
The definitions of all these functions are the same as in the programming language C.

A.2.3. [expression] integer evaluation

Brackets are used for integer evaluations of expressions, where the rounded result of the evaluation replaces the brackets. This statement is very useful to generate identifiers.

A.2.4. `For`-loop

The `For`-loop makes BRIDE's syntax checking system re-read a given text portion a given number of times. This is its syntax:

For control_variable **From** initial_value **To** final_value **{**
text portion read repeatedly
}

The first time the syntax checking system reads the text portion in curly brackets the control variable assumes the initial value. By any further reading the value of the control variable is increased by one (this means that the initial value has to be smaller

than the final one). When the control variable assumes the final value the text portion is read for the last time.

This is an example showing the statement used to generate some identifiers and coordinates:

```
...
For i From 0 To 3 {
  Node mast_[i] 4 0 3*i ;
}
...
```

The former instructions would be expanded as follows by the preprocessor:

```
...
  Node mast_0 4 0 0 ;
  Node mast_1 4 0 3 ;
  Node mast_2 4 0 6 ;
  Node mast_3 4 0 9 ;
...
```

A.2.5. `If...Else...` construct

This construct allows one to switch on and off portions of text inside the input file depending on the evaluation of the expression in parentheses: if the expression doesn't equal zero or if it is a *true* logical expression the first block of text in curly brackets is read otherwise the second one. This is the construct's syntax:

If (*expression*) **{**
text read if expression evaluation≠0 or true
} Else {
text read if expression evaluation=0 or false
}

The statement can also finish right before the `Else`, if no alternative is needed. Here is an example used in combination with a `For`-loop:

```
...
For i From 0 To 3 {
  If ( i == 0) { Node deck_0 4 0 3*i ; }
  Else { Node mast_[i] 4 0 3*i ; }
}
...
```

The former instructions would be expanded as follows by the preprocessor:

```
...
  Node deck_0 4 0 0 ;
  Node mast_1 4 0 3 ;
```

```
  Node mast_2 4 0 6 ;
  Node mast_3 4 0 9 ;
...
```

A.2.6. `Include` statement

When the preprocessor encounters an `Include` statement it replaces it with the content of the included file. In this example a file named "my_bride_libraries/sections.brd" has been included in the input file:

```
...
Include my_bride_libraries/sections.brd
...
```

A.2.7. Comments

Portions of the input file can be commented out with a `(*` at the beginning and a `*)` at the end of the commentary.

A.2.8. Finish

The very last word of the outermost input file (the file not included by any other) has to be the keyword *Finish*.

B. Graphic user interface of the program BRIDE

The program BRIDE has been conceived as a simulation program for engineers who know how such simulations are performed. Therefore the goal was set for the design of the graphic user interface (GUI) of reducing as much as possible the distance between the user and the inner workings of the program. This has been attempted by choosing a combination of GUI components which reproduce the inner structure of the program and its state (see figure B.1 and the program's snapshots in appendix A).

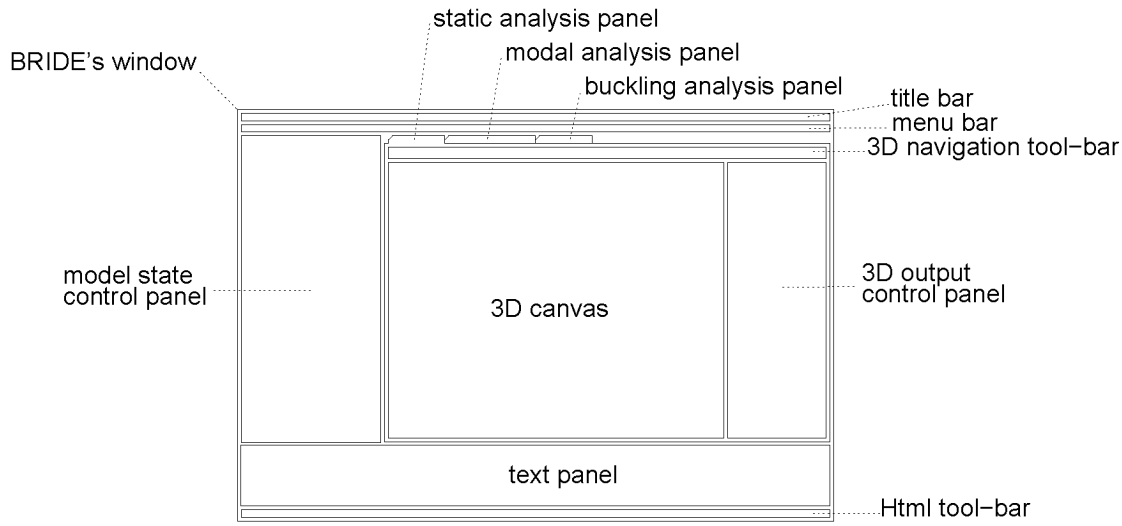


Figure B.1.: GUI structure of the program BRIDE.

The program BRIDE appears in a window when it is started. At the very top of this window there is a title bar in which the name of the program and the path of the opened file appear. Inside the window below the title bar there is the menu bar containing the "file" menu with which the input file can be opened and closed and the input file automatically generated by the program can be saved. Below the menu bar there are the two main GUI components: the "model state control panel" on the left (with which the stage-by-stage iteration and the cable re-dimensioning can be started and the model options, the stage and the load case can be set) and the tabbed panel with the three analysis panels on the right. Each analysis panel contains a "3D canvas", in which the model and the analysis's results are represented three-dimensionally, and a "3D output

control panel" in which the user can choose what should be represented in the 3D canvas by selecting the appropriate check boxes. At the bottom there is a "text panel" in which there appears a formatted text describing the three-dimensional graphic object selected by the user with the mouse. At the very bottom of the window there is the "Html tool-bar" with which the user can save the computation's results in an Html document containing snapshots of the 3D canvas and of the text panel.

B.1. Definition of some standard GUI components

In this section some standard GUI components are defined. The use of standard GUI components is very important for user-friendliness because it allows the user to operate the GUI instinctively based on his experience with other programs.

Button: region of the GUI firing an event if clicked with the mouse mostly delimited by a border and containing either an icon or a string.

Toggle-button: the same as a button except that when it is selected it remains selected and needs to be clicked again to be deselected. In toggle-buttons selection is expressed through a darker rendering of the button region.

Check-box: the same as a toggle-button except that the selection state is showed by a check-mark instead of a darker rendering of the button region.

Combo-box: a region of the GUI showing one item at a time selected by the user out of a list of items which appear only if the mouse clicks once on the combo-box region.

Menu: button opening a list of items to be selected when clicked. Menus appear usually in a bar on the top of the program's window and carry typically the same names in all programs (e.g. "file", "edit", "help",...) thus allowing the user to guess which kind of program functionality they contain.

Bar, tool-bar: stripe-shaped region of the GUI containing more components with related functions.

Panel: rectangular region of the GUI containing one or more components with related functions.

Tabbed-panel: GUI component allowing one to show only one panel at a time out of a group of panels. The presence of the other panels is expressed through selectable tabs on the side of the panel being shown.

3D canvas: special panel allowing the rendering of three-dimensional models.

B.2. Menu bar

The menu bar (see figure B.2) contains the file menu which allows one to open a file, re-open it keeping the current output options (if the input file has been slightly modified), save an input file reflecting the content of the actual loaded model (i.e. considering the possibly changed cable sections and the computed conditional loads), close the model and exit the program.

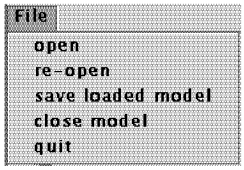


Figure B.2.: The file menu.

B.3. Model state control panel

The model state control panel (see figure B.3) contains the GUI components needed for the following computation options: second order, cable sag, tension/compression only (in elements such as cables, supports and joints) and shrinkage.

In the middle on the left side a vertical toggle-button is used to start and stop the repeated execution of the stage-by-stage iteration. On its right are the check-boxes for the option related only to such an iteration: locked-in displacements, creep, conditional loads, the slider to set the error feed-back factor α explained in the subsection 7.3.2 and the check-box to set $\alpha = -1$ for the last stage-by-stage iteration as explained in section 7.3.3. The chart in the white rectangle represents the error in the fulfilment of the conditions of conditional loads prescribing a given absolute translation and is used to show the user if convergence is taking place during the stage-by-stage iteration and to help him operate the slider for the feed-back factor α. A label right above the chart gives the error in the last iteration and the darker lines in the chart's background represent a power of 10.

The button carrying the label "dimension cables" starts the cable dimensioning. The user can read the weight of all cables in the label right above the button (in tons if the section areas of the cables have been expressed in square metres and the material density in tons per cubic metre).

The two combo-boxes on the lower end contain the identifiers of all construction stages (the first combo-box) and of all load cases and load combinations (the second one) defined by the user in the input file. The user can select whatever construction stage/load pair is desired to view in the analysis panels. The buttons carrying the labels "+" and "-" allow one to select the combo-box item coming before or after the one being selected and are an alternative in addition to the standard combo-box selection mode.

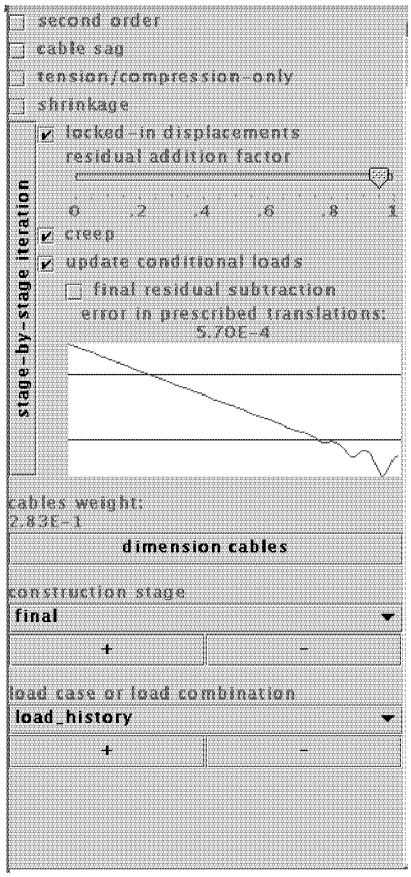


Figure B.3.: The model state control panel.

B.4. Elements common to all three analysis panels

Figure B.4.: Tabs of the tabbed-panel.

In the program BRIDE a specific "analysis panel" with a 3D canvas, a 3D navigation tool bar and a 3D output control panel has been dedicated to each kind of analysis the program BRIDE can perform: static, buckling and modal analysis. These three panels have been stacked by means of a tabbed-panel GUI component (see figure B.4). When the user selects a tab with the mouse, the corresponding analysis is performed or, respectively, the corresponding panel appears on the top of the panel stack.

The 3D canvas looks the same in the three analysis panels: a rectangle in which three-dimensional objects are rendered. Snapshots of the picture rendered in the 3D canvas can be included in the Html document selecting a button of the Html tool bar.

The 3D navigation tool bar is common to the three analysis panels but is quite complex in its construction and merits a closer examination (see figure B.5).

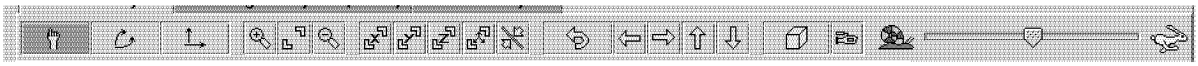


Figure B.5.: 3D navigation tool-bar.

The navigation in the three-dimensional world represented inside the 3D canvas is defined through two points: the "eye" point in which the eye of the viewer is located and the "center" point to which the eye is looking at. The first three buttons on the left can be selected only one at a time and define the mouse behaviour inside the 3D canvas. If the first button is selected, by clicking any three-dimensional object a formatted text describing the clicked object (which can be either a model object or the representation of a result) appears in the text panel. If the second button is selected the viewing direction of the user in the three-dimensional world represented in the 3D canvas can be changed by clicking on the three-dimensional representation: the point of the representation on which it was clicked becomes the new center of the picture. If the third button is selected by clicking any three-dimensional object the center is set on the clicked point but the view direction is not changed, i.e. the eye is also moved collaterally. The fourth button allows one to zoom in, the fifth to center the image and the sixth to zoom out. The seventh, eight and ninth allow one to choose a viewing direction parallel to a main axis and to center the image. The tenth allows one to choose the viewing direction [-1,-1,-1] and to center the image. The eleventh allows one to view the three-dimensional representation from the other side. The twelfth button is a toggle-button which allows one to define the eye moving behaviour of the subsequent four arrow buttons: if it is selected the eye is moved around the center of the image changing the viewing direction, if it is not selected the eye movement is translatory. The last but one button is a toggle button allowing one to define if the three-dimensional representation should be iso-axial or perspective. The last button makes a pop-up menu pop-up in which the user can define the coordinates of the eye and of the center manually. The slider allows one to change the speed of the eye: if the slider is moved toward the rabbit the eye becomes faster and if the slider is moved toward the snail the eye becomes slower.

The 3D output definition panels in the three analysis panels also have common elements: in all 3D output definition panels the upper side is organized in three columns, the two first columns contain check boxes and the third column labels (see figure B.6). The first column, denoted by an icon representing an I-profile, contains the check boxes which allow one to require the representation of the objects specified in the label on the same row. The second column, denoted by an icon representing a "T", contains the check-boxes for the representation of text elements such as identifiers (for model objects) and maxima (for the results) relative to the objects specified in the label. On the bottom of the three columns there are three sliders, the first allowing one to adjust the scale of some three-dimensional objects not having a constraining physical dimension (nodes, joints and supports), the second one is used to set the number of points in the element

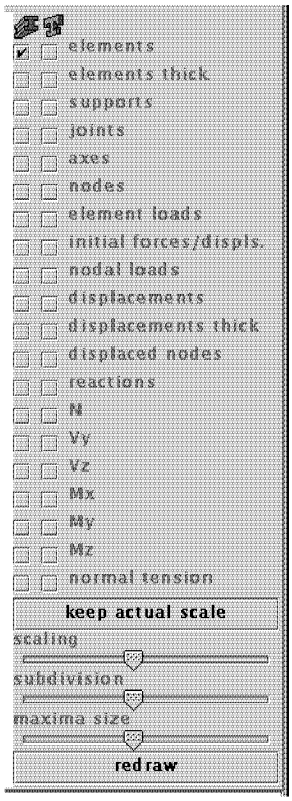


Figure B.6.: The 3D output definition panel in the static analysis panel.

in which the results have to be interpolated during post-processing and the third slider allows one to set the size of the maxima to which the results have to be scaled.

B.5. Specific analysis panel components of the 3D output definition panels

B.5.1. Static analysis panel

By selecting the "elements" check-box (see figure B.6) the elements are represented as wire-frames, by selecting the "elements thick" check-box they are represented as boxes having the width and the height specified in the section object with the maxima and the minima in the local y and z directions. The difference between "displacements" and "displacements thick" is the same as between "elements" and "elements thick": selecting the "displacements" check-box the displaced shape of the structure is shown as a wire-frame spline where the elements in traction are coloured in red and those in compression in blue, while selecting the "displacements thick" check-box the spline is represented by boxes between the interpolation points having the same cross section as the boxes used for the thick elements. If the "normal tension" check-box is selected the same appears

as with "elements thick", but if such an element is selected, a line of charts representing the normal tension in its interpolation points along the local z axes appears in the text panel. If the toggle-button "keep actual scale" is selected the scale for the amplification of the results representation is kept unchanged; this is very useful for comparing results.

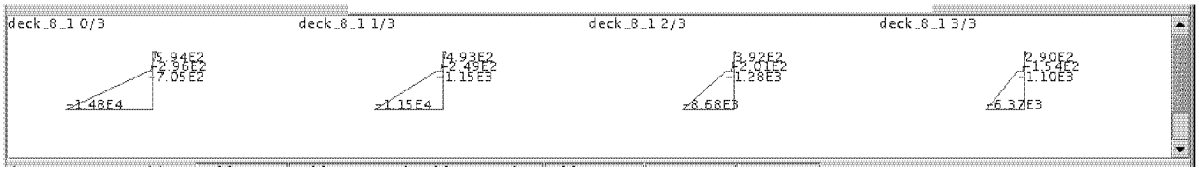


Figure B.7.: Text panel with the charts of the normal tension along the local z axis in the interpolation points of a composite element carrying the identifier "deck_8_1".

B.5.2. Buckling analysis panel

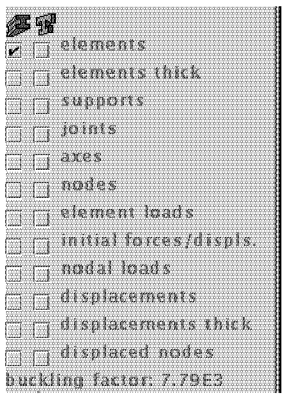


Figure B.8.: Components of the 3D output specification panel in the buckling analysis panel.

The only characteristic of this panel (see figure B.8) is the label showing the buckling factor. The displacements represent the buckling shape and are normed, i.e. their magnitude has no physical meaning.

B.5.3. Modal analysis panel

The only not self-explanatory components in this panel (see figure B.9) are the buttons with the label "+" and "-" to choose the natural oscillation and the label with the resonance frequency in Hertz.

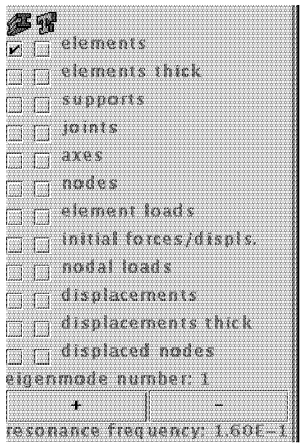


Figure B.9.: Components of the 3D output specification panel in the modal analysis panel.

Figure B.10.: The Html tool-bar.

B.6. Text panel

In the text panel there appear a formatted text description of the three-dimensional graphic objects selected with the mouse and the charts of the normal tensions (see figure B.7). Snapshots of the text panel content can be added to the Html document using a button of the Html tool bar. At the beginning of the Html document an index constituted by links leading to the different snapshots is inserted.

B.7. Html tool-bar

The Html tool-bar (see figure B.10) allows the user to generate an Html[1] document constituted by snapshots of the 3D canvas and of the text panel and to store it. With the first button on the left it is possible to add to the Html document a snapshot of the text panel, the error and the cable weight with the second button, a snapshot of the 3D canvas with the third, with the fourth button the generated document can be saved and with the last button it can be reset.

B.8. Syntax highlighting

Syntax highlighting is a very useful feature not directly implemented in the program BRIDE but in most text editors used by programmers. To custom an editor for a

[1]The Hyper Text Markup Language is the format of most Internet pages and can be read not only by all web browsers, but also by the main word processors.

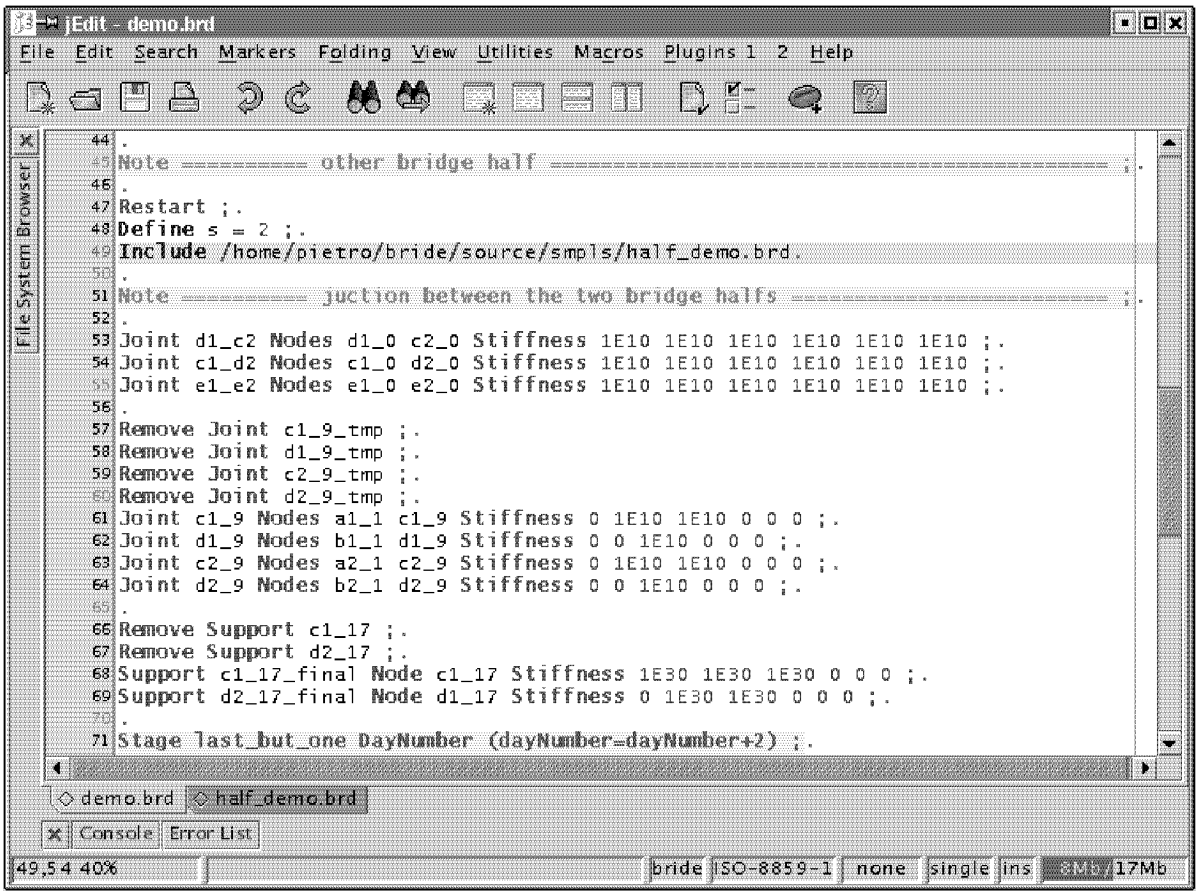


Figure B.11.: Syntax highlighting with the editor Jedit.

special syntax, such as the input syntax of the program BRIDE, it is usually necessary to enter the syntax keywords in some editor's configuration file. Figure B.11 shows a snapshot of the editor Jedit (downloadable for free at www.jedit.org) highlighting the syntax of an input file for the program BRIDE.

C. Information technology tools used to develop the program BRIDE

The program BRIDE has been written using both programming languages C++ and Java and can be compiled without any modification on the operative systems Linux and Windows. Mac OS has not been tested but the code should be fully compatible with that operating system too.

In the program BRIDE the reading of the input file and all computations are performed by a shared object (or a "dynamic link library" in the Windows jargon) written in C++ (see [Stroustrup]) and compiled with the compiler from the GNU foundation (see www.gnu.org). While the operating system Linux is the natural one for the GNU compiler, for the Windows operating system the GNU compiler embedded in the Minimal GNU Windows environment (see www.mingw.org) has been used.

The graphic user interface (GUI) has been written in Java using the GUI components of the Swing package (see [Java Tutorial]). The three-dimensional renderings have been implemented with the Java 3D library (see [Java 3D]), which is a Java interface to the 3D native libraries Open GL and DirectX. The GUI has been compiled using the Java compiler by Sun Microsystems (www.java.sun.com). When the GUI is started, it loads the shared object and (according to the wishes of the user) begins sending him queries and receiving back the information needed to represent the results. The communication between the GUI and the C++ has been implemented using the Java Native Interface (JNI, see [JNI]).

All compilers and libraries used can be downloaded for free on the Internet.

During the development of the program BRIDE it was very important to stick to the object-oriented programming paradigm to keep the increasing complexity of the source code under control; a few books on the subject are listed in the bibliography (see [FAQs, Refactoring, Patterns]).

D. Input files discussed in chapter 9

D.1. Main input file

```
(*  1*) Note ========== materials and sections ====================================== ;
(*  2*)
(*  3*) Material concrete_45_35 E 30E6 SigMax 0 G 11E6 Rho 2.5
(*  4*)   MaxCreepFactor 2 DaysForHalfCreep 40
(*  5*)   MaxShrinkageStrain 0.0005 DaysForHalfShrinkage 300 ;
(*  6*) Section 3_3 SectionGroup not_a_cable A 6.60 It 1 Iy 24.89 Iz  8.38
(*  7*)   yMax 1.5 yMin (-1.5) zMax 2.75 zMin (-2.75) Material concrete_45_35 ;
(*  8*) Section ab_1  SectionGroup not_a_cable A  5.1 It 1 Iy  2.53 Iz 13.38
(*  9*)   yMax 1 yMin (-1) zMax 2.5 zMin (-2.5) Material concrete_45_35 ;
(* 10*) Section 2_2 SectionGroup not_a_cable A 5.72 It 1 Iy 14.09 Iz  6.88
(* 11*)   yMax 1.5 yMin (-1.5) zMax 2.2 zMin (-2.2) Material concrete_45_35 ;
(* 12*) Section 1_1 SectionGroup not_a_cable A 4.52 It 1 Iy  7.90 Iz  2.03
(* 13*)   yMax 1 yMin (-1) zMax 1.9 zMin (-1.9) Material concrete_45_35 ;
(* 14*) Section cd_9 SectionGroup not_a_cable A .96 It 1 Iy .2 Iz .03
(* 15*)   yMax .8 yMin (-.8) zMax .3 zMin (-.3) Material concrete_45_35 ;
(* 16*) Section slab_long SectionGroup notACable A 2.38 It .05 Iy .012 Iz 18.032
(* 17*)   yMax 4.765 yMin (-4.765) zMax 0.125 zMin (-0.125) Material concrete_45_35 ;
(* 18*) Section slab_long_end SectionGroup notACable A 20.5 It 27 Iy 7.89 Iz 155
(* 19*)   yMax 4.765 yMin (-4.765) zMax 1.075 zMin (-1.075) Material concrete_45_35 ;
(* 20*) Section slab_cross SectionGroup notACable A 1.125 It .023 Iy .006 Iz 1.898
(* 21*)   yMax 2.75 yMin (-2.75) zMax 0.125 zMin (-0.125) Material concrete_45_35 ;
(* 22*) Section slab_cross_end SectionGroup notACable A 3.225 It 1.4 Iy 1.24 Iz .60
(* 23*)   yMax .75 yMin (-.75) zMax 1.075 zMin (-1.075) Material concrete_45_35 ;
(* 24*)
(* 25*) Material steel_360 E 195E6 SigMax 0 G 140E3*0.7 Rho 7.85
(* 26*)   MaxCreepFactor 0 DaysForHalfCreep 0
(* 27*)   MaxShrinkageStrain 0 DaysForHalfShrinkage 0 ;
(* 28*) Section I_profile SectionGroup notACable A 0.14 It 1E-4 Iy .067 Iz .001
(* 29*)   yMax .25 yMin (-.25) zMax .95 zMin (-.95) Material steel_360 ;
(* 30*)
(* 31*) Material 0.6_strands_steel E 195E6 SigMax 1770E3*0.7 G 0 Rho 7.85
(* 32*)   MaxCreepFactor 0 DaysForHalfCreep 0
(* 33*)   MaxShrinkageStrain 0 DaysForHalfShrinkage 0 ;
(* 34*) For i From 1 To 100 {
(* 35*)   Section [i]_strands SectionGroup cables A 150E-6*i It 0 Iy 0 Iz 0
(* 36*)     yMax dc = sqrt(150E-6*i/3.14) yMin (-dc) zMax dc zMin (-dc)
(* 37*)     Material 0.6_strands_steel ;
(* 38*) }
(* 39*)
(* 40*) Note ========== one bridge half ============================================== ;
(* 41*)
(* 42*) Define s = 1 ;
```

```
(* 43*) Include /home/pietro/bride/source/smpls/half_demo_numbered.brd
(* 44*)
(* 45*) Note ========== other bridge half ========================================== ;
(* 46*)
(* 47*) Restart ;
(* 48*) Define s = 2 ;
(* 49*) Include /home/pietro/bride/source/smpls/half_demo_numbered.brd
(* 50*)
(* 51*) Note ========== juction between the two bridge halfs ======================== ;
(* 52*)
(* 53*) Joint d1_c2 Nodes d1_0 c2_0 Stiffness 1E10 1E10 1E10 1E10 1E10 1E10 ;
(* 54*) Joint c1_d2 Nodes c1_0 d2_0 Stiffness 1E10 1E10 1E10 1E10 1E10 1E10 ;
(* 55*) Joint e1_e2 Nodes e1_0 e2_0 Stiffness 1E10 1E10 1E10 1E10 1E10 1E10 ;
(* 56*)
(* 57*) Remove Joint c1_9_tmp ;
(* 58*) Remove Joint d1_9_tmp ;
(* 59*) Remove Joint c2_9_tmp ;
(* 60*) Remove Joint d2_9_tmp ;
(* 61*) Joint c1_9 Nodes a1_1 c1_9 Stiffness 0 1E10 1E10 0 0 0 ;
(* 62*) Joint d1_9 Nodes b1_1 d1_9 Stiffness 0 0 1E10 0 0 0 ;
(* 63*) Joint c2_9 Nodes a2_1 c2_9 Stiffness 0 1E10 1E10 0 0 0 ;
(* 64*) Joint d2_9 Nodes b2_1 d2_9 Stiffness 0 0 1E10 0 0 0 ;
(* 65*)
(* 66*) Remove Support c1_17 ;
(* 67*) Remove Support d2_17 ;
(* 68*) Support c1_17_final Node c1_17 Stiffness 1E30 1E30 1E30 0 0 0 ;
(* 69*) Support d2_17_final Node d1_17 Stiffness 0 1E30 1E30 0 0 0 ;
(* 70*)
(* 71*) Stage last_but_one DayNumber (dayNumber=dayNumber+2) ;
(* 72*)
(* 73*) CompositeBeam c1_0_conc FormworkBeam c1_0 Dz 1.075 Section slab_long ;
(* 74*) ElementLoad c1_0_conc_l Element c1_0_conc Force 0 0 (-27.22) LoadCase live ;
(* 75*)
(* 76*) CompositeBeam d1_0_conc FormworkBeam d1_0 Dz 1.075 Section slab_long ;
(* 77*) ElementLoad d1_0_conc_l Element d1_0_conc Force 0 0 (-27.22) LoadCase live ;
(* 78*)
(* 79*) CompositeBeam e1_0_conc FormworkBeam e1_0 Dz 1.075 Section slab_long ;
(* 80*) ElementLoad e1_0_conc_l Element e1_0_conc Force 0 0 (-27.22) LoadCase live ;
(* 81*)
(* 82*) CompositeBeam c2_0_conc FormworkBeam c2_0 Dz 1.075 Section slab_long ;
(* 83*) ElementLoad c2_0_conc_l Element c2_0_conc Force 0 0 (-27.22) LoadCase live ;
(* 84*)
(* 85*) CompositeBeam d2_0_conc FormworkBeam d2_0 Dz 1.075 Section slab_long ;
(* 86*) ElementLoad d2_0_conc_l Element d2_0_conc Force 0 0 (-27.22) LoadCase live ;
(* 87*)
(* 88*) CompositeBeam e2_0_conc FormworkBeam e2_0 Dz 1.075 Section slab_long ;
(* 89*) ElementLoad e2_0_conc_l Element e2_0_conc Force 0 0 (-27.22) LoadCase live ;
(* 90*)
(* 91*) Stage last DayNumber (dayNumber=dayNumber+1) ;
(* 92*)
(* 93*) Note ========== cable dimensioning stuff ================================== ;
(* 94*)
```

```
(* 95*) LoadCombination history_and_live_1.7
(* 96*)   LoadCase load_history 1.7 LoadCase live 1.7 ;
(* 97*)
(* 98*) RedimensionCables
(* 99*)   For a From 1 To 2 {
(*100*)     For i From 0 To 7 {
(*101*)       c[a]_[8-i]_cab d[a]_[8-i]_cab c[a]_[10+i]_cab d[a]_[10+i]_cab
(*102*)     }
(*103*)   }
(*104*)   LoadCombination history_and_live_1.7 Stage last ;
(*105*)
(*106*) Finish
```

Line 1: a note describing what is coming next in the input file.

Lines 3-38: definition of materials and sections needed in the model.

Lines 34-38: definition of some cable sections not used directly in the model but needed to choose from during the cables dimensioning. The sections are generated using a preprocessor's for-loop.

Lines 42-43: definition of the first half of the bridge.

Line 42: the variable "s" used inside the file "half_demo_numbered.brd" to define the sign of the X and Y coordinates is set to 1.

Line 43: the file "half_demo_numbered.brd" listed in section D.2 is included using the pre-processor command "Include".

Lines 47-49: definition of the second half of the bridge.

Line 47: since the two halves of the bridge are erected simultaneously a restart object is needed before the definition of the second half of the bridge.

Line 48: the variable "s" used inside the file "half_demo_numbered.brd" to define the sign of the X and Y coordinates is set to 2.

Line 49: the file "half_demo_numbered.brd" listed in section D.2 is included using the pre-processor command "Include".

Lines 53-91: definition of the objects joining together the two halves of the bridge.

Lines 53-55: definition of the joints joining together the deck in the middle.

Lines 57-64: substitution of some temporary joints used during erection with less constraining ones for the final state.

Lines 57-60: removal of the temporary joints.

Lines 61-64: definition of the final joints.

Lines 66-69: substitution of some temporary supports used during erection with less constraining ones for the final state.

Lines 66-67: removal of the temporary supports.

Lines 68-69: definition of the final supports.

Line 71: stage object definition, the variable "dayNumber" is defined inside the included file.

Lines 73-89: definition of the composite beams cast over the steel profiles joined together in the middle of the mid-span.

Lines 95-96: definition of the load combination used to dimension the cables.

Lines 98-104: definition of a cable dimensioning object. The identifiers of the cables to be dimensioned are generated with a preprocessor's for-loop.

Line 106: "Finish" command needed at the end of every outermost input file (i.e. not included by any other input file).

D.2. Input file included (twice) in the main input file

```
(*  1*) Note ========== definition of some variables ================================ ;
(*  2*)
(*  3*) Define sf = (s-1)*2-1 ;
(*  4*) Define dayNumber = s/2-.5 ;
(*  5*) Define tmpx = 0 ;
(*  6*)
(*  7*) Note ========== mast ============================================================ ;
(*  8*)
(*  9*) Node s[s]_foundation 80*sf 0 0 ;
(* 10*) Support s[s]_foundation Node s[s]_foundation
(* 11*)   Stiffness 1E30 1E30 1E30 1E30 1E30 1E30 ;
(* 12*)
(* 13*) Node a[s]_1 80*sf (-9.53)*sf 13.25 ;
(* 14*) Beam a[s]_1 Nodes s[s]_foundation Excentricity 0 (-10.7)*sf 0
(* 15*)   a[s]_1 Excentricity 0 (-1.5*sf) (-2.35) Section 3_3 YOrientation 0 sf 0 ;
(* 16*) ElementLoad a[s]_1_s Element a[s]_1 SelfWeight LoadCase load_history ;
(* 17*) ElementLoad a[s]_1_p Element a[s]_1 InitialDisplacement .01 0 0 0 0 0
(* 18*)   LoadCase load_history Translation Global 0 0 1 Element a[s]_1
(* 19*)   Normed_x 1 Value 0 StageForCondition half[s]_last ;
(* 20*)
(* 21*) Node b[s]_1 80*sf 9.53*sf 13.25 ;
(* 22*) Beam b[s]_1 Nodes s[s]_foundation Excentricity 0 10.7*sf 0
(* 23*)   b[s]_1 Excentricity 0 1.5*sf (-2.35) Section 3_3 YOrientation 0 sf 0 ;
(* 24*) ElementLoad b[s]_1_s Element b[s]_1 SelfWeight LoadCase load_history ;
(* 25*) ElementLoad b[s]_1_p Element b[s]_1 InitialDisplacement .01 0 0 0 0 0
(* 26*)   LoadCase load_history Translation Global 0 0 1 Element b[s]_1 Normed_x 1
(* 27*)   Value 0 StageForCondition half[s]_last ;
```

```
(* 28*)
(* 29*) Beam ab[s]_1 Nodes a[s]_1 Excentricity 0 (-1.5*sf) (-2.35)
(* 30*)   b[s]_1 Excentricity 0 1.5*sf (-2.35) Section ab_1 YOrientation sf 0 0 ;
(* 31*) ElementLoad ab[s]_1_s Element ab[s]_1 SelfWeight LoadCase load_history ;
(* 32*) ElementLoad ab[s]_1_p Element ab[s]_1 InitialDisplacement .01 0 0 0 0 0
(* 33*)   LoadCase load_history Translation Global 0 1 0 Element ab[s]_1
(* 34*)   Normed_x 0 Value 0 StageForCondition half[s]_last ;
(* 35*)
(* 36*) Node a[s]_2 80*sf (-9.53)*sf 35.3 ;
(* 37*) Beam a[s]_2 Nodes a[s]_1 Excentricity 0 (-1.5*sf) (-2.35) a[s]_2
(* 38*)   Section 2_2 YOrientation 0 sf 0 ;
(* 39*) ElementLoad a[s]_2_s Element a[s]_2 SelfWeight LoadCase load_history ;
(* 40*) ElementLoad a[s]_2_p Element a[s]_2 InitialDisplacement .01 0 0 0 0 0
(* 41*)   LoadCase load_history Translation Global 0 0 1 Element a[s]_2 Normed_x 1
(* 42*)   Value 0 StageForCondition half[s]_last ;
(* 43*)
(* 44*) Node b[s]_2 80*sf 9.53*sf 35.3 ;
(* 45*) Beam b[s]_2 Nodes b[s]_1 Excentricity 0 1.5*sf (-2.35) b[s]_2
(* 46*)   Section 2_2 YOrientation 0 sf 0 ;
(* 47*) ElementLoad b[s]_2_s Element b[s]_2 SelfWeight LoadCase load_history ;
(* 48*) ElementLoad b[s]_2_p Element b[s]_2 InitialDisplacement .01 0 0 0 0 0
(* 49*)   LoadCase load_history Translation Global 0 0 1 Element b[s]_2 Normed_x 1
(* 50*)   Value 0 StageForCondition half[s]_last ;
(* 51*)
(* 52*) Beam ab[s]_2 Nodes a[s]_2 b[s]_2 Section 1_1 YOrientation sf 0 0 ;
(* 53*) ElementLoad ab[s]_2_s Element ab[s]_2 SelfWeight
(* 54*)   LoadCase load_history ;
(* 55*) ElementLoad ab[s]_2_p1 Element ab[s]_2 InitialDisplacement .01 0 0 0 0 0
(* 56*)   LoadCase load_history Translation Global 0 1 0 Element ab[s]_2 Normed_x 0
(* 57*)   Value 0 StageForCondition half[s]_last ;
(* 58*)
(* 59*) For i From 0 To 7 {
(* 60*)   Node a[s]_[3+i] 80*sf (-9.53)*sf 47+i*.9 ;
(* 61*)   Beam a[s]_[3+i] Nodes a[s]_[3+i-1] a[s]_[3+i] Section 1_1
(* 62*)     YOrientation 0 sf 0 ;
(* 63*)   ElementLoad a[s]_[3+i]_s Element a[s]_[3+i] SelfWeight LoadCase load_history ;
(* 64*)   ElementLoad a[s]_[3+i]_p Element a[s]_[3+i] InitialDisplacement .01 0 0 0 0 0
(* 65*)     LoadCase load_history Translation Global 0 0 1 Element a[s]_[3+i]
(* 66*)     Normed_x 1 Value 0 StageForCondition half[s]_last ;
(* 67*)
(* 68*)   Node b[s]_[3+i] 80*sf 9.53*sf 47+i*.9 ;
(* 69*)   Beam b[s]_[3+i] Nodes b[s]_[3+i-1] b[s]_[3+i] Section 1_1
(* 70*)     YOrientation 0 sf 0 ;
(* 71*)   ElementLoad b[s]_[3+i]_s Element b[s]_[3+i] SelfWeight LoadCase load_history ;
(* 72*)   ElementLoad b[s]_[3+i]_p Element b[s]_[3+i] InitialDisplacement .01 0 0 0 0 0
(* 73*)     LoadCase load_history Translation Global 0 0 1 Element b[s]_[3+i]
(* 74*)     Normed_x 1 Value 0 StageForCondition half[s]_last ;
(* 75*) }
(* 76*)
(* 77*) Stage mast[s] DayNumber (dayNumber=dayNumber+1) ;
(* 78*)
(* 79*) Note ========== deck ========================================================== ;
```

```
(* 80*)
(* 81*) Node c[s]_9 80*sf (-9.53)*sf 13.25 ;
(* 82*) Joint c[s]_9_tmp Nodes a[s]_1 c[s]_9 Stiffness 1E10 1E10 1E10 1E10 1E10 1E10 ;
(* 83*)
(* 84*) Node d[s]_9 80*sf 9.53*sf 13.25 ;
(* 85*) Joint d[s]_9_tmp Nodes b[s]_1 d[s]_9 Stiffness 1E10 1E10 1E10 1E10 1E10 1E10 ;
(* 86*)
(* 87*) Node e[s]_9 80*sf 0 13.25 ;
(* 88*) Beam ce[s]_9 Nodes c[s]_9 e[s]_9 Section cd_9 YOrientation sf 0 0 ;
(* 89*) ElementLoad ce[s]_9_s Element ce[s]_9 SelfWeight LoadCase load_history ;
(* 90*) ElementLoad ce[s]_9_cast Element ce[s]_9 WeightFromSection slab_cross
(* 91*)   LoadCase load_history ;
(* 92*)
(* 93*) Beam de[s]_9 Nodes e[s]_9 d[s]_9 Section cd_9 YOrientation sf 0 0 ;
(* 94*) ElementLoad de[s]_9_s Element de[s]_9 SelfWeight LoadCase load_history ;
(* 95*) ElementLoad de[s]_9_cast Element de[s]_9 WeightFromSection slab_cross
(* 96*)   LoadCase load_history ;
(* 97*)
(* 98*) Stage platform[s] DayNumber (dayNumber=dayNumber+1) ;
(* 99*)
(*100*) CompositeBeam ce[s]_9_conc FormworkBeam ce[s]_9 Dz 1.075 Section slab_cross ;
(*101*) CompositeBeam de[s]_9_conc FormworkBeam de[s]_9 Dz 1.075 Section slab_cross ;
(*102*)
(*103*) Stage platform[s]_hard DayNumber (dayNumber=dayNumber+1) ;
(*104*)
(*105*) For i From 0 To 7 {
(*106*)   Node c[s]_[8-i] (80-(6.5+9.8*i))*sf (-9.53)*sf 13.25 ;
(*107*)   Beam c[s]_[8-i] Nodes c[s]_[8-i+1] c[s]_[8-i] Section I_profile
(*108*)     YOrientation 0 sf 0 ;
(*109*)   ElementLoad c[s]_[8-i]_s Element c[s]_[8-i] SelfWeight LoadCase load_history ;
(*110*)   ElementLoad c[s]_[8-i]_cast Element c[s]_[8-i]
(*111*)     WeightFromSection slab_long LoadCase load_history ;
(*112*)   ElementLoad c[s]_[8-i]_p Element c[s]_[8-i] InitialDisplacement .01 0 0 0 0 0
(*113*)     LoadCase load_history Translation Global 1 0 0 Element c[s]_[8-i]_conc
(*114*)     Normed_x 1 Value 0 StageForCondition half[s]_last ;
(*115*)
(*116*)   Node d[s]_[8-i] (80-(6.5+9.8*i))*sf 9.53*sf 13.25 ;
(*117*)   Beam d[s]_[8-i] Nodes d[s]_[8-i+1] d[s]_[8-i] Section I_profile
(*118*)     YOrientation 0 sf 0 ;
(*119*)   ElementLoad d[s]_[8-i]_s Element d[s]_[8-i] SelfWeight LoadCase load_history ;
(*120*)   ElementLoad d[s]_[8-i]_cast Element d[s]_[8-i]
(*121*)     WeightFromSection slab_long LoadCase load_history ;
(*122*)   ElementLoad d[s]_[8-i]_p Element d[s]_[8-i] InitialDisplacement .01 0 0 0 0 0
(*123*)     LoadCase load_history Translation Global 1 0 0 Element d[s]_[8-i]_conc
(*124*)     Normed_x 1 Value 0 StageForCondition half[s]_last ;
(*125*)
(*126*)   Node e[s]_[8-i] (80-(6.5+9.8*i))*sf 0 13.25 ;
(*127*)   Beam e[s]_[8-i] Nodes e[s]_[8-i+1] e[s]_[8-i]
(*128*)     Section I_profile YOrientation 0 sf 0 ;
(*129*)   ElementLoad e[s]_[8-i]_s Element e[s]_[8-i] SelfWeight LoadCase load_history ;
(*130*)   ElementLoad e[s]_[8-i]_cast Element e[s]_[8-i]
(*131*)     WeightFromSection slab_long LoadCase load_history ;
```

```
(*132*)
(*133*)   Beam ce[s]_[8-i] Nodes c[s]_[8-i] e[s]_[8-i] Section I_profile
(*134*)     YOrientation sf 0 0 ;
(*135*)   ElementLoad ce[s]_[8-i]_s Element ce[s]_[8-i] SelfWeight
(*136*)     LoadCase load_history ;
(*137*)
(*138*)   Beam de[s]_[8-i] Nodes e[s]_[8-i] d[s]_[8-i] Section I_profile
(*139*)     YOrientation sf 0 0 ;
(*140*)   ElementLoad de[s]_[8-i]_s Element de[s]_[8-i] SelfWeight
(*141*)     LoadCase load_history ;
(*142*)
(*143*)   Cable c[s]_[8-i]_cab Nodes c[s]_[8-i] Excentricity 0 0 0.770 a[s]_[3+i]
(*144*)     Section 1_strands ;
(*145*)   ElementLoad c[s]_[8-i]_cab_s Element c[s]_[8-i]_cab SelfWeight
(*146*)     LoadCase load_history ;
(*147*)   ElementLoad c[s]_[8-i]_cab Element c[s]_[8-i]_cab
(*148*)     InitialDisplacement .01 0 0 0 0 0 LoadCase load_history
(*149*)     Translation Global 0 0 1 Element c[s]_[8-i]_cab Normed_x 0 Value 0
(*150*)     StageForCondition half[s]_last ;
(*151*)
(*152*)   Cable d[s]_[8-i]_cab Nodes d[s]_[8-i] Excentricity 0 0 0.770 b[s]_[3+i]
(*153*)     Section 1_strands ;
(*154*)   ElementLoad d[s]_[8-i]_cab_s Element d[s]_[8-i]_cab SelfWeight
(*155*)     LoadCase load_history ;
(*156*)   ElementLoad d[s]_[8-i]_cab Element d[s]_[8-i]_cab
(*157*)     InitialDisplacement .01 0 0 0 0 0 LoadCase load_history
(*158*)     Translation Global 0 0 1 Element d[s]_[8-i]_cab Normed_x 0 Value 0
(*159*)     StageForCondition half[s]_last ;
(*160*)
(*161*)   Stage mid_deck[s]_[i] DayNumber (dayNumber=dayNumber+1) ;
(*162*)
(*163*)   CompositeBeam c[s]_[8-i]_conc
(*164*)     FormworkBeam c[s]_[8-i] Dz 1.075 Section slab_long ;
(*165*)   ElementLoad c[s]_[8-i]_conc_l Element c[s]_[8-i]_conc Force 0 0 (-27.22)
(*166*)     LoadCase live ;
(*167*)
(*168*)   CompositeBeam d[s]_[8-i]_conc
(*169*)     FormworkBeam d[s]_[8-i] Dz 1.075 Section slab_long ;
(*170*)   ElementLoad d[s]_[8-i]_conc_l Element d[s]_[8-i]_conc Force 0 0 (-27.22)
(*171*)     LoadCase live ;
(*172*)
(*173*)   CompositeBeam e[s]_[8-i]_conc
(*174*)     FormworkBeam e[s]_[8-i] Dz 1.075 Section slab_long ;
(*175*)   ElementLoad e[s]_[8-i]_conc_l Element e[s]_[8-i]_conc Force 0 0 (-27.22)
(*176*)     LoadCase live ;
(*177*)
(*178*)   CompositeBeam ce[s]_[8-i]_conc
(*179*)     FormworkBeam ce[s]_[8-i] Dz 1.075 Section slab_cross ;
(*180*)
(*181*)   CompositeBeam de[s]_[8-i]_conc
(*182*)     FormworkBeam de[s]_[8-i] Dz 1.075 Section slab_cross ;
(*183*)
```

```
(*184*)   Stage mid_deck[s]_[i]_hard DayNumber (dayNumber=dayNumber+1) ;
(*185*)
(*186*)   If (10+i<15) {
(*187*)     Node c[s]_[10+i] tmpx=(80+(6.5+9.8*i))*sf (-9.53)*sf 13.25 ;
(*188*)   }
(*189*)   If (10+i==15) {
(*190*)     Node c[s]_[10+i] tmpx=(80+(6.5+9.8*i)-1.5)*sf (-9.53)*sf 13.25 ;
(*191*)     Support c[s]_[10+i] Node c[s]_[10+i] Stiffness 0 0 1E30 0 0 0 ;
(*192*)   }
(*193*)   If (10+i==16) {
(*194*)     Node c[s]_[10+i] tmpx=(80+(6.5+9.8*i)-9.8)*sf (-9.53)*sf 13.25 ;
(*195*)     Support c[s]_[10+i] Node c[s]_[10+i] Stiffness 0 0 1E30 0 0 0 ;
(*196*)   }
(*197*)   If (10+i==17) {
(*198*)     Node c[s]_[10+i] tmpx=(80+(6.5+9.8*i)-18.1)*sf (-9.53)*sf 13.25 ;
(*199*)     Support c[s]_[10+i] Node c[s]_[10+i] Stiffness 0 0 1E30 0 0 0 ;
(*200*)   }
(*201*)
(*202*)   Beam c[s]_[10+i] Nodes c[s]_[10+i-1] c[s]_[10+i] Section I_profile
(*203*)     YOrientation 0 (-sf) 0 ;
(*204*)   ElementLoad c[s]_[10+i]_s Element c[s]_[10+i] SelfWeight
(*205*)     LoadCase load_history ;
(*206*)   ElementLoad c[s]_[10+i]_cast Element c[s]_[10+i]
(*207*)     WeightFromSection slab_long LoadCase load_history ;
(*208*)   ElementLoad c[s]_[10+i]_p Element c[s]_[10+i]
(*209*)     InitialDisplacement .01 0 0 0 0 0 LoadCase load_history
(*210*)     Translation Global 1 0 0 Element c[s]_[10+i]_conc
(*211*)     Normed_x 1 Value 0 StageForCondition half[s]_last ;
(*212*)
(*213*)   Node d[s]_[10+i] tmpx 9.53*sf 13.25 ;
(*214*)   If (10+i==15) {
(*215*)     Support d[s]_[10+i] Node d[s]_[10+i] Stiffness 0 0 1E30 0 0 0 ;
(*216*)   }
(*217*)   If (10+i==16) {
(*218*)     Support d[s]_[10+i] Node d[s]_[10+i] Stiffness 0 0 1E30 0 0 0 ;
(*219*)   }
(*220*)   If (10+i==17) {
(*221*)     Support d[s]_[10+i] Node d[s]_[10+i] Stiffness 0 0 1E30 0 0 0 ;
(*222*)   }
(*223*)
(*224*)   Beam d[s]_[10+i] Nodes d[s]_[10+i-1] d[s]_[10+i] Section I_profile
(*225*)     YOrientation 0 (-sf) 0 ;
(*226*)   ElementLoad d[s]_[10+i]_s Element d[s]_[10+i] SelfWeight
(*227*)     LoadCase load_history ;
(*228*)   ElementLoad d[s]_[10+i]_cast Element d[s]_[10+i]
(*229*)     WeightFromSection slab_long LoadCase load_history ;
(*230*)   ElementLoad d[s]_[10+i]_p Element d[s]_[10+i]
(*231*)     InitialDisplacement .01 0 0 0 0 0 LoadCase load_history
(*232*)     Translation Global 1 0 0 Element d[s]_[10+i]_conc
(*233*)     Normed_x 1 Value 0 StageForCondition half[s]_last ;
(*234*)
(*235*)   Node e[s]_[10+i] tmpx 0 13.25 ;
```

```
(*236*)   Beam e[s]_[10+i] Nodes e[s]_[10+i-1] e[s]_[10+i] Section I_profile
(*237*)     YOrientation 0 (-sf) 0 ;
(*238*)   ElementLoad e[s]_[10+i]_s Element e[s]_[10+i] SelfWeight
(*239*)     LoadCase load_history ;
(*240*)   ElementLoad e[s]_[10+i]_cast Element e[s]_[10+i]
(*241*)     WeightFromSection slab_long LoadCase load_history ;
(*242*)
(*243*)   Beam ce[s]_[10+i] Nodes c[s]_[10+i] e[s]_[10+i]
(*244*)     Section I_profile YOrientation sf 0 0 ;
(*245*)   ElementLoad ce[s]_[10+i]_s Element ce[s]_[10+i]
(*246*)     SelfWeight LoadCase load_history ;
(*247*)
(*248*)   Beam de[s]_[10+i] Nodes e[s]_[10+i] d[s]_[10+i]
(*249*)     Section I_profile YOrientation sf 0 0 ;
(*250*)   ElementLoad de[s]_[10+i]_s Element de[s]_[10+i]
(*251*)     SelfWeight LoadCase load_history ;
(*252*)
(*253*)   Cable c[s]_[10+i]_cab Nodes c[s]_[10+i] Excentricity 0 0 0.770 a[s]_[3+i]
(*254*)     Section 1_strands ;
(*255*)   ElementLoad c[s]_[10+i]_cab_s Element c[s]_[10+i]_cab
(*256*)     SelfWeight LoadCase load_history ;
(*257*)
(*258*)   Cable d[s]_[10+i]_cab Nodes d[s]_[10+i] Excentricity 0 0 0.770 b[s]_[3+i]
(*259*)     Section 1_strands ;
(*260*)   ElementLoad d[s]_[10+i]_cab_s Element d[s]_[10+i]_cab
(*261*)     SelfWeight LoadCase load_history ;
(*262*)
(*263*)   If (10+i<15) {
(*264*)     ElementLoad c[s]_[10+i]_cab Element c[s]_[10+i]_cab
(*265*)       InitialDisplacement .01 0 0 0 0 0 LoadCase load_history
(*266*)       Translation Global 0 0 1 Element c[s]_[10+i]_cab
(*267*)       Normed_x 0 Value 0 StageForCondition half[s]_last ;
(*268*)     ElementLoad d[s]_[10+i]_cab Element d[s]_[10+i]_cab
(*269*)       InitialDisplacement .01 0 0 0 0 0 LoadCase load_history
(*270*)       Translation Global 0 0 1 Element d[s]_[10+i]_cab
(*271*)       Normed_x 0 Value 0 StageForCondition half[s]_last ;
(*272*)   }
(*273*)   If (10+i==15) {
(*274*)     ElementLoad c[s]_[10+i]_cab Element c[s]_[10+i]_cab
(*275*)       InitialDisplacement .01 0 0 0 0 0 LoadCase load_history
(*276*)       Translation Global 1 0 0 Element c[s]_[10+i]_cab Normed_x 1 Value 0
(*277*)       StageForCondition half[s]_last ;
(*278*)     ElementLoad d[s]_[10+i]_cab Element d[s]_[10+i]_cab
(*279*)       InitialDisplacement .01 0 0 0 0 0 LoadCase load_history
(*280*)       Translation Global 1 0 0 Element d[s]_[10+i]_cab Normed_x 1
(*281*)       Value 0 StageForCondition half[s]_last ;
(*282*)   }
(*283*)   If (10+i==16) {
(*284*)     ElementLoad c[s]_[10+i]_cab Element c[s]_[10+i]_cab
(*285*)       InitialDisplacement .01 0 0 0 0 0 LoadCase load_history
(*286*)       Translation Global 1 0 0 Element c[s]_[10+i]_cab
(*287*)       Normed_x 1 Value 0 StageForCondition half[s]_last ;
```

```
(*288*)     ElementLoad d[s]_[10+i]_cab Element d[s]_[10+i]_cab
(*289*)       InitialDisplacement .01 0 0 0 0 0 LoadCase load_history
(*290*)       Translation Global 1 0 0 Element d[s]_[10+i]_cab
(*291*)       Normed_x 1 Value 0 StageForCondition half[s]_last ;
(*292*)   }
(*293*)   If (10+i==17) {
(*294*)     ElementLoad c[s]_[10+i]_cab Element c[s]_[10+i]_cab
(*295*)       InitialDisplacement .01 0 0 0 0 0 LoadCase load_history
(*296*)       Translation Global 1 0 0 Element c[s]_[10+i]_cab
(*297*)       Normed_x 1 Value 0 StageForCondition half[s]_last ;
(*298*)     ElementLoad d[s]_[10+i]_cab Element d[s]_[10+i]_cab
(*299*)       InitialDisplacement .01 0 0 0 0 0 LoadCase load_history
(*300*)       Translation Global 1 0 0 Element d[s]_[10+i]_cab
(*301*)       Normed_x 1 Value 0 StageForCondition half[s]_last ;
(*302*)   }
(*303*)
(*304*)   Stage side_deck[s]_[i] DayNumber (dayNumber=dayNumber+1) ;
(*305*)
(*306*)   If (10+i<=15) {
(*307*)     CompositeBeam c[s]_[10+i]_conc FormworkBeam c[s]_[10+i] Dz 1.075
(*308*)       Section slab_long ;
(*309*)     ElementLoad c[s]_[10+i]_conc_l Element c[s]_[10+i]_conc
(*310*)       Force 0 0 (-27.22) LoadCase live ;
(*311*)
(*312*)     CompositeBeam d[s]_[10+i]_conc FormworkBeam d[s]_[10+i] Dz 1.075
(*313*)       Section slab_long ;
(*314*)     ElementLoad d[s]_[10+i]_conc_l Element d[s]_[10+i]_conc
(*315*)       Force 0 0 (-27.22) LoadCase live ;
(*316*)
(*317*)     CompositeBeam e[s]_[10+i]_conc FormworkBeam e[s]_[10+i] Dz 1.075
(*318*)       Section slab_long ;
(*319*)     ElementLoad e[s]_[10+i]_conc_l Element e[s]_[10+i]_conc
(*320*)       Force 0 0 (-27.22) LoadCase live ;
(*321*)
(*322*)     CompositeBeam ce[s]_[10+i]_conc FormworkBeam ce[s]_[10+i] Dz 1.075
(*323*)       Section slab_cross ;
(*324*)
(*325*)     CompositeBeam de[s]_[10+i]_conc FormworkBeam de[s]_[10+i] Dz 1.075
(*326*)       Section slab_cross ;
(*327*)   } Else {
(*328*)     CompositeBeam c[s]_[10+i]_conc
(*329*)       FormworkBeam c[s]_[10+i] Dz .125 Section slab_long_end ;
(*330*)     ElementLoad c[s]_[10+i]_conc_l Element c[s]_[10+i]_conc
(*331*)       Force 0 0 (-27.22) LoadCase live ;
(*332*)
(*333*)     CompositeBeam d[s]_[10+i]_conc
(*334*)       FormworkBeam d[s]_[10+i] Dz .125 Section slab_long_end ;
(*335*)     ElementLoad d[s]_[10+i]_conc_l Element d[s]_[10+i]_conc
(*336*)       Force 0 0 (-27.22) LoadCase live ;
(*337*)
(*338*)     CompositeBeam e[s]_[10+i]_conc
(*339*)       FormworkBeam e[s]_[10+i] Dz .125 Section slab_long_end ;
```

```
(*340*)     ElementLoad e[s]_[10+i]_conc_l Element e[s]_[10+i]_conc
(*341*)       Force 0 0 (-27.22) LoadCase live ;
(*342*)
(*343*)     CompositeBeam ce[s]_[10+i]_conc FormworkBeam ce[s]_[10+i] Dz .125
(*344*)       Section slab_cross_end ;
(*345*)
(*346*)     CompositeBeam de[s]_[10+i]_conc FormworkBeam de[s]_[10+i] Dz .125
(*347*)       Section slab_cross_end ;
(*348*)   }
(*349*)
(*350*)   Stage side_deck[s]_[i]_hard DayNumber (dayNumber=dayNumber+1) ;
(*351*) }
(*352*)
(*353*) Node c[s]_0 0 (-9.53)*sf 13.25 ;
(*354*) Beam c[s]_0 Nodes c[s]_0 c[s]_1 Section I_profile YOrientation 0 (-sf) 0 ;
(*355*) ElementLoad c[s]_0_s Element c[s]_0 SelfWeight LoadCase load_history ;
(*356*) ElementLoad c[s]_0_cast Element c[s]_0 WeightFromSection slab_long
(*357*)   LoadCase load_history ;
(*358*)
(*359*) Node d[s]_0 0 9.53*sf 13.25 ;
(*360*) Beam d[s]_0 Nodes d[s]_0 d[s]_1 Section I_profile YOrientation 0 (-sf) 0 ;
(*361*) ElementLoad d[s]_0_s Element d[s]_0 SelfWeight LoadCase load_history ;
(*362*) ElementLoad d[s]_0_cast Element d[s]_0 WeightFromSection slab_long
(*363*)   LoadCase load_history ;
(*364*)
(*365*) Node e[s]_0 0 0 13.25 ;
(*366*) Beam e[s]_0 Nodes e[s]_0 e[s]_1 Section I_profile YOrientation 0 (-sf) 0 ;
(*367*) ElementLoad e[s]_0_s Element e[s]_0 SelfWeight LoadCase load_history ;
(*368*) ElementLoad e[s]_0_cast Element e[s]_0 WeightFromSection slab_long
(*369*)   LoadCase load_history ;
(*370*)
(*371*) Stage half[s]_last DayNumber (dayNumber=dayNumber+1) ;
```

Line 3: definition of the variable "sf" which assumes the value 1 if the variable "s" defined in the main input file listed in section D.1 is set to 2 and -1 if "s" is set to 1.

Line 4: definition of the variable "dayNumber" needed to generated the day numbers assigned to the stage objects. This variable is also initialized using the variable "s".

Line 5: definition of the variable "tmpx" used to generate X coordinates.

Lines 9-77: definition of the mast.

Lines 17-19: first definition of a conditional load.

Lines 59-75: generation with a pre-processor for-loop of some mast elements with their related objects (nodes and loads).

Line 77: first definition of a stage object.

Lines 105-371: definition of the model objects related to the deck and the cables.

Lines 81-103: definition of the junction between the mast and the deck.

Line 100: first definition of a composite beam element.

Lines 105-351: for-loop in which the following model objects are generated eight times using the loop's control variable "i":

- a deck segment on the mid-span side of the mast,
- the cables supporting it,
- its cast concrete slab (taken into account with composite elements),
- a deck segment on the side-span side of the mast,
- the cables supporting it and,
- its cast concrete slab.

Lines 105-184: definition of the mid-span deck segment with its cables and cast concrete.

Lines 186-350: definition of the side-span deck segment with its cables and cast concrete.

Lines 186-188: first of many if-checks needed to take into account the changes of the deck at the supports while generating the model objects in the for-loop.

Lines 353-371: Model objects simulating the deck (without concrete) from the last cable anchoring to the center of the mid-span.

Curriculum Vitae

Personal data

first name	Pietro
name	Pedrozzi
born	November 21st, 1975
email	pietro.pedrozzi@alumni.ethz.ch

Education

03/1999 - 06/2004	**Dissertation** at the Institute of Structural Engineering of the Swiss Federal Institute of Technology (ETH) Zurich, Switzerland.
10/1994 - 02/1999	**Diploma** in civil engineering at the ETH Zurich.
09/1990 - 06/1994	**Matura** at the Liceo Cantonale di Bellinzona.

Professional activities

03/1999 - 06/2004	**Assistant** at the Institute of Structural Engineering of the ETH Zurich.
07/1997 - 08/1997	**Internship** at the civil engineering firm Passera & Pedretti in Lugano.
10/1996 - 03/1997	**Assistant** at the Institute for Geotechnical Engineering of the ETH Zurich.