Czech Technical University in Prague
Faculty of Civil Engineering
Department of Mechanics

SOFTWARE TOOL DEVELOPMENT FOR CIVIL ENGINEERING PROBLEMS

by

Tomáš Koudelka

Habilitation Thesis

Prague, February 2023

Acknowledgement

I would like to express my deep gratitude to Professor Zdeněk Bittnar, who gave me the opportunity to become a member of the Department of Mechanics and was my supervisor during my doctoral studies. Another source of inspiration, tremendous support and encouragement also came from Professor Jiří Šejnoha, who led my research activities at the beginning of my research carrier. In addition, a special appreciation is to my friends and colleagues Tomáš Krejčí, Jaroslav Kruis and Jiří Maděra, partners in the software development group of which I am honoured to be a member. They provided valuable comments and inspiring discussions on topics addressed in this thesis.

I would also like to thank members of the Department, Michal Šejnoha, Daniel Rypl, Matěj Lepš and Bořek Patzák, for their encouragement, assistance and many enlightening discussions. Lastly, I would like to thank all the other people who encouraged and supported me with patience in writing this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many problems in the field of civil engineering can be described with the help of differential equations. Their manual solution is very demanding and for real engineering problems impossible in many cases. The development of numerical methods for their solution was thus tightly connected with the emergence of computer technology in the common practice in the last decades. Differential equations describing the given problem is often solved with the help of finite element method (FEM) and there is a lot of commercial software packages employing FEM that were developed in past decades ranging from more general ones such as ABAQUS, ADINA, ANSYS, COMSOL to more specific such as ATHENA, CSI Bridge, RAM, RFEM, SCIA Engineer, SAP2000 to name only a few of them.

Many of these commercial software packages involve possibility of implementation user defined element, material model or even a solver but there are areas such as nonlocal material models that can be hardly implemented without the access to source codes that are not available in these cases. There may also be some hidden implementation details that can influence the computation significantly and that are not clearly described in the documentation.

These reasons led many researchers to develop their own codes including author's research group. The development of FEM package SIFEL has started in 2001 within the EU project MAECENAS in the part addressed to the simulation of concrete behavior used at nuclear reactor vessels. Fundamental concepts of SIFEL stemmed from the requirements of several universities involved in the project and most of them was preserved in course of development time. Authors also decided to develop code as an open source under the GPL license.

The document is organized as follows. The first part contains three chapters. The first chapter deals with the history and fundamental concepts of the SIFEL package. It is based on the extended paper presented at the 10th International Conference on Computational Structures Technology 2010. The next chapter is devoted to exploiting new advanced features of the programming language in the finite element (FE) code design, and there is also a performance comparison of particular approaches.

The second part deals with the implementation of stage construction. It contains three chapters with real-world engineering problems solved by SIFEL. The motivation originated in 2005 when the analysis of the casting procedure of the thick foundation slab in Těšnov was conducted. The analysis of the foundation slab is described in the first chapter of the second part.

Later in 2013, some lack of implementation was revealed in connection with the TAČR project, where simulation of the creep behaviour of the pre-stressed box girder bridge in Mělník where the simulation of construction stages played a significant role. Thus the second chapter addresses the creep analysis of the bridge in Mělník and a simulation of the gradual bridge construction.

The last extension of the implementation in this field was connected with the simulation of engineering barriers in nuclear waste repositories where the nonlinear material model needed a special treatment of load increment vector due to removed elements. The last chapter, therefore, concerns the implementation of the model for the expansive clay and the simulation of the structural part removal. The proposed techniques are demonstrated in the trench excavation problem.

Part I

Development strategies of the finite element code

Chapter 2

Moderate use of object oriented programming for scientific computing

2.1 Introduction

Engineering and scientific computing was mostly based on the usage of the FORTRAN programming language for a long time. Since 90s, other programming languages have emerged in the field. The most important languages are C and later C++. The C language enables dynamic memory allocation and better memory management which was understood as a significant advantage in comparison with the FORTRAN 77 common blocks. The C++ language is an object-oriented language enabling data encapsulation, function/operator overloading, multiple inheritance and templates. Many additional extensions were established by the newer standards starting with C++11 up to the latest C++20.

Over time, the FORTRAN was changed, and some features of C and C++ languages were incorporated. The newer language versions, called FORTRAN 90 and FORTRAN 95, adopted dynamic memory allocation and some object-oriented features. These changes made the FORTRAN language attractive again. The latest FORTRAN 2008 involved support for object-oriented programming and support for parallel programming. In addition to FORTRAN, some other object-oriented languages have emerged, such as Java and C#. With respect to experience with the C++ concept, the new languages adopted only some features of the original C++. One-level inheritance was an example, while the multiple inheritance was replaced by so-called interfaces, which seemed more acceptable.

This chapter summarizes experiences with programming languages based on the development of large finite element code SIFEL. It is an open-source code [SIFEL, 2022], [Koudelka et al., 2011] that has been developing for more than 20 years at the Department of Mechanics of Civil Engineering Faculty of the Czech Technical University in Prague. Except for the author, there are Tomáš Krejčí and Jaroslav Kruis who belongs to the principal developers of the software. The development of the code started in 2001 within the scope of the European research project MAECENAS which dealt with the assessment of properties of reactor vessels of nuclear power plants at the end of their service life. The project was solved at several universities across Europe (Glasgow, Nantes, Padova,

Prague and Sheffield), and the aim was to develop extensible computer code for coupled hydro-thermo-mechanical analysis. There were some computer codes for particular problems at involved universities, but their connection or merging was impossible. Therefore, the development of a new code was started, and the project participants determined the following requirements:

- Portability of the code - universities had different hardware and software equipment. Especially, the portability between different operating systems was required (Linux, Windows, Unix).
- Simple programming techniques - the members of the project were experts in the branch of mechanical and transport processes and they knew programming languages but they were not professional programmers. Source codes should be comprehensible for all team members as well as for new participants.
- Speed of code - the programming language should be compiled (FORTRAN, C++) rather than precompiled and interpreted (Java).

Interpreted languages were rejected due to speed requirements. FORTRAN 77, FORTRAN 90 and C++ were compared from the above points of view and the summary follows:

C++
+ very portable language,
+ source codes using basics of C++ are comprehensible,
+ C++ compilers produce fast executables,
− extensive usage of object oriented programming techniques decreases clarity for new participants.

F77
+ very portable language,
+ language syntax is comprehensible,
+ compilers can produce from the numeric source code very fast executables,
− memory management is poor because the standard does not support dynamic memory allocation,
− memory organized in Fortran COMMON blocks can be understood with problems.

F90
+ language syntax is comprehensible, it supports dynamic memory management,
+ compilers produce very fast numeric code,
− portability was limited due to non-availability of compilers supporting FORTRAN 90 at that time,
− uncertainties about the future development of FORTRAN 90 compilers.

It was concluded that C++ language would be used but without most object-oriented programming features and concepts, which cause main difficulties for new users.

2.2 Object oriented concepts of C++ in practical usage

2.2.1 Main concepts of the OOP

The object-oriented programming consists of three main concepts:

- Data abstraction and encapsulation
- Inheritance
- Polymorphism

Data abstraction means that data and operations, which operate on that data, are viewed as one data aggregate, and they are considered by the language similarly to built-in data types. These data aggregates have to be defined by users, and therefore they are sometimes called user-defined types. C++ calls these data aggregates *classes* and variables type of class are called *objects* or *instances*. Every class has its data members - *attributes* and functions called *methods*. Access to data members can be limited by the application of different specifiers (*private, public, protected*). Merging data and functions for manipulation of that data in one data type is called encapsulation.

Classes can be composed of the basic data types, but they can also be derived from the existing ones. The derivation of a new class from one or more existing classes is called inheritance. The new class may expand the inherited content depending on the actual needs, and usually, it changes the implementation of some or even all of the methods. Thus, the inheritance provides reusability of the existing code, which can be adjusted for user purposes.

Every class should have a separate part of the user interface and its implementation. The interface shows the user *what* the class tends to do, and implementation shows *how* it is done. Changes in implementation should not influence the interface, and they remain hidden from the user of the class. It is possible to create several classes by inheritance from the base class with the same interface but a different implementation. This feature is called polymorphism, and it makes possible unified access to instances of different classes via interfaces even though the performed actions can be (and usually they are) different.

It should be noted that the C++ implementation of polymorphism is established on the pointers to objects of base class type, and the rule that a derived class type may be used if the base class type is required. If the object of the derived class is used directly instead of the base class type object, the slicing problem takes place. In such a case, new data members, together with all specialized features of the derived class object, are dropped, which is not desirable behaviour usually. Thus the using of C++ polymorphism causes, in practice, memory fragmentation for large arrays of pointers to polymorphic objects due to the need of dynamic memory allocation of the individual array elements. There may also be some performance penalties due to the need for pointer dereferencing.

2.2.2 Strong and weak points of the C++ OOP and generic programming

In this subsection, the strong and weak points of OOP and generic programming will be discussed with respect to the given requirements for the SIFEL development. From the portability point of view, most of the OOP features in C++ were perfectly portable across the compilers and platforms. The other thing was generic programming with the Standard Template Library (STL), which contained many useful data structures and algorithms that were easy to use, but its implementation suffered from compatibility issues on former C++ compilers until 2005 at least. The situation changed gradually after the release of the C++11 standard, and also compiler producers made a significant effort to make their compilers more standard compliant. Another issue with templates was automatic template code generation by the compiler, which could lead to huge executable files. There were also difficulties with compiler error messages regarding templates and debugging of code with templates. These reasons led to the suppression of templates in the SIFEL code to a minimum level at the beginning of the development. Similar conclusions about templates and exceptions were made by Mozilla.org and Google (see [Mozilla, 2014], [Google, 2014]). On the other hand, this might be revised in view of new C++ standards where new features like `constexpr` or lambda expressions together with template meta-programming allowed for polymorphic behaviour without memory fragmentation, and the performance improvement might be comparable with the C style programming. More details are given in Chapter 3.

The second requirement was the comprehensibility of the code. SIFEL was developed by a core team, which remained almost the same over time, and a team of participants, which changed occasionally. These participants were often master or doctoral students with low or zero programming skills. In many cases, they solved some specific problems not implemented in the code, and they needed to implement them in a reasonable time.

Data abstraction and encapsulation were found as good concepts, and they were comprehensible for all participants of the development team. Data were combined with essential and interface functions that initialise them and performed basic operations. Compared to the usual recommendations, the data was left public at the beginning, and it could be changed to private later depending on needs and experiences when the code become stable.

Polymorphism and inheritance were found that might cause problems. In procedural programming, these concepts correspond to classical `switch` or `if` statements, which can be quite huge, but the user knows which function may be called at the compile time already. Even well-written OOP code using inheritance and polymorphism can become hardly readable because the type of object is known at runtime only and which function is being called is hidden in the so-called virtual method table (VMT). Generally, for the given function call, there is no portable way how to determine which functions could be called at the source code level. That can be partially mitigated by the use of development tools with the intelligent code analyser and visualisation of class relationships, but there are issues with the portability of these tools.

Inheritance was found to be a powerful tool for systems whose hierarchy is known in advance and does not change too much in time. We found that a working cycle of implementing an engineering problem starts with a low knowledge base. At this level, the

proper design of classes is complicated, and with increasing knowledge of the problem, the hierarchy and system of classes change. New experiences and new needs often required a redesign of class hierarchy, and from this point of view, classical procedural programming was found to be more flexible.

On the other hand, a well-designed class system with inheritance and polymorphism decreases the amount of work for maintenance, updates or extension, which is a strong point of OOP. This feature is often preferred over performance reasons because as the code grows, the maintenance time becomes much more significant, and it should be proportionally added to the total sum of time needed to solve a particular problem.

From the performance point of view, the polymorphism causes a performance penalty due to the pointer dereferencing and mainly due to memory fragmentation, as was stated in the previous subsection. Of course, these penalties depend on the C++ compiler implementation and used hardware. Some comparisons of different FEM code implementation strategies will be given in Chapter 3. Exceptions also belong among object properties of C++ language. They represents an attempt to solve the error handling problem but at the cost of performance degradation and executable code bloating [Google, 2014], [Clang, 2012]. Also, writing exception-safe code can be quite demanding; see [Meyers, 2005] and [Alexandrescu, 2000]. It was concluded not to use exceptions in SIFEL.

2.3 SIFEL

The development of SIFEL code was originally intended for the solution of coupled hydro-thermo-mechanical problems and the code was supposed to be easily extended. This section is split into several subsections. Subsection 2.3.1 describes very briefly theoretical background of coupled thermo-hydro-mechanical problem discretized by FEM. Subsection 2.3.2 deals with the code structure while the last subsection 2.3.3 describes the extensibility of the code.

2.3.1 Problem description

The thermo-hydro-mechanical (THM) analysis can serve as an example of general coupled problem. FEM discretization in space of a THM problem leads to the following system of ordinary differential equations

$$\begin{pmatrix} \boldsymbol{C}_{uu} & \boldsymbol{C}_{uT} & \boldsymbol{C}_{up_1} & \boldsymbol{C}_{up_2} \\ \boldsymbol{C}_{Tu} & \boldsymbol{C}_{TT} & \boldsymbol{C}_{Tp_1} & \boldsymbol{C}_{Tp_2} \\ \boldsymbol{C}_{p_1u} & \boldsymbol{C}_{p_1T} & \boldsymbol{C}_{p_1p_1} & \boldsymbol{C}_{p_1p_2} \\ \boldsymbol{C}_{p_2u} & \boldsymbol{C}_{p_2T} & \boldsymbol{C}_{p_2p_1} & \boldsymbol{C}_{p_2p_2} \end{pmatrix} \begin{pmatrix} \dot{\boldsymbol{d}}_u \\ \dot{\boldsymbol{d}}_T \\ \dot{\boldsymbol{d}}_{p_1} \\ \dot{\boldsymbol{d}}_{p_2} \end{pmatrix} + \tag{2.1}$$

$$+ \begin{pmatrix} \boldsymbol{K}_{uu} & \boldsymbol{K}_{uT} & \boldsymbol{K}_{up_1} & \boldsymbol{K}_{up_2} \\ \boldsymbol{K}_{Tu} & \boldsymbol{K}_{TT} & \boldsymbol{K}_{Tp_1} & \boldsymbol{K}_{Tp_2} \\ \boldsymbol{K}_{p_1u} & \boldsymbol{K}_{p_1T} & \boldsymbol{K}_{p_1p_1} & \boldsymbol{K}_{p_1p_2} \\ \boldsymbol{K}_{p_2u} & \boldsymbol{K}_{p_2T} & \boldsymbol{K}_{p_2p_1} & \boldsymbol{K}_{p_2p_2} \end{pmatrix} \begin{pmatrix} \boldsymbol{d}_u \\ \boldsymbol{d}_T \\ \boldsymbol{d}_{p_1} \\ \boldsymbol{d}_{p_2} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_u \\ \boldsymbol{f}_T \\ \boldsymbol{f}_{p_1} \\ \boldsymbol{f}_{p_2} \end{pmatrix},$$

where $\boldsymbol{K}$ denotes the stiffness or conductivity matrices, $\boldsymbol{C}$ denotes the capacity matrices and $\boldsymbol{f}$ denotes the prescribed nodal forces or prescribed nodal flux resultants, $\boldsymbol{d}$ denotes the vector of unknowns and $\dot{\boldsymbol{d}}$ denotes the time derivates of unknowns. The subscript

u denotes the displacements, T denotes the temperature, p_1 and p_2 denote quantities associated with the mass transfer. In the case of moisture transfer, they may denote the pressure of liquid water and water vapour. It should be noted that the system (2.1) is nonlinear generally.

The stiffness and conductivity matrices (denoted by $\boldsymbol{K}$ with appropriate subscripts) generally have the form

$$\boldsymbol{K}_{\alpha\beta} = \int_{\Omega} \boldsymbol{B}_{\alpha}^{\mathrm{T}} \boldsymbol{D}_{\alpha\beta} \boldsymbol{B}_{\beta} \mathrm{d}\Omega \ , \tag{2.2}$$

where $\boldsymbol{B}_{\alpha}$ and $\boldsymbol{B}_{\beta}$ denote the gradient matrices, $\boldsymbol{D}_{\alpha\beta}$ denotes the matrix of stiffness or conductivity of the material and the subscripts α and β substitute any of subscripts u, T, p_1 or p_2. Similarly, the capacity matrices (denoted by $\boldsymbol{C}$ with appropriate subscripts) have generally the form

$$\boldsymbol{C}_{\alpha\beta} = \int_{\Omega} \boldsymbol{N}_{\alpha}^{\mathrm{T}} \boldsymbol{H}_{\alpha\beta} \boldsymbol{N}_{\beta} \mathrm{d}\Omega \ , \tag{2.3}$$

where $\boldsymbol{N}_{\alpha}$ and $\boldsymbol{N}_{\beta}$ denote the matrices of base functions and $\boldsymbol{H}_{\alpha\beta}$ denotes the matrix of material parameters.

Prescribed nodal force vector can be defined as

$$\boldsymbol{f}_{u} = \int_{\Omega} \boldsymbol{N}_{u}^{\mathrm{T}} \boldsymbol{N}_{u} \hat{\boldsymbol{b}} \, \mathrm{d}\Omega + \int_{\Gamma_t} \boldsymbol{N}_{u}^{\mathrm{T}} \boldsymbol{N}_{u} \hat{\boldsymbol{t}} \, \mathrm{d}\Gamma_t, \tag{2.4}$$

where the vectors $\hat{\boldsymbol{b}}$ and $\hat{\boldsymbol{t}}$ represent nodal values of body forces and surface tractions, respectively.The vectors $\boldsymbol{f}_{T}$ and $\boldsymbol{f}_{pi}$ contain flux resultants due to prescribed nodal fluxes and can be defined in the from

$$\boldsymbol{f}_{T} = \int_{\Gamma_T} \boldsymbol{N}_{T}^{\mathrm{T}} \hat{q}_{T} \mathrm{d}\Gamma_{T} \ , \quad \boldsymbol{f}_{pi} = \int_{\Gamma_{pi}} \boldsymbol{N}_{pi}^{\mathrm{T}} \hat{q}_{pi} \mathrm{d}\Gamma_{pi}, \tag{2.5}$$

where $\hat{q}_T$ denotes the prescribed heat boundary flux and $\hat{q}_{pi}$ denotes the prescribed mass boundary fluxes of the i-th quantity .

For better understanding, the following extension of the mechanical analysis is presented. Let an elastic material be assumed. The constitutive equation (Hook's law) has the form

$$\boldsymbol{\sigma} = \boldsymbol{D}_{uu} \boldsymbol{\varepsilon}(\boldsymbol{u}) \tag{2.6}$$

and it relates the strains $\boldsymbol{\varepsilon}$ and stresses $\boldsymbol{\sigma}$. It should be noted that the strains depend on displacements $\boldsymbol{u}$ which are discretized and the nodal displacements are denoted by $\boldsymbol{d}_u$. The mechanical problem with negligible inertial forces can be written in the form

$$\boldsymbol{K}_{uu} \boldsymbol{d}_{u} = \boldsymbol{f}_{u} \ , \tag{2.7}$$

where $\boldsymbol{K}_{uu}$ denotes the stiffness matrix and $\boldsymbol{f}_{u}$ denotes the vector of prescribed nodal forces. Equation (2.7) expresses the equilibrium condition.

If the non-constant temperature plays a role, the constitutive relationship (2.6) has to be replaced by the following constitutive equation

$$\boldsymbol{\sigma} = \boldsymbol{D}_{uu} \boldsymbol{\varepsilon}(\boldsymbol{u}) + \boldsymbol{D}_{uT} \nabla T \ , \tag{2.8}$$

where the temperature T occurs. Moreover, the constitutive relationship between the heat flux $\boldsymbol{q}$ and the temperature gradient is needed and it has the form

$$\boldsymbol{q} = \boldsymbol{D}_{TT} \nabla T \ . \tag{2.9}$$

It is usually accepted that the heat flux is independent of the displacements $\boldsymbol{u}$. Equilibrium condition (2.7) is therefore extended with the heat balance equation for the stationary heat transfer and the system of equations has the form

$$\begin{pmatrix} \boldsymbol{K}_{uu} & \boldsymbol{K}_{uT} \\ \boldsymbol{0} & \boldsymbol{K}_{TT} \end{pmatrix} \begin{pmatrix} \boldsymbol{d}_u \\ \boldsymbol{d}_T \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_u \\ \boldsymbol{f}_T \end{pmatrix} \ . \tag{2.10}$$

The first equation in the system (2.10) expresses the equilibrium condition while the second equation in the system (2.10) expresses the heat balance condition. The zero block in the heat balance equation is consequence of the independence of the heat transfer on the mechanical problem but on the contrary, the mechanical problem is coupled with the heat transfer by the matrix $\boldsymbol{K}_{ut}$.

Additional variables can be introduced in the constitutive equations and additional balance equations can be added to the system. The thermo-mechanical problem (2.10) extended by the pore pressures and capacity terms due to the non-stationary transfer results in the form (2.1).

2.3.2 Code structure

The code is split into independent parts which deal with a single physics problem. The part dealing with mechanical analysis is denoted MEFEL, and the part dealing with transport processes is denoted TRFEL. There is also a part called GEFEL that collects common features of all problems, and the part METR addresses the coupling of individual physics problems.

Let the matrix $\boldsymbol{K}$ defined in (2.1) be assumed. It can be split into submatrices indicated by the lines

$$\boldsymbol{K} = \left(\begin{array}{c|ccc} \boldsymbol{K}_{uu} & \boldsymbol{K}_{uT} & \boldsymbol{K}_{up_1} & \boldsymbol{K}_{up_2} \\ \hline \boldsymbol{K}_{Tu} & \boldsymbol{K}_{TT} & \boldsymbol{K}_{Tp_1} & \boldsymbol{K}_{Tp_2} \\ \boldsymbol{K}_{p_1u} & \boldsymbol{K}_{p_1T} & \boldsymbol{K}_{p_1p_1} & \boldsymbol{K}_{p_1p_2} \\ \boldsymbol{K}_{p_2u} & \boldsymbol{K}_{p_2T} & \boldsymbol{K}_{p_2p_1} & \boldsymbol{K}_{p_2p_2} \end{array} \right) \tag{2.11}$$

The diagonal block $\boldsymbol{K}_{uu}$ is the stiffness matrix and it associated with a pure mechanical analysis. This submatrix is assembled in the part MEFEL. Second diagonal block

$$\begin{pmatrix} \boldsymbol{K}_{TT} & \boldsymbol{K}_{Tp_1} & \boldsymbol{K}_{Tp_2} \\ \boldsymbol{K}_{p_1T} & \boldsymbol{K}_{p_1p_1} & \boldsymbol{K}_{p_1p_2} \\ \boldsymbol{K}_{p_2T} & \boldsymbol{K}_{p_2p_1} & \boldsymbol{K}_{p_2p_2} \end{pmatrix} \tag{2.12}$$

is the block of conductivity matrix associated with the transport process where heat and moisture is assumed. This submatrix is assembled in the part TRFEL. Two offdiagonal submatrices

$$\begin{pmatrix} \boldsymbol{K}_{uT} & \boldsymbol{K}_{up_1} & \boldsymbol{K}_{up_2} \end{pmatrix} \tag{2.13}$$

and

$$\begin{pmatrix} \boldsymbol{K}_{Tu} \\ \boldsymbol{K}_{p_1 u} \\ \boldsymbol{K}_{p_2 u} \end{pmatrix} \tag{2.14}$$

describe coupling between mechanical behaviour and transport processes and they are assembled in the part METR.

Structure of MEFEL, TRFEL and METR

For each module (MEFEL, TRFEL, METR), the data describing the given problem are split in five large classes.

- `probdesc` - class containing the problem description,
- `gtop` - class containing data describing general topology of finite element mesh,
- `top` - class containing data describing properties of finite element mesh in the given problem type,
- `mat` - class containing data describing used material models,
- `crsec` - class containing data describing cross-sections,
- `bclc` - class containing data describing boundary conditions and loading.

The names of classes differ for particular problems by a pre/postfix created from the problem name abbreviation. The data of these classes are necessary almost everywhere in the code, and this led to making them global objects. Thus, each class has one instance, which is a global variable. This approach reduces the number of passed parameters in functions. In addition to that, each module contains global objects connected with the system matrices and vectors of unknowns.

The class `probdesc` contains attributes describing the solved problem. There is a group of attributes describing the type of problem, computed quantities and solver of the systems of linear equations. Depending on the problem type, the object of class `hdbcontr` is initialized, which controls the storage/restore of individual time steps to/from the disk. In the case of nonlinear or time-dependent problems, `probdesc` initializes also objects of class `timecon` and `nonlinman`. The `timecon` holds data controlling time steps while the `nonlinman` contains control parameters for the Newton-Raphson or arclength methods.

The class `gtop` contains pure topological data connected with the mesh of elements. It mainly contains two arrays of classes `gnode` and `gelem`. The class `gnode` contains nodal coordinates, the array of the number of degrees of freedom (DOF) at particular nodes and arrays of DOF numbers for each node. The class `gelem` contains array of nodal connectivity, number of DOFs on elements and eventually, the array of DOF numbers in the case that they are defined on elements (e.g. definition of joined beam elements). Both classes, `gnode` and `gelem`, also contain data about the hanging nodes and identifiers of time functions that control nodal/element switching on/off for the problems with changing geometry. These data are initialized if they are used in the given problem only.

The class `top` contains data about finite element mesh connected with the given problem. There are two main arrays of objects of classes `node` and `element`. The class `node` holds data about the nodal local coordinate systems and the type of nodal cross-sections. There are also arrays of the resulting nodal reactions, strains, stresses, fluxes, gradients and other state variables of the material model used. The class `element` holds data about the finite element type (identifies the element shape and approximation functions), element cross-section, material model type and the arrays of integration point identifiers used for the numerical integration of quantities on individual elements. If the problem solved contains the changing geometry in time, there is initialized array of initial values of primary unknowns (e.g. displacements). Additionally, the `top` contains arrays of adjacent nodes, elements and distances of integration points which may be used, e.g. in the non-local material models.

The array of objects of `intpoints` is the most important data member of the class `mat`. The class `intpoints` contains state variables computed in the particular integration points, such as strains, stresses, fluxes, gradients and other state values. There are also arrays of initial conditions for integrations points, an array of values of unknowns from coupled problems, etc. For example, in the mechanical part, `mat` class contains arrays of temperature and moisture values at integration points. The `mat` class also holds arrays of objects of supported material types (i.e. implemented material models). Each material type has one object per set of material parameters. These instances of material models are referenced from elements and integration points by the corresponding indices.

The class `crsec` contains arrays of objects for particular cross-section types. There are also methods for retrieving basic cross-section parameters such as thickness or area. These cross-section properties are referenced from nodes and elements.

The `bclc` class collects data about boundary conditions that are arranged in particular load cases. Several load cases can be defined in time-independent and also in time-dependent problems. Every load case can contain several sub-load cases due to better control of the time-dependent load. The boundary conditions can be specified for the given load case at nodes, elements, edges and surfaces. Thus, `bclc` class contains the array of objects of the `loadcase` class, in which the boundary conditions are stored, an array of initial conditions and several auxiliary data members. The `bclc` class has only several methods for data manipulation, and most of the functionality provides the `loadcase` class.

2.3.3 Extensibility with respect to mass transport problem

Contrary to the mechanical problems and heat transfer problems, where the description with the help of displacement and temperature vectors is well established, there is no common general model for mass transport in porous media. The primary unknowns are selected with respect to the number of phenomena captured in the material behaviour, and also accessibility of material parameters plays a significant role. That was the reason for increased demands on the extensibility of the transport part of the code TRFEL and METR. The coupled problems with many variables can be introduced in the transport part of the code with the help of the system of row and column indices. It can be documented on the assembling of conductivity matrix block (2.12), where three primary unknowns are used in the model. These unknowns are temperature T, pore pressure p_1

```
1   for (i=0;i<ntm;i++){
2     for (j=0;j<ntm;j++){
3       conductivity_matrix (i,eid,i,j,lkm);
4       codnum (rcn,i);
5       codnum (ccn,j);
6       mat_localize (km,lkm,rcn,ccn);
7     }
8   }
```

Table 2.1: Assembling of the conductivity matrix.

and pore pressure p_2.

Table 2.1 shows a part of the code that computes and assembles a finite element's conductivity matrix. `ntm` denotes the number of primary unknowns. In the case of matrix (2.12), `ntm`=3. Third row of Table 2.1 represents subroutine which computes a submatrix defined by Equation (2.2). The matrix is stored in `lkm`. Appropriate row and column indices are obtained by the subroutine `codnum` (lines 4 and 5) and they are stored in `rcn` and `ccn`. The submatrix (2.2) is localized to the conductivity matrix of a finite element denoted `km`. Later on, at the global level of conductivity matrix assembling, the element matrix is added to the matrix $\boldsymbol{K}$ of the system (2.1).

Chapter 3

Using new C++ standard extensions in the finite element code design

3.1 Introduction

The design of the SIFEL code favors composition over inheritance, and therefore the source code contains a number of functions with `switch` statements that are intended for computations on individual elements and material points. Traditional `switch` statements are easy to understand even for less experienced programmers and have better performance than dispatching calls to virtual functions, as will be shown later with an example. This C-style design also does not lead to memory fragmentation due to arrays of pointers to base class type objects. All of these advantages lead to good code performance, but there are some problematic aspects of this design.

One of the known major drawbacks can be observed when a new element or material point is introduced into the system. The user has to add new cases to the `switch` statements, and if any of them are missing, an error may be reported at runtime only contrary to the concept of abstract base class where the compiler reports error if some of the pure virtual function would not be implemented.

Also, code maintenance in the case of using inheritance is usually easier. For example, errors caused by changing the number of parameters or their types in the base class methods are clearly reported by the compiler and the number of source code changes can be significantly less than the case with composition design approach. Hence, the inheritance leads to performance hit at runtime while the composition results to 'performance hit' at the code maintenance.

New C++11/14 standard extensions can be exploited for a compromising code design concept inspired by interfaces in C# and Java. The concept should preserve performance of the procedural/composition style with arrays of objects (not pointers to objects) and reduce maintenance costs by elimination of `switch` statements and by the reporting of interface incompleteness at compile time. The key new components for the interface implementation are function templates, *metaprogramming* techniques, `std::tuple` class, `std::function` class and *lambda expressions*.

The concept can be demonstrated by implementing such an interface for general class `Element`. The class `Element` represents one general finite element of mesh with common properties, while the interface provides an implementation of various FE types (bars,

triangles, quadrilaterals, tetrahedrons, and hexahedrons). Typically, the interface should provide a call to the appropriate function to calculate the stiffness matrix or internal force vector, which varies according to the formulation of the given finite element, e.g. bar with linear approximation or quadrilateral element with quadratic approximation. In the composition/procedural style, the interface comprises functions with `switch` statements over implemented FE types where the `switch` statement is typically controlled by the element type identifier `et`.

Data about the element's geometry is stored in the general element class. The class methods that provide calculations for a given FE type get data through their arguments or by accessing global variables. Thus only one instance of the given FE type class is needed. In the new concept, these particular element type objects are stored in the class derived from class template *std::tuple*, representing a fixed-size collection of heterogeneous values whose size must be known at compile time.

3.2 FE interface - the direct approach

The interface of obligatory defined functions on individual FE types can be defined with the help of the class whose initialization is provided by a constructor template. Assuming the general `Element` class holds all runtime-defined element data, the interface class can be defined by the code in Table 3.1. The definition of class `IElemDef` starts on line 2.

```
1   enum elemtype {noelem=0, linbar=1, lintr=2, linquad=3};
2   class IElemDef {
3    public:
4     // template of constructor intializing interface
5     // from element type T
6     template<class T> IElemDef(T& t) :
7     et{t.et},
8     get_ndofe{T::get_ndofe},
9     stiff_matrix{T::stiff_matrix},
10    int_forces{T::int_forces}
11    {}; // constructor body
12
13    // element type
14    elemtype et;
15
16    // returns the number of DOFs on element
17    int (*get_ndofe)();
18
19    // computes stiffness matrix
20    void (*stiff_matrix)
21         (matrix &sm, int id, gtop &g, top &t, crsec &c, mat &m);
22
23    // computes internal force vector
24    void (*int_forces)
25         (vector &ifor, int id, gtop &g, top &t, crsec &c, mat &m);
26  };
```

Table 3.1: Template of element interface - the direct approach.

The class contains four data members defined on lines 14, 17, 20 and 24.

Line 14 defines FE type identifier whose values are given by enumeration `elemtype` defined on line 1. Lines 17, 20 and 24 represents interface functions that should be defined for every FE type. Typically, the interface function needs for the calculation data about topology, materials and cross-sections, which are stored in mentioned global structures `top`, `gtop`, `crsec` and `mat`. These data can be passed to the function by arguments or stored in the globally accessible variables. Access to the element data in these structures is realized with the help of element index `id`. All remaining data are considered to be constant for the given FE element type, and they are defined in the corresponding class of FE, e.g. `barlin`, `trlin` or `quadlin`. This is important because in such a case, all data members may be defined as `static` constants, and all methods in FE classes may be of `static` kind. A static method can be called without an instance of the given class, and a pointer to a static method is defined. Thus due to constantness of the FE classes, the interface functions can be stored directly in the form of pointer to function, see data member type on lines 17, 20 and 24.

The key part of the class is the constructor template `IElemDef(T& t)` whose definition starts on line 6. It is defined with one template parameter `T`, which is supposed to be instantiated by the FE class type. The initializer list of the constructor on lines 7–10 initializes all data members, where pointers to the interface functions (lines 8–10) are initialized directly by the static methods of the given FE element class type `T`, e.g. `T::get_ndofe`.

The simplified structure of a general element class is illustrated in Table 3.2. The

```
1   class Element{
2    public:
3    // default constructor
4    Element() : t{noelem}, aux[0]{'\0'}, defifc{nullptr} {};
5    // reading element data from file in (et, aux)
6    int read(FILE *in);
7    // FE type identifier
8    elemtype et;
9    // member aux substitutes all element data defined at runtime
10   char aux[184];
11   // pointer to interface class
12   const IElemDef *defifc;
13  };
```

Table 3.2: Class of general element with interface.

`Element` class is intended to be instantiated within an array by the default constructor on line 4. Pointer to the `IElemDef` class `defifc` is initialized with null value. Initialization on the appropriate pointer to `IElemDef` object is supposed to be performed after the reading of element data from an input file when the proper FE type identifier `et` is known (line 8). Instances of `IElemDef` class for particular FE types are considered to be stored in an static array to access during the initialization of `Element` class member `et`.

For convenience, classes of all FE types are collected in the `std::tuple` object, which allows for the traversing of particular FE types by the index known at compile time. A new FE type is just added to that collection; thus, no other common code is touched

by implementing that new FE type. The tuple object is defined at line 2 of Table 3.3. It is initialized by the class names of particular FE types `LinBarElem`, `LinTrElem` and `LinQuadElem`. The number of FE types in `elobjs` is known at compile time, and

```
1   // tuple of particular FE classes
2   static constexpr std::tuple
3          <LinBarElem, LinTrElem, LinQuadElem> elobjs;
4
5   // the total number of FE type classes
6   static constexpr int nelemt =
7          std::tuple_size<decltype(elobjs)>::value;
8
9   // type name alias for the array of FE type interface
10  using eldefifc = std::array<IElemDef, nelemt>;
11
12  // array of FE interface objects
13  const static eldefifc eldefifcarray{eldefifcfiller(elobjs)};
14
15  // comparison operator needed in std::find applied to eldefifcarray
16  inline bool operator==(const IElementDefault &lhs,
17                         const elemtype &rhs){
18    return lhs.et == rhs;
19  }
```

Table 3.3: Storage of interface objects in `std::tuple`, array of FE interface objects.

therefore it is retrieved as the compile time constant at line 6. Such constants can be used in the definition of a static array as exploited further in the code. The static array of the `IElemDef` objects is defined at lines 10 and 13. It is initialized with the particular FE types by a metaprogramming technique, which is represented by the set of function templates `eldefifcfiller`. These function templates are instantiated recursively for particular FE types stored at the tuple `elobjs`. Finally, they return a static array initialized by an initializer list composed from objects of `IElemDef` interfaces. Thus the constructor call of static array at line 13 of Table 3.3 will be expanded by the compiler as indicated in Table 3.4 where, e.g., line 3 represents object of FE interface constructed

```
1   const static eldefifc eldefifcarray{
2                  std::array{
3                    IElemDef(LinBarElem),
4                    IElemDef(LinTrElem),
5                    IElemDef(LinQuadElem)
6                   }
7                 };
```

Table 3.4: Initialization of array of FE interfaces.

from the FE type class `LinBarElem`. More details are given further in Table 3.5.

Initialization phase of individual objects of `Element` class requires an access to components of array `eldefifcarray` in order to obtain interface object corresponding to some FE type identifier `Element.et`. STL algorithm template `std::find` can be

used for the searching of the interface object with the given FE type identifier in array `eldefifc`. Algorithm `std::find` requires the operator `==(IElemDef, elemtype)` whose definition is given at lines 16–19 of Table 3.3.

Table 3.5 summarizes the code of the function templates used for the filling of static array type of `eldefifc`. The first function template has value argument `N` whose de-

```
1   // recursively called function template for all FE type objects
2   // stored in elobjs tuple except the first one
3   template <size_t N=nelemt, typename... Tn>
4   constexpr typename std::enable_if< 0<N, eldefifc>::type
5   eldefifcfiller(decltype(eobjs) &elems, const Tn& ...rest)
6   {
7       return eldefifcfiller<N-1>
8     (elems,
9      IElemDef(std::get<N-1>(elems)),
10     rest...);
11  }
12
13  // function template for the first FE type object
14  // in elobjs tuple that stops recursion and returns
15  // the resulting array of interface objects
16  template <size_t N, typename... Tn>
17  constexpr typename std::enable_if<N==0, eldefifc>::type
18  eldefifcfiller(decltype(eobjs) &elems, const Tn& ...rest)
19  {
20    return eldefifc{rest...};
21  }
```

Table 3.5: Function templates for the filling of a static array type of `eldefifc`.

fault value is the number of FE types, i.e. `nelemt`, see line 3. This value argument is followed by the type argument pack denoted by `Tn`. The function template can be instantiated if the value argument `N` is greater than zero. This is provided by the standard metafunction `std::enable_if` which defines the return type of function to be `eldefifc` type if the condition `N` > 0 is met, see line 4. The first argument of function itself (`elems`) is type of tuple of FE types (`decltype(eobjs)`) while the number of remaining function arguments is variable and they are defined with the help of parameter type pack `Tn ...rest`, see line 5. The function body contains just one return statement (line 7), which returns the result of the instance of the same template function (template recursion), but it decrements the value argument `N` of the template (line 7 `eldefifcfiller<N-1>`). Moreover to the first function argument `elems`, it adds one new object of the FE interface class `IElemDef` in the argument list, which is constructed from the FE type class of the (`N-1`)-th object stored in the tuple `elems`, see line 9. Finally, the remaining argument pack denoted as `rest` is passed at the end of function argument list (line 10).

The recursive call of instances of the first function template is stopped by the second function template, which represents specialization for the value argument $N == 0$. If the recursion reaches this template specialization, then the function argument list `Tn ...rest` contains objects of interfaces created for all FE types collected in the tuple

`eobjs`. Thus the static array can be initialized by the initializer list of objects in `rest` and the resulting static array is returned from function at line 20 (cf. with Table 3.4).

Every FE type is represented by the dedicated class, which contains the implementation of the FEM computational interface. Table 3.6–3.8 contains a reduced form of class definition for the bar element with linear approximation `LinBarElem`, triangular element with linear approximation `LinTrElem` and quadrilateral element with linear approximation `LinQuadElem`.

```
1   class LinBarElem{
2    public:
3     static constexpr int ncomp = 1; // number of strain components
4     static constexpr int ndofe = 4; // number of DOFs on element
5     static constexpr elemtype et = linbar;
6     static constexpr int get_ndofe() {return ndofe;};
7     static void stiff_matrix (matrix &sm, int id, gtop &g,
8                               mtop &t, mcrsec &c, mmat &m){
9       // code for the computing \int_V B^T D B dV
10    };
11    static void int_forces (vector &ifor, int id, gtop &g,
12                                mtop &t, mcrsec &c, mmat &m){
13      // code for the computing \int_V B^T \sigma dV
14    };
15  };
```

Table 3.6: Definition of class `LinBarElem`.

```
1   class LinTrElem{
2    public:
3     static constexpr int ncomp = 3; // number of strain components
4     static constexpr int ndofe = 6; // number of DOFs on element
5     static constexpr elemtype et = lintr;
6     static constexpr int get_ndofe() {return ndofe;};
7     static void stiff_matrix (matrix &sm, int id, gtop &g,
8                               mtop &t, mcrsec &c, mmat &m){
9       // code for the computing \int_V B^T D B dV
10    };
11    static void int_forces (vector &ifor, int id, gtop &g,
12                                mtop &t, mcrsec &c, mmat &m){
13      // code for the computing \int_V B^T \sigma dV
14    };
15  };
```

Table 3.7: Definition of class `LinTrElem`.

The runtime variable data of particular FE elements are collected in the general element class `Element` and thus the content of classes can be specified to be **`static`** or **`static constexpr`**. Data members and methods with such specifications may be used/called without an instance of the given class.

Finally, Table 3.9 represents a code excerpt of the global computational procedure where the usage of the developed concept is being demonstrated. There is the initialization

```
1   class LinQuadElem{
2    public:
3     static constexpr int ncomp = 3; // number of strain components
4     static constexpr int ndofe = 8; // number of DOFs on element
5     static constexpr elemtype et = linquad;
6     static constexpr int get_ndofe() {return ndofe;};
7     static void stiff_matrix (matrix &sm, int id, gtop &g,
8                               mtop &t, mcrsec &c, mmat &m){
9       // code for the computing \int_V B^T D B dV
10    };
11    static void int_forces (vector &ifor, int id, gtop &g,
12                                  mtop &t, mcrsec &c, mmat &m){
13      // code for the computing \int_V B^T \sigma dV
14    };
15  };
```

Table 3.8: Definition of class `LinQuadElem`.

of `Element` class member `defifc` with the corresponding pointer to the FE interface object and usage of the interface for assembling the global stiffness matrix. There are instances of the mentioned five large classes at lines 3–7. Data members of these classes are read from the input file `in` at lines 14–17. Class `mtop` contains data member `elems` that represents array of `Element` class instances. These instances have their `et` members set to the proper FE type identifier after the call of `mt.read` at line 15 and thus their `defifc` member can intialized. Apropriate FE interface object can be found at the static array `eldefifc` (Table 3.3 line 13) with respect to the FE type identifier with the help of standard function `std::find` at line 22. This function requires specifying the range of the lookup, which is defined in terms of iterators of `eldefifc`. In this case, the range is set across the whole array by the start iterator `eldefifc.begin()` (line 22) and final iterator `eldefifc.end()`(line 23). The searched value is the last parameter of `std::find` where the FE type identifier of the i-th element is passed by `mt.elems[i].et`. The `std::find` returns an iterator to the found element or position one past the last array element if nothing is found. The index of FE interface object `j` can be calculated as a difference between `std::find` result and the beginning of array (line 25). Finally, after the check of interface index at line 26, the `defifc` of i-th element is being initialized with the pointer to the j-th interface at line 27.

3.3 FE interface - the function wrapper approach

The FE interface proposed in Table 3.2 is defined in terms of direct pointers to function and therefore FE type classes `LinBarElem`, `LinTrElem` and `LinQuadElem` must define the implementation of interface functions to be static. The static functions must not access runtime non-static data members in the given class. That restriction can be avoided by the data decomposition approach, like in the case of the general `Element` class, which collects all runtime-defined data. In this case, the static function specification requirement is not too restrictive because most data directly connected with the given FE type and used in the computational procedures (number of nodes, number of DOFs,

```
1   // ... other code
2   // instances of main classes
3   mprobdesc  mp;
4   gtop       gt;
5   mtop       mt;
6   mcrsec     mc;
7   mmat       mm;
8   // input file pointer
9   FILE      *in;
10  // ... other code
11
12  // read input data from file referenced by in
13  // (problem description, mesh topology, cross-sections, materials)
14  mp.read(in);
15  mt.read(in, gt);// array of Element class is stored in mt.elems
16  mc.read(in);
17  mm.read(in);
18  // ... other code
19
20  // initialize element interface
21  for (i=0; i<mt.ne; i++){
22    size_t j = std::find(eldefifcarray.begin(),
23                         eldefifcarray.end(),
24                         mt.elems[i].et)
25               - eldefifcarray.begin();
26    assert(j<nelemt);
27    mt.elems[i].defifc = &eldefifcarray[j];
28  }
29  // ... other code
30
31  // Assembling of the system stiffness matrix
32  skyline smat(gt); // system stiffness matrix
33  matrix lsm;       // element stiffness matrix
34  ivector eldofn;   // vector of DOF numbers at one element
35
36  smat.null();
37  for (int i=0; i<mt.ne; i++){
38    int ndofe = mt.elems[i].defifc->get_ndofe();
39    lsm.realloc(ndofe, ndofe);
40    eldofn.realloc(ndofe);
41    gt.gelems[i].give_eldof(eldofn);
42    mt.elems[i].defifc->stiff_matrix(sm, i, gt, mt, mc, mm);
43    smat.localize(eldofn, lsm); // localization of lsm matrix
44  }
```

Table 3.9: Setup and usage of FE interfaces.

integration point coordinates, ...) are known at compile time. Thus they can be defined as constants allowing access by static methods. On the other hand, if some setting of FE type is needed at runtime, e.g. order of numerical integration, a different representation of interface procedures has to be used. Ordinary methods of class access data members with the help of `this` pointer which refers to the given class instance for which the method is being called. Pointer `this` is a hidden argument passed to the class's methods which

is an implementation detail of compilers, and therefore, the methods can only be called within the scope of the class instance. It also implies that the pointer to the ordinary method cannot be obtained.

A solution to that problem can be a general-purpose polymorphic function wrapper represented by the class `std::function`, which is the part of the standard template library (STL). It allows for the uniform handling of callable targets – functions (via pointers to that), lambda expressions, bind expressions, or other function objects, and pointers to member functions. The modified FE interface class `IElemDefM` is captured in Table 3.10. There are two key differences compared to the original class definition. The first

```
1   class IElemDefM {
2    public:
3     // template of constructor intializing interface
4     // from element type T
5     template<class T> IElemDefM(T& t) :
6     et{t.et},
7     get_ndofe{[&t]()->int {return t.get_ndofe();}},
8     stiff_matrix{[&t](matrix &sm, int id, gtop &g
9                       top &mt, crsec &c, mat &m)->void
10                {return t.stiff_matrix(sm, id, g, mt, c, m);}},
11    int_forces{[&t](vector &ifor int id, gtop &g,
12                    top &mt, crsec &c, mat &m)->void
13                {return t.int_forces(ifor, id, g, mt, c, m);}}
14    {}; // constructor body
15    // element type
16    elemtype et;
17
18    // returns the number of DOFs on element
19    std::function<int ()> get_ndofe;
20
21    // computes stiffness matrix
22    std::function<void (matrix &sm, int id, gtop &g,
23                        top &t, crsec &c, mat &m)> stiff_matrix;
24
25    // computes internal force vector
26    std::function<void (vector &ifor, int id, gtop &g,
27                        top &t, crsec &c, mat &m)> int_forces;
28  };
```

Table 3.10: Modified FE interfaces - `std::function`.

one consists in the changing type of interface data members `get_ndofe`, `stiff_matrix` and `int_forces` which represents computational procedures. They are now created by instances of `std::function` template which are instantiated by the return type and list of parameters, see lines 19, 22 and 26. The second key point is the initialization of these data members at the constructor template, which starts at line 5. The constructor has one parameter `t` (line 5) of type `T`, i.e. instance of FE type class. This instance is needed for calling its methods and must be somehow stored in the instance of `std::function` class. Particular data members are initialized in the constructor initializer list (lines 6–13) while the constructor body remains empty (line 14). Data members `get_ndofe`, `stiff_matrix` and `int_forces` are initialised with the help of lambda

expressions which is a kind of in-place definition of short functions. The usage can be demonstrated on data member `stiff_matrix` where `[&t](matrix &sm, ...)->void` represents start of definition of lambda expression which captures the object `t` by reference from the actual scope, i.e. constructor. The captured outter variables can be used in the lambda expression body as if they were defined in its scope. The part `(matrix &sm, ...)` represents the list of parameters used in the lambda expression call. Finally, `->void` specifies the lambda expression return type to be `void`. The body of the lambda expression consists just of the return statement where the appropriate function `stiff_matrix` is being invoked for the given object `t` of the constructor parameter (line 10).

The remaining parts of the interface can be defined in the same way as in Section 3.2 except FE type classes that may involve non-static data members. For example, class `LinBarElem` might define `ndofe` to be ordinary member (Table 3.11, line 4) which can be set later according to some specification of the 2D or 3D case of the bar element at runtime.

```
1   class LinBarElem{
2    public:
3     static constexpr int ncomp = 1;
4     int ndofe;
5     static constexpr elemtype et = linbar;
6     LinBarElem : ndofe{0} {};
7     int get_ndofe() {return ndofe;};
8     void stiff_matrix (matrix &sm, int id, gtop &g,
9     mtop &t, mcrsec &c, mmat &m){
10      // code for the evaluation of \int_V B^T D B dV
11    };
12    void int_forces (vector &ifor, int id, gtop &g,
13    mtop &t, mcrsec &c, mmat &m){
14      // code for the evaluation of \int_V B^T \sigma dV
15    };
16  };
```

Table 3.11: Definition of modified class `LinBarElem` with ordinary member `ndofe`.

Setup of `LinBarElem.ndofe` at runtime is captured in Table 3.12, where the type `LinBarElem` is passed to the `std::get` function which returns the given FE type object from the tuple `elobjs` and thus its `ndofe` member can be accessed. The setup of `ndofe` member is performed with respect to member `probdim` of `probdesc` class which is determined according to data in the input file, see lines 9–12.

3.4 Comparison of particular approaches

Proposed approaches which involve advanced features of C++ were compared with traditional approaches based on procedural programming and OOP inheritance.

```
1   // read input data from file referenced by in
2   // (problem description, mesh topology, cross-sections, materials)
3   mp.read(in);
4   mt.read(in, gt); // array of Element class is stored in mt
5   mc.read(in);
6   mm.read(in);
7   // ... other code
8
9   if (mp.probdim == 3)  // 3D problem
10    std::get<LinBarElem>(elobjs).ndofe = 6;
11  else                  // 2D problem
12    std::get<LinBarElem>(elobjs).ndofe = 4;
13
14  // initialize element interface
15  for (i=0; i<mt.ne; i++){
16    // the same code as the original
17  }
```

Table 3.12: Setup and usage of modified FE interfaces.

3.4.1 The procedural approach

In the procedural approach, there are procedures with `switch` statements where the appropriate procedure from FE type classes is called according to the FE type member `et` of the general FE class `Element`. Tables 3.13, 3.15 and 3.15 summarizes the code of interface functions `get_ndofe`, `stiff_matrix` and `int_forces`.

```
1   int get_ndofe(elemtype et) {
2     switch (et){
3       case linbar:
4         return LinBarElem::ndofe;
5       case lintr:
6         return LinTrElem::ndofe;
7       case linquad:
8         return LinQuadElem::ndofe;
9       default:
10        fprintf(stderr, "Unknown type of element.\n");
11        exit(1);
12    }
13    return 0;
14  }
```

Table 3.13: The procedural approach - `get_ndofe` interface function.

General FE class `Element` no more contains the interface member `ifcdef`, and the usage is demonstrated in Table 3.16. Again, there is a loop for assembling the system stiffness matrix at lines 26–34, which iterates over all elements. All interface functions need as the first argument FE type alias for the given `i`-th element, and therefore it is retrieved from the general element `elem[i]` of class `mtop` at line 27 and stored in the local variable `et`. The number of element DOFs is obtained by the call of `get_ndofe` function at line 27

```
1   void stiff_matrix(elemtype et, matrix &sm,  int id, gtop &g,
2                     mtop &t, mcrsec &c, mmat &m){
3     switch (et){
4       case linbar:
5         return LinBarElem::stiff_matrix(sm, id, g, t, c, m);
6       case lintr:
7         return LinTrElem::stiff_matrix(sm, id, g, t, c, m);
8       case linquad:
9         return LinQuadElem::stiff_matrix(sm, id, g, t, c, m);
10      default:
11        fprintf(stderr, "\nUnknown␣type␣of␣element.\n");
12        exit(1);
13    }
14    return;
15  }
```

Table 3.14: The procedural approach - `stiff_matrix` interface function.

```
1  void int_forces(elemtype et, vector &ifor,  int id, gtop &g,
2                  mtop &t, mcrsec &c, mmat &m) {
3     switch (et){
4       case linbar:
5         return LinBarElem::int_forces(ifor, id, q, t, c, m);
6       case lintr:
7         return LinTrElem::int_forces(ifor, id, q, t, c, m);
8       case linquad:
9         return LinQuadElem::int_forces(ifor, id, q, t, c, m);
10      default:
11        fprintf(stderr, "\nUnknown␣type␣of␣element.\n");
12        exit(1);
13    }
14    return;
15  }
```

Table 3.15: The procedural approach - `int_forces` interface function.

and `i`-the element stiffness matrix is computed by the call of `stff_matrix` function at line 32.

3.4.2 The OOP inheritance and polymorphism approach

The interface is defined in the base class, which represents the general FE type and corresponds to the class `Element`. The base class defines required interface methods as *pure virtual methods* while optional interface methods should also be virtual with a default implementation. An example of the base class design can be seen in Table 3.17. FE interface methods `get_ndofe()`, `stiff_matrix(...)` and `int_forces(...)` are defined to be pure virtual which is specified by the zero assignment after the method declaration, see lines 10–12. A class with pure virtual methods is called an abstract class and cannot be instantiated directly; only a pointer is allowed to that class. Classes derived from the abstract base class must provide their own implementation of pure virtual

```
1   // ... other code
2   // instances of main classes
3   mprobdesc  mp;
4   gtop       gt;
5   mtop       mt;
6   mcrsec     mc;
7   mmat       mm;
8   // input file pointer
9   FILE      *in;
10  // ... other code
11
12  // read input data from file referenced by in
13  // (problem description, mesh topology, cross-sections, materials)
14  mp.read(in);
15  mt.read(in, gt); // array of Element class is stored in mt
16  mc.read(in);
17  mm.read(in);
18  // ... other code
19
20  // Assembling of the system stiffness matrix
21  skyline smat(gt); // system stiffness matrix
22  matrix lsm;       // element stiffness matrix
23  ivector eldofn;   // vector of DOF numbers at one element
24
25  smat.null();
26  for (int i=0; i<mt.ne; i++){
27    elemtype et = mt.elems[i].et;
28    int ndofe = get_ndofe(et);
29    lsm.realloc(ndofe, ndofe); // memory alloc. of elem. stiff. mat.
30    eldofn.realloc(ndofe);     // memory alloc. of DOF number array
31    gt.gelems[i].give_eldof(eldofn);
32    stiff_matrix(et, sm, i, gt, mt, mc, mm);
33    smat.localize(eldofn, lsm);
34  }
```

Table 3.16: Setup and usage of the procedural approach.

functions otherwise the compiler would report an error. An example of the implementation of `LinQuadElem` class is given in Table 3.18. The class implements a quadrilateral element with linear approximation functions. A public inheritance from the base class `Element` is specified at line 1 while the specific implementations of interface methods are declared at lines 6, 7 and 10. The other FE classes `LinBarElem` and `LinTrElem` are defined similarly.

A computational code which uses the inheritance approach is given in Table 3.19, which depicts a code excerpt for the assembling of the system stiffness matrix.

```
1   struct Element{
2    public:
3     elemtype et;
4     int nn;
5     char aux[184];
6     Element() : et{elemtype(0)}, nn{0} {};
7     virtual void get_eldof(ivector &ed) {
8       // store element DOF in the argument ed
9     };
10    virtual int get_ndofe() const = 0;
11    virtual void stiff_matrix(matrix &sm) const = 0;
12    virtual void int_forces(vector &ifor) const = 0;
13  };
```

Table 3.17: OOP inheritance approach - base class of general FE.

```
1   class LinQuadElem : public Element{
2    public:
3     static constexpr int ncomp = 3;
4     static constexpr int ndofe = 8;
5     static constexpr elemtype et = linquad;
6     virtual int get_ndofe() const {return ndofe;};
7     virtual void stiffmat(matrix &sm) const{
8       // implementation of sm =  B^T. D. B
9     };
10    virtual void int_forces(vector &ifor) const{
11      // implementation of sm =  B^T. sigma
12    };
13  };
```

Table 3.18: OOP inheritance approach - derived class `LinQuadElem`.

```
1
2   // reading number of elements ne
3
4   Element **elems = new (Element*)[ne];
5
6
7   for (i=0; i<ne; i++){
8     //
9     // reading of element type elems[i].et from file
10    //
11    switch (elems[i]->et){
12      case linbar:
13        elems[i] = new LinBarElem;
14        break;
15      case lintr:
16        elems[i] = new LinTrElem;
17        break;
18      case linquad:
19        elems[i] = new LinQuadElem;
20        break;
21      default:
22        return 1;
23    }
24    // reading of remaing element property
25    elems[i]->read(in);
26  }
27
28  // other code
29
30  // Assembling of the system stiffness matrix
31  skyline smat(gt); // system stiffness matrix
32  matrix lsm;       // element stiffness matrix
33  ivector eldofn;   // vector of DOF numbers at one element
34
35  smat.null();
36  for (int i=0; i<ne; i++){
37    int ndofe = elems[i]->get_ndofe();
38    sm.realloc(ndofe, ndofe);
39    eldofn.realloc(ndofe);
40    elems[i]->give_eldof(gt, eldofn);
41    elems[i]->stiff_matrix(lsm);
42    smat.localize(eldofn, lsm);
43  }
```

Table 3.19: Setup and usage of OOP interface procedures.

3.4.3 Performance comparison

The performance of particular approaches was compared through the repeated of the stiffness matrix assembling and computation of the internal force vector. The following setups of particular approaches were tested:

1. procedural approach with `switch` statements
   (a) set of 50,000 FE with randomly selected type from the set `LinBarElem`, `LinTrElem` and `LinQuadElem`,
   (b) set of 50,000 LinQuadElem elements with linear approximation.
2. OOP inheritance approach
   (a) set of 50,000 FE with randomly selected type from the set `LinBarElem`, `LinTrElem` and `LinQuadElem`,
   (b) set of 50,000 LinQuadElem elements with linear approximation.
3. Interface metaprogramming - a direct approach
   (a) set of 50,000 FE with randomly selected type from the set `LinBarElem`, `LinTrElem` and `LinQuadElem`,
   (b) set of 50,000 LinQuadElem elements with linear approximation.
4. Interface metaprogramming - function wrapper approach
   (a) set of 50,000 FE with randomly selected type from the set `LinBarElem`, `LinTrElem` and `LinQuadElem`,
   (b) set of 50,000 LinQuadElem elements with linear approximation.

For all mentioned sets of finite elements, assembling the system stiffness matrix and internal force vector was repeated 100 times to simulate some nonlinear analysis. For the illustration, every local stiffness matrix was localized in the system stiffness matrix, represented by a vector of $50{\cdot}10^6$ `doubles` simulating skyline storage. DOF numbers were generated on particular elements to cover access to all skyline elements. The source code of the `main` procedures of the benchmark tests can be found in Appendix A. Benchmark versions with random element type are listed in Tables A.6-A.9. The version for `LinQuadElem` element type can be obtained simply by the modification of line 7, where `readarray[i] = linquad` should substitute random element type generation.

The resulting averaged times, minimum, maximum and median times from 20 runs of the different approaches are summarized in the Table 3.20. They were obtained on the desktop computer with six-core Intel i5-9500 @ 3 GHz processor with 16 GB of RAM. Elapsed times in Table 3.20 show that differences are not too significant; however, the procedural approach performs best for both sets of elements, while the performance of OOP inheritance was somewhat lower. The proposed approaches with interfaces generated by metaprogramming techniques had slightly better performance in most tests than the OOP inheritance approach except for the test on the uniform set of `LinQuadElem` elements. The differences in the performance stem from the following aspects of particular approaches:

Time [*s*]	Proc. random 1(a)	Proc. `LinQ` 1(b)	OOP inherit. random 2(a)	OOP inherit. `LinQ` 2(b)	Iface direct random 3(a)	Iface direct `LinQ` 3(b)	Iface f. wrap. random 4(a)	Iface f. wrap. `LinQ` 4(b)
Avg	4.05	7.91	4.48	8.80	4.39	8.73	4.08	8.97
Min	4.03	7.87	4.25	8.38	4.31	8.51	3.93	8.74
Max	4.09	8.14	5.01	9.49	4.88	9.30	4.56	9.52
Med	4.05	7.88	4.29	8.41	4.33	8.58	3.96	8.79

Table 3.20: Elapsed times in [s] for particular programming approaches.

- OOP inheritance approach requires the elements to be stored as an array of pointers to the dynamically allocated objects. Thus the access requires dereferencing, and it has a worse memory locality of the element data, which may result in higher cache memory misses. The remaining approaches allow for the usual array of element objects with better memory locality, requiring no dereferencing and therefore, there is a better cache memory performance potentially.
- OOP inheritance and interface approaches have tables of pointers to the virtual/interface functions. Therefore, there is a penalty due to dereferencing table pointer and computation of the pointer to the given interface function. These approaches are not also friendly to the branch prediction performed by the processors.
- Procedural approach uses `switch` statements which do not require dereferencing of table pointer. However, the code for the appropriate calling function is also not too friendly to the branch prediction of the processors.

It should be noted that elapsed times are influenced significantly by the actual data cache hits/misses at the runtime, and keeping equal conditions for particular tests is difficult. Elapsed times are similar and should be understood as a rough relative measure for the comparison.

3.4.4 Conclusions

It can be concluded that the proposed function interface approach based template metaprogramming represents a reasonable alternative to the traditional approaches either based on the `switch` statements or OOP inheritance. It allows for the good extensibility of the code because a new type can be just placed in at the `std::tuple` object in one header file. This is similar to the factory design pattern, which can be implemented with the help of abstract classes and inheritance. It also preserves good performance because objects with an interface can be stored in a usual array. There are also weak points that are related, namely to template metaprogramming, which is an advanced programming technique and requires more experienced programmers. Fortunately, the amount of such code is limited, and ordinary users can be quite effectively spared from such code parts. The other flaw is that the error/warning messages are more difficult to understand in the case of compile time errors.

Chapter 4

Example of problem solved

The program SIFEL was used to analyse several real-world engineering problems. The first real-world problem solved by the code was a detailed THM and limit analysis of a reactor vessel in the nuclear power plant in Hinkley (UK) in order to elongate its service life. The vessel was composed of a cylindrical prestressed concrete structure reinforced by a steel lining at the internal surface. Because of symmetry, only one eighth of the vessel was taken into account. Figure 4.1 shows the mesh of the modelled segment. The hydro-thermo-mechanical time-dependent analysis was performed. The creep of the concrete vessel was modelled by Bazant's B3 model Baweja and Bažant [1995]. In the selected times, the nonlinear limit analyses using the damage model have been performed. Papers describing the problem can be found in [Kruis et al., 2005].

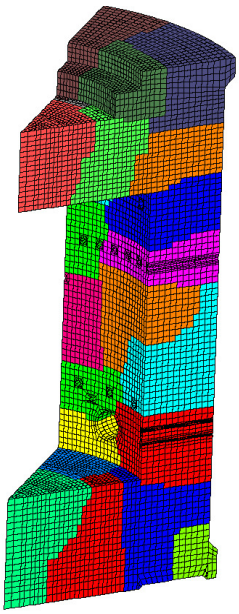


Figure 4.1: Model of reactor vessel segment.

Distribution of damage in the containment wall of the nuclear power plant Temelín (CR) was another complex problem solved by SIFEL. Figure 4.2 captures a cylindrical segment from the containment wall. This problem was described by the coupled thermo-

mechanical analysis. Results from the performed analyses were published in [Koudelka et al., 2009] and [Krejčí et al., 2009].

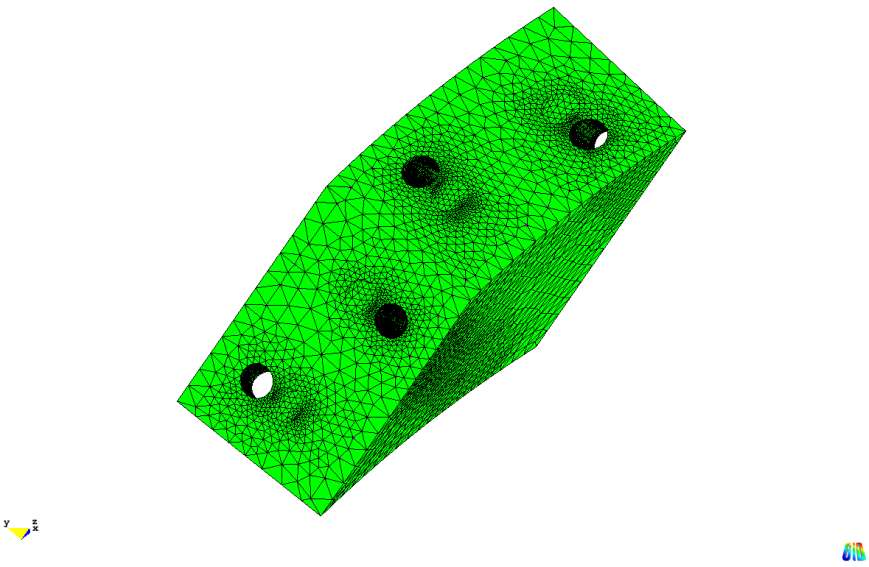


Figure 4.2: Model of segment of containment in the nuclear power plant Temelín (Czech Republic).

Rock slope stability is another problem solved by SIFEL. Figure 4.3 depicts the mesh of the heterogeneous rock clif in Prague. The rock body consists of layered slates with quartzite inlays. The rock body strata crop out of the slope at an angle of 45^o–55^o with skew 0^o–7^o and exhibit polish on strata surfaces. There is a road built in the cut of the slope partially, and part of the road is on a bridge. Several nonlinear limit analyses were performed with the Mohr-Coulomb plasticity model. More details about the problem can be found in [Koudelka and Koudelka, 2005] or [Koudelka and Koudelka, 2007].

Collaboration with Zakládání staveb a.s. within the CIDEAS research project led authors of the program to the solution of impermeable concrete foundation slabs. That was significant impulse for the code extension because a sequential casting procedure of the foundation slabs and hydration heat generation were simulated, and damage evolution in concrete was analysed. Figure 4.4 depicts a model of a foundation slab constructed at two levels. The results from these analyses were published in [Krejčí et al., 2006], [Krejčí et al., 2007] and [Koudelka et al., 2007] and Chapter 6 addressed this problem in detail.

The problem of simulation of the mechanical response of a church influenced by climatic conditions was the aim of the research project No. 18-24867S, supported by the Czech Science Foundation. It was the church of All Saints in Broumov (see Figure 4.5) group of churches. Because the church has walls built from masonry (see Figure 4.6a, the multi-scale modelling of mechanical properties was adopted in this case. The homogenized tensile and compressive strengths were determined with the help of hybrid first-order homogenization techniques. Figure 4.6b captures the crack distribution of one specimen loaded by increasing horizontal macro strain while preserving plane stress conditions, i.e. zero macro stresses in the remaining directions.

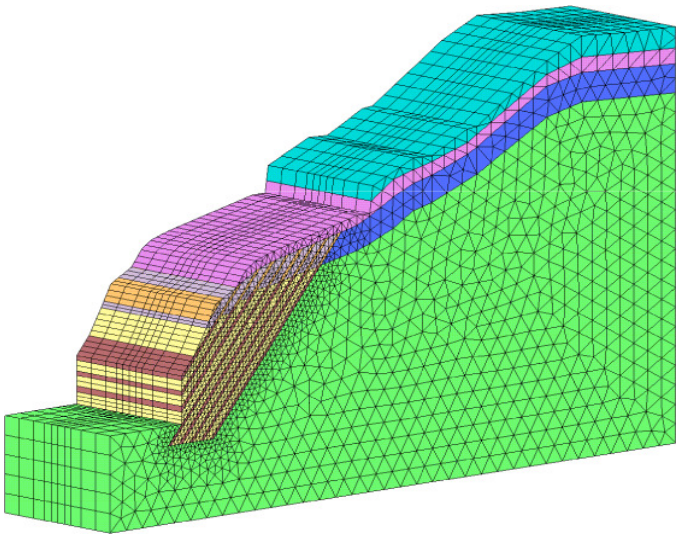
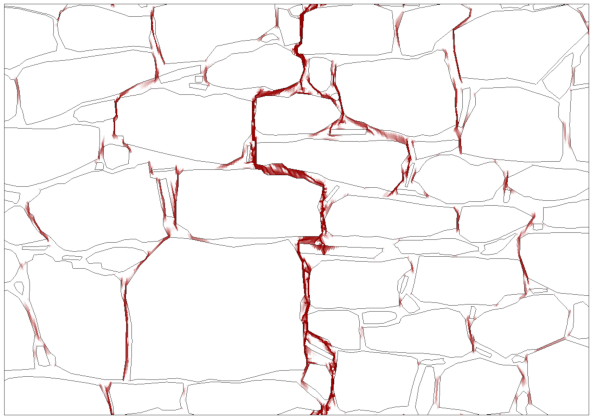


Figure 4.3: Mesh of heterogeneous rock slope in Prague (Czech Republic).

Figure 4.4: Model of the foundation slab in Prague-Těšnov (Czech Republic).

Figure 4.5: The church of All Saints in Broumov.

(a) Sample of masonry wall



(b) Final crack pattern

Figure 4.6: Modelling of church's wall masonry.

Part II

Modelling of construction phases in engineering problems

Chapter 5

Introduction

Construction phases have to be taken into account in the design of many structures. The stress/strain states as well as deformation of structures may differ significantly at particular construction phases from the final state. Typical examples represent construction phases of concrete or steel bridges, excavation of deep trenches in the soils or rocks and building of tunnels.

This part is addressed on the implementation of construction phases in SIFEL code with a special attention to the mechanical problems where the implementation is connected with several issues. The first example consists of the simulation of casting procedure of foundation slab with special attention to the crack evolution due to a non-uniform hydration heat generation.

Several approaches will be demonstrated in connection with the gradual construction of a box girder bridge built using the balanced cantilever construction method with cast-in-place segments supported by form travellers. The aim was to determine the evolution of the bridge deflection with respect to the concrete creep and relaxation of the prestressing tendons.

The remaining techniques will be introduced in connection with the excavation problem of the deep trench, where the soil exhibits highly nonlinear behaviour. The hypoplastic model for soils was used in this case which does not allow for the tensional stress states, and special care had to be paid to the soil block removal.

Chapter 6

Analysis of casting procedure of thick foundation slab

The research project Centre for Integrated Design of Advanced Structures (CIDEAS) was concerned with advanced materials, construction, energy, environment, extreme situations and risk assessment in civil engineering. Three Czech civil engineering faculties (FCE CTU in Prague, FCE Brno UT and FCE TU of Ostrava) collaborated with industrial partners in this project. One of the topics was the analysis of the water impermeability of the foundation slab, which was proposed by the industrial partner Zakládání staveb a.s. The motivation for this research was the construction of the commercial building Diamond Point in Prague near the Vltava river (Figures 6.1 and 6.2). The building was founded on a slab 10 meters under the groundwater level, which led to the increased demands on the water impermeability of the used concrete. The foundation slab also had a significant thickness, and therefore the casting was divided into several phases in order to reduce cracks due to hydration heat.

Figure 6.1: The commercial building Diamond Point.

The foundation of buildings in deep foundation pits is usually under the groundwater level, which in this case, maybe be elevated significantly during floods because of near river Vltava. The potential damage to the foundation slab is difficult to repair, and

Figure 6.2: View of foundation pit at Těšnov

reconstructions are costly. The damage to the slab may be because of cracking arising due to the combination of hydration processes in concrete and contact forces between the foundation slab and the subsoil and may result in the loss of the slab impermeability. Concrete watertightness is influenced by a variety of factors, especially:

- concrete mixture composition,
- degree and form of reinforcement,
- technology of casting procedure,
- arrangement of working gaps,
- proper curing of concrete during hydration.

Watertight concrete is often designed as high-performance, self or easy-compacting where, except the basic material parameters such as water-cement ratio and particular aggregates, an important role also play the admixtures such as superplastifiers and hardening accelerators. These admixtures influence significantly evolution of hydration heat and autogeneous shrinkage whose values are raised when compared to the usual concrete.

These factors are necessary to take into account in computer simulation of slab behavior. The computer simulation should represent

- used casting procedure by particular layers and shrinkage parts (the slab has to be cast in several layers with thickness 300∼600 mm),
- curing of concrete (watering and protection against sun radiation),
- autogeneous shrinkage in early stages,
- drying shrinkage in late stages,
- increasing stiffness and strength of concrete in course of time,
- creep of concrete,
- possible damage of concrete.

All these effects depend on time, temperature and humidity and thus the coupled thermo-hydro-mechanical analysis was performed.

6.1 Governing equations of mechanical problem

Assuming the mechanical problem, the system of static equations which describes equilibrium in the three dimensional domain Ω can be written as follows

$$\boldsymbol{\partial}^{\mathrm{T}}\boldsymbol{\sigma} + \boldsymbol{b} = \boldsymbol{o}, \tag{6.1}$$

where $\boldsymbol{\sigma}^{\mathrm{T}} = (\sigma_x, \sigma_y, \sigma_z, \tau_{yz}, \tau_{zx}, \tau_{xy})$ is the stress vector, $\boldsymbol{b}$ is the vector of body forces and $\boldsymbol{o}$ is the zero vector and $\boldsymbol{\partial}^{\mathrm{T}}$ is the operator matrix defined as

$$\boldsymbol{\partial} = \begin{pmatrix} \dfrac{\partial}{\partial x} & 0 & 0 \\ 0 & \dfrac{\partial}{\partial y} & 0 \\ 0 & 0 & \dfrac{\partial}{\partial z} \\ 0 & \dfrac{\partial}{\partial z} & \dfrac{\partial}{\partial y} \\ \dfrac{\partial}{\partial z} & 0 & \dfrac{\partial}{\partial x} \\ \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} & 0 \end{pmatrix}. \tag{6.2}$$

Constitutive equations relate the stresses from Equation (6.1) with strains and they can be written in the from

$$\boldsymbol{\sigma} = \boldsymbol{D}(\boldsymbol{\varepsilon})\boldsymbol{\varepsilon}, \tag{6.3}$$

where $\boldsymbol{D}(\boldsymbol{\varepsilon})$ represents the stiffness matrix and $\boldsymbol{\varepsilon}$ is the strain vector. The relation between strains $\boldsymbol{\varepsilon}$ and unknown displacements is given by the strain-displacement equations

$$\boldsymbol{\varepsilon} = \boldsymbol{\partial}\boldsymbol{u}, \tag{6.4}$$

where $\boldsymbol{u}$ is the displacement vector.

Equations (6.1), (6.3) and (6.4) must be supplemented by appropriate boundary conditions,

$$\boldsymbol{u} = \bar{\boldsymbol{u}} \quad \text{on} \quad \Gamma_u, \tag{6.5}$$

$$\boldsymbol{S}\boldsymbol{\sigma} = \bar{\boldsymbol{t}} \quad \text{on} \quad \Gamma_t, \tag{6.6}$$

where $\bar{\boldsymbol{u}}$ is the vector of prescribed displacements on the boundary Γ_u and $\bar{\boldsymbol{t}}$ is the vector of surface tractions on the boundary Γ_t. The following relations have to be satisfied for the boundaries Γ_u and Γ_t

$$\Gamma = \Gamma_u \cup \Gamma_t, \tag{6.7}$$

$$\Gamma_u \cap \Gamma_t = \emptyset, \tag{6.8}$$

where the symbol Γ stands for the total boundary of the domain Ω. These boundaries are parts of the domain solved. The matrix $\boldsymbol{S}$ contains components of the unit normal vector, $\boldsymbol{n}$, to the boundary

$$\boldsymbol{S} = \begin{pmatrix} n_x & 0 & 0 & 0 & n_z & n_y \\ 0 & n_y & 0 & n_z & 0 & n_x \\ 0 & 0 & n_z & n_y & n_x & 0 \end{pmatrix}. \tag{6.9}$$

These basic equations for problems in solid mechanics can be discretized using the standard displacement version of the FEM [Zienkiewicz and Taylor, 2000], [Hughes, 1987], [Bittnar and Šejnoha, 1996], where components of the displacement field are approximated as linear combinations of given approximation (shape) functions $N_k(\boldsymbol{x}), k = 1, \ldots, N_n$, where $\boldsymbol{x}$ represents the spatial coordinates. The domain considered is divided by elements connecting N_n nodes where each of the node is associated with one shape function. Every shape function has to met conditions that $N_k(\boldsymbol{x}_k) = 1$ in the given k-th node and $N_k(\boldsymbol{x}_j) = 0, \quad k \neq j$ for all remaining nodes. The displacement field can be then approximated as

$$u_i(\boldsymbol{x}) \approx \sum_{k=1}^{N_n} N_k(\boldsymbol{x}) d_{ki}, \qquad i = 1, 2, 3 \tag{6.10}$$

where d_{ki} are unknown nodal displacements. Equation (6.10) can also be written in the matrix form as

$$\boldsymbol{u}(\boldsymbol{x}) \approx \boldsymbol{N}_u(\boldsymbol{x}) \boldsymbol{d}, \tag{6.11}$$

where $\boldsymbol{N}_u(\boldsymbol{x})$ is the matrix of approximation functions and $\boldsymbol{d}$ is the vector of unknown nodal displacements.

Employing Equation (6.11) in (6.4) the following relationship is obtained

$$\boldsymbol{\varepsilon} \approx \boldsymbol{B}(\boldsymbol{x}) \boldsymbol{d}, \tag{6.12}$$

where $\boldsymbol{B}(\boldsymbol{x})$ is the strain-displacement matrix containing the appropriate derivatives of the shape functions with respect to spatial coordinates.

Substitution of Equation (6.12) into the constitutive equations (6.3) yields the following stress approximation

$$\boldsymbol{\sigma}(\boldsymbol{x}) \approx \boldsymbol{D}(\boldsymbol{\varepsilon}) \boldsymbol{B}(\boldsymbol{x}) \boldsymbol{d}. \tag{6.13}$$

The static equations (6.1) can be replaced by the principle of virtual work [Hughes, 1987] which results in the weak form of the equilibrium equations given by the equality

$$\int_\Omega \boldsymbol{\sigma}^{\mathrm{T}} \delta\boldsymbol{\varepsilon} \, \mathrm{d}\Omega = \int_\Omega \boldsymbol{b}^{\mathrm{T}} \delta\boldsymbol{u} \, \mathrm{d}\Omega + \int_{\Gamma_t} \bar{\boldsymbol{t}}^{\mathrm{T}} \delta\boldsymbol{u} \, \mathrm{d}\Gamma_t. \tag{6.14}$$

The equality (6.1) must be valid for arbitrary virtual displacement field $\delta\boldsymbol{u}$ and virtual strain field $\delta\boldsymbol{\varepsilon}$ which satisfies the kinematic equation $\delta\boldsymbol{\varepsilon} = \boldsymbol{\partial}\delta\boldsymbol{u}$ in Ω, and kinematic boundary conditions $\delta\boldsymbol{u} = \boldsymbol{o}$ on Γ_u. Assuming the same approximation of the virtual displacement field

$$\delta\boldsymbol{u} \approx \boldsymbol{N}_u \delta\boldsymbol{d}, \tag{6.15}$$

yields the virtual strain field in the form

$$\delta\boldsymbol{\varepsilon} \approx \boldsymbol{B} \delta\boldsymbol{d}, \tag{6.16}$$

where $\delta\boldsymbol{d}$ is the vector of virtual displacement parameters. Moreover, similar approximations can be introduced for body forces and tractions given by equations (6.1) and (6.6)

$$\boldsymbol{b} \approx \boldsymbol{N}_b(\boldsymbol{x})\hat{\boldsymbol{b}}, \tag{6.17}$$

$$\bar{\boldsymbol{t}} \approx \boldsymbol{N}_t(\boldsymbol{x})\hat{\boldsymbol{t}}, \tag{6.18}$$

where the vectors $\hat{\boldsymbol{b}}$ and $\hat{\boldsymbol{t}}$ represent nodal values of body forces and surface tractions, respectively. Generally, the approximations of the body forces and tractions are realized with the help of feasible shape functions whose values may be collected in matrices $\boldsymbol{N}_b$ and $\boldsymbol{N}_t$. Typically, the same type of the shape functions is used in the whole problem assuming $\boldsymbol{N}_u = \boldsymbol{N}_b = \boldsymbol{N}_t$.

Substitution of Equations (6.15) and (6.16) into the Equation (6.14), leads to the following condition

$$\int_\Omega \boldsymbol{d}^{\mathrm{T}}\boldsymbol{B}^{\mathrm{T}}(\boldsymbol{x})\boldsymbol{D}^{\mathrm{T}}(\boldsymbol{x})\boldsymbol{B}(\boldsymbol{x})\delta\boldsymbol{d}\,\mathrm{d}\Omega = \int_\Omega \hat{\boldsymbol{b}}^{\mathrm{T}}\boldsymbol{N}_b^{\mathrm{T}}(\boldsymbol{x})\boldsymbol{N}_u(\boldsymbol{x})\delta\boldsymbol{d}\,\mathrm{d}\Omega + \int_{\Gamma_t} \hat{\boldsymbol{t}}^{\mathrm{T}}\boldsymbol{N}_t^{\mathrm{T}}(\boldsymbol{x})\boldsymbol{N}_u(\boldsymbol{x})\delta\boldsymbol{d}\,\mathrm{d}\Gamma_t. \tag{6.19}$$

The vectors $\delta\boldsymbol{d}$ and $\boldsymbol{d}$ may be taken out of integrals because they are not functions of the spatial coordinates. The stiffness matrix can be then defined as

$$\boldsymbol{K} = \int_\Omega \boldsymbol{B}^{\mathrm{T}}(\boldsymbol{x})\boldsymbol{D}(\boldsymbol{x})\boldsymbol{B}(\boldsymbol{x})\,\mathrm{d}\Omega, \tag{6.20}$$

and the load vector as

$$\boldsymbol{f}_{ext} = \int_\Omega \boldsymbol{N}_u^{\mathrm{T}}(\boldsymbol{x})\boldsymbol{N}_b(\boldsymbol{x})\hat{\boldsymbol{b}}\,\mathrm{d}\Omega + \int_{\Gamma_t} \boldsymbol{N}_u^{\mathrm{T}}(\boldsymbol{x})\boldsymbol{N}_t(\boldsymbol{x})\hat{\boldsymbol{t}}\,\mathrm{d}\Gamma_t. \tag{6.21}$$

Assuming the above definitions, Equation (6.19) can be rewritten the form

$$\boldsymbol{K}\boldsymbol{d} = \boldsymbol{f}_{ext}, \tag{6.22}$$

from which the unknown nodal displacements, $\boldsymbol{d}$, can be determined.

For the linear elastic and homogeneous material, the coefficients of the stiffness matrix $\boldsymbol{D}$ are constants and Equation (6.22) represents the system of linear algebraic equations. In the case of nonlinear material behaviour, the stress-strain relation can be written with the help of constitutive operator,$\bar{\boldsymbol{\sigma}}$, as

$$\boldsymbol{\sigma} = \bar{\boldsymbol{\sigma}}(\boldsymbol{\varepsilon}). \tag{6.23}$$

Adopting the same approximation of the strain yields

$$\boldsymbol{\sigma}(\boldsymbol{x}, \boldsymbol{d}) = \bar{\boldsymbol{\sigma}}(\boldsymbol{B}(\boldsymbol{x})\boldsymbol{d}). \tag{6.24}$$

Substitution of the new stress definition in the weak form of the equilibrium equations leads to the following form

$$\int_\Omega \bar{\boldsymbol{\sigma}}^{\mathrm{T}}(\boldsymbol{B}(\boldsymbol{x})\boldsymbol{d})\boldsymbol{B}(\boldsymbol{x})\delta\boldsymbol{d}\,\mathrm{d}\Omega = \int_\Omega \hat{\boldsymbol{b}}^{\mathrm{T}}\boldsymbol{N}_b^{\mathrm{T}}(\boldsymbol{x})\boldsymbol{N}_u(\boldsymbol{x})\delta\boldsymbol{d}\,\mathrm{d}\Omega + \int_{\Gamma_t} \hat{\boldsymbol{t}}^{\mathrm{T}}\boldsymbol{N}_t^{\mathrm{T}}(\boldsymbol{x})\boldsymbol{N}_u(\boldsymbol{x})\delta\boldsymbol{d}\,\mathrm{d}\Gamma_t \tag{6.25}$$

which can be arranged to the final form representing the discretized equations of equilibrium

$$\boldsymbol{f}_{int}(\boldsymbol{d}) = \boldsymbol{f}_{ext}. \tag{6.26}$$

In Equation (6.26), $\boldsymbol{f}_{int}(\boldsymbol{d})$ denotes the vector of internal forces that can be written as

$$\boldsymbol{f}_{int}(\boldsymbol{d}) = \int_{\Omega} \boldsymbol{B}^{\mathrm{T}}(\boldsymbol{x})\bar{\boldsymbol{\sigma}}(\boldsymbol{B}(\boldsymbol{x})\boldsymbol{d})\,\mathrm{d}\Omega. \tag{6.27}$$

The resulting system of algebraic equation (6.26) is nonlinear, so it has to be solved numerically. Typically, the Newton-Raphson iteration procedure is used for solving such systems.

6.2 Mechanical material models

Concrete belongs to heterogeneous materials whose behaviour is very complex. There are creep, shrinkage and thermal dilatancy, which represent time-dependent phenomenons coupled with heat and moisture transfer. The mechanical phenomena are represented by plasticity, damage or crack propagation, to name the most important ones. Usually, capturing the most relevant phenomena requires a combination of several material models. Several creep models can be found in [CEB, 2008], [Jirásek and Bazant, 2002], [Bazant and Jirásek, 2018], crack and damage models are proposed in [de Borst, 1987], [Lemaitre and Chaboche, 1994], [Papa and Taliercio, 1996] and models of plasticity can be found in [Chen and Chen, 1975], [Jirásek and Bazant, 2002], [Ottosen and Ristinmaa, 2005] or [Grassl et al., 2013]. This section is addressed to mechanical material models for the concrete that were implemented in the SIFEL environment, and they were used in analyses of concrete structures mentioned in this chapter.

Assuming small strains, the total strain can be additively decomposed into several components

$$\boldsymbol{\varepsilon}_{tot} = \boldsymbol{\varepsilon}_e + \boldsymbol{\varepsilon}_p + \boldsymbol{\varepsilon}_d + \boldsymbol{\varepsilon}_c + \boldsymbol{\varepsilon}_{sh} + \boldsymbol{\varepsilon}_{ag} + \boldsymbol{\varepsilon}_t, \tag{6.28}$$

where $\boldsymbol{\varepsilon}_{tot}$ denotes the total strain, $\boldsymbol{\varepsilon}_e$ denotes the elastic strain, $\boldsymbol{\varepsilon}_p$ stands for the plastic strain, $\boldsymbol{\varepsilon}_d$ stands for damage strain, $\boldsymbol{\varepsilon}_c$ is creep strain, $\boldsymbol{\varepsilon}_{sh}$ denotes part of strain caused by shrinkage, $\boldsymbol{\varepsilon}_{ag}$ stands for strain caused by ageing, $\boldsymbol{\varepsilon}_t$ is free thermal strain. The resulting stress $\boldsymbol{\sigma}$ can be expressed from the Hook's law in the form

$$\boldsymbol{\sigma} = \boldsymbol{D}_e\boldsymbol{\varepsilon}_e, \tag{6.29}$$

where $\boldsymbol{D}_e$ is the elastic stiffness of material.

6.2.1 Analysis of structural shrinkage and creep in concrete

Under assumption of uniaxial stress σ [Bažant, 1988], the expression Equation (6.28) can be arranged as follows

$$\varepsilon_{tot}(t) = \varepsilon_\sigma + \varepsilon_0, \tag{6.30}$$

where ε_σ is the strain depending on the stress state, ε_0 is the strain due to other effects that are not caused by the applied mechanical load.

Experiments showed that stress-strain relation is almost linear (linear creep) under service load levels, i.e. $\sigma < 0.4 f_{\mathrm{ck}}$, where f_{ck} is the compressive strength. In cases where the Boltzmann principle of superposition can be applied and the stress σ varies in time, the total strain at time t due to variation of stress in time is a sum of strain increments caused by the particular constant stress increments $\Delta\sigma_i$, applied at time τ_i

$$\varepsilon_{tot} = J(t, t_0)\sigma(t_0) + \int_{t_0}^{t} J(t, \tau)\mathrm{d}\sigma(\tau) + \varepsilon_0. \tag{6.31}$$

In Equation (6.31), $J(t, \tau)$ is the compliance function of linear viscoelastic materials which yields the strain at time t due to unit stress $\sigma = 1$ applied at time τ. Definition of the compliance function varies among the particular creep models. Bazant's compliance function, known as the B3 model, belongs to the most popular ones in the engineering community. It originates from the set of experiments carried out since 1970 [Baweja and Bažant, 1995].

Continuous model

The integral constitutive relation in Equation (6.31) can be solved with the help of the continuous Kelvin chain model with an infinite number of units and retardation times with infinite close spacing [Bažant and Xi, 1995]. In [Baweja and Bažant, 1995], this approach was applied on the B3 compliance function with log-power law written in the form

$$J(t, \tau) = q_1 + C(\xi), \tag{6.32}$$

where $\xi = t - \tau$, t represents the concrete age and creep compliance function $C(\xi)$ is defined as

$$C(\xi) = q_3 \ln\left[1 + \left(\frac{\xi}{\lambda_0}\right)^n\right]. \tag{6.33}$$

The compliance function, $C(\xi)$, has parameters q_3, $\lambda_0 = 1$ and n. It can be approximated in the continuous form

$$C(\xi) = \int_0^{\infty} L(\tau)(1 - e^{-\xi/\tau})\mathrm{d}(\ln\tau). \tag{6.34}$$

where τ is the time of the load application, $L(\tau)$ denotes the continuous retardation spectrum whose meaning in the logarithmic scale corresponds to the material compliance of the Kelvin unit in the actual time scale. Derivation $L(\tau)$ from the known compliance function of the material was proposed in [Tschoegl, 1971] and [Tschoegl, 1989].

Using Equation (6.34) and setting $\tau = 1/\zeta$ with $\mathrm{d}(\ln\tau) = -\mathrm{d}(\ln\zeta)$, the creep compliance function reads

$$\begin{aligned} C(\xi) &= \int_0^{\infty} L(\zeta^{-1})(1 - e^{-\xi\zeta})\zeta^{-1}\mathrm{d}\zeta = \\ &= \int_0^{\infty} L(\zeta^{-1})\zeta^{-1}\mathrm{d}(\zeta) - \int_0^{\infty} L(\zeta^{-1})e^{-\xi\zeta}\zeta^{-1}\mathrm{d}\zeta. \end{aligned} \tag{6.35}$$

Application of the Laplace transform on Equation (6.35) results in the form

$$C(\xi) = f(0) - f(\xi), \tag{6.36}$$

$$f(\xi) = \int_0^\infty L(\zeta^{-1}) e^{-\xi\zeta} \zeta^{-1} \mathrm{d}\zeta, \tag{6.37}$$

where $f(\xi)$ represents Laplace transform of the function $L(\zeta^{-1})\zeta^{-1}$ with the transformed variable ξ. The Laplace transform is inverted with the help of the inversion operator proposed in [Widder, 1946]

$$F_{k,\zeta}[f(\xi)] = \frac{(-1)^k}{k!} \left(\frac{k}{\zeta}\right)^{k+1} f^{(k)}\left(\frac{k}{\zeta}\right) \tag{6.38}$$

which has the following property

$$\lim_{k\to\infty} F_{k,\zeta}[f(\xi)] = \lim_{k\to\infty} \left[\frac{(-1)^k}{k!} \left(\frac{k}{\zeta}\right)^{k+1} f^{(k)}\left(\frac{k}{\zeta}\right)\right] = L(\zeta^{-1})\zeta^{-1}. \tag{6.39}$$

In Eq (6.39), $f^{(k)}$ is the k-th derivative of function f ($f(0)$ is constant). For $k \geq 1$,

$$L(\tau) = -\lim_{k\to\infty} \frac{(-k\tau)^k}{(k-1)!} C^{(k)}(k\tau). \tag{6.40}$$

The finite value of k yields the approximate spectrum of the k-th order. The transform can be rewritten using Equations (6.33), (6.36) and (6.37) as

$$f(\xi) = q_3 \ln(1+\xi^n) - \int_0^\infty L(\zeta^{-1})\zeta^{-1} \mathrm{d}\zeta, \tag{6.41}$$

Assuming $k = 3$ should be sufficient in practice, and for that value, Equation (6.40) yields the approximation

$$\begin{aligned} L(\tau) &= \left[\frac{-2n^2(3\tau)^{2n-3}[n-1-(3\tau)^n]}{[1+(3\tau)^n]^3}\right] \frac{(3\tau)^3}{2} q_3 \\ &+ \left[\frac{n(n-2)(3\tau)^{n-3}[n-1-(3\tau)^n] - n^2(3\tau)^{2n-3}}{[1+(3\tau)^n]^2}\right] \frac{(3\tau)^3}{2} q_3. \end{aligned} \tag{6.42}$$

In the numerical computation, the integral in (6.34) is approximated by the finite sum and $\ln(\tau)$ is subdivided into time intervals $\Delta\ln(\tau) = \ln 10 \Delta(\log(\tau_\mu))$

$$C(\xi) = \sum_{\mu=1}^{M} L(\tau_\mu) \left[1 - e^{-\xi/\tau_\mu}\right] \ln 10 \Delta(\log(\tau_\mu)), \quad \text{or} \quad C(\xi) = \sum_{\mu=1}^{M} B_\mu \left[1 - e^{-\xi/\tau_\mu}\right], \tag{6.43}$$

where

$$B_\mu = L(\tau_\mu) \ln 10 \Delta(\log(\tau_\mu)). \tag{6.44}$$

$L(\tau_\mu)$ is given by Equation (6.42) and $\Delta(\log(\tau_\mu)$ is equal to the time interval between two adjacent Kelvin units in the logarithmic scale [Bažant and Xi, 1995].

6.2.2 B3 creep model

One of the most popular creep models represents Bazant's B3 model with logarithmic-power law where compliance function $J(t,\tau)$ is given by

$$J(t,\tau) = q_1 + q_2 Q(t,\tau) + q_3 \ln\left[1 + \left(\frac{t-\tau}{\lambda_0}\right)^n\right] + q_4 \ln\left(\frac{t}{\tau}\right). \tag{6.45}$$

Parameter q_1 is the instantaneous strain due to unit stress, the coefficient q_2 is the ageing viscoelastic compliance, q_3 represents non-ageing viscoelastic compliance, and q_4 is the flow compliance. The coefficient λ_0 is considered to be almost 1.0, and $Q(t,\tau)$ is a binomial integral. The more details about the coefficients can be found in [Baweja and Bažant, 1995] and Bazant and Jirásek [2018].

6.2.3 Moisture and temperature effects in the concrete model

Temperature and moisture changes in concrete influence namely three phenomena in concrete creep (solidification theory model [Bazant et al., 2004]):

- The ageing of concrete, which causes a significant decrease of creep with the age at loading. There are two types of ageing:
  - Short-term chemical ageing, which ceases at room temperature after about a year .
  - Long-term non-chemical ageing, which is observed even many years after the degree of hydration of cement ceased to grow.
- The drying creep effect, (the stress-induced shrinkage) which takes a place during drying and it is manifested by larger apparent creep than the basic creep.
- The transitional creep, which represents a transient increase of creep after a temperature change, both heating and cooling.

There are two effects of temperature on concrete creep [Bazant et al., 2004], generated by two different mechanism:

- A temperature increase accelerates the bond breakages and restorations causing creep, and thus increases the creep rate.
- The temperature rise accelerates the chemical process of cement hydration and thus the ageing of concrete, which reduces the creep rate.

The effect of bond breakage usually prevails, and thus the temperature rise leads to increased creep.

The effect of moisture changes in the ageing of concrete is described in [Bazant et al., 2004]. A decrease in relative humidity, φ, is linked with a decrease in the rate of hydration and creep. Levels of φ close to 0.3 results in an almost zero rate of ageing. The special time quantities are introduced in the creep model:

- Reduced time t_r characterizing the changes in the rate of bond breakages and restoration on the microstructural level

$$t_r(t) = \int_0^t \psi(\tau)\mathrm{d}\tau \leq t, \tag{6.46}$$

where

$$\begin{aligned} \psi(t) &= \psi_T(t)\psi_\varphi(t), & (6.47)\\ \psi_T(t) &= \exp\left\{\frac{Q_v}{R}\left(\frac{1}{T_0}-\frac{1}{T}\right)\right\}, & (6.48)\\ \psi_\varphi(t) &= \alpha_\varphi + (1-\alpha_\varphi)\varphi^2(t). & (6.49) \end{aligned}$$

In the above equations, T is the absolute temperature, T_0 is the reference temperature, R is the gas constant, Q_v is the activation energy for the viscous processes and α_φ is a material parameter that has to be determined experimentally. From the large set of experiments conducted by Bazant, the following values can be considered $T_0 = 294$ K, $Q_v/R = 5000$ K and $\alpha_\varphi = 0.1$.

- Equivalent time t_e, which characterizes the degree of hydration indirectly

$$t_e(t) = \int_0^t \beta(\tau)\mathrm{d}\tau, \quad t_e \geq t \tag{6.50}$$

where

$$\begin{aligned} \beta(t) &= \beta_T(t)\beta_\varphi(t), & (6.51)\\ \beta_T(t) &= \exp\left\{\frac{Q_h}{R}\left(\frac{1}{T_0}-\frac{1}{T}\right)\right\}, & (6.52)\\ \beta_\varphi(t) &= \{1+[a_\varphi - a_\varphi\varphi(t)]^4\}^{-1}. & (6.53) \end{aligned}$$

In the above equation, Q_h is the activation energy, $Q_h/R = 2700$ K and $a_\varphi = 5$.

In the case of zero stress level, the effects of temperature and humidity changes (structural thermal expansion and shrinkage) can be defined in terms of strain rates as

- rate of the thermal expansion strain

$$\dot{\boldsymbol{\varepsilon}}_t = \boldsymbol{\alpha}\dot{T}, \tag{6.54}$$

- rate of the drying shrinkage strain

$$\dot{\boldsymbol{\varepsilon}}_{sh} = \boldsymbol{k}\dot{\varphi} \tag{6.55}$$

where the incremental shrinkage coefficient vector, $\boldsymbol{k}^{\mathrm{T}} = (k_{11}, k_{22}, k_{33}, k_{23}, k_{31}, k_{12})$, can be expressed in terms of φ, T and t_e. $\boldsymbol{\alpha}^{\mathrm{T}} = (\alpha_{11}, \alpha_{22}, \alpha_{33}, \alpha_{23}, \alpha_{31}, \alpha_{12})$ denotes the thermal expansion coefficient vector.

Considering the shrinkage and thermal expansion to be stress independent, they can be expressed as follows

$$\boldsymbol{k} = \varepsilon_{sh}^0\psi\boldsymbol{m}, \quad \boldsymbol{\alpha} = \alpha^0\boldsymbol{m}, \tag{6.56}$$

where $(-\varepsilon^0_{sh})$ ranges in [0.0002; 0.001], α^0 is an empirical constant, and $(-\psi) = E(t_0)/E(t_e)3\varphi^2$ for $0.4 \leq \varphi \leq 0.98$.

If the material is subjected to stresses, the linear functions of the stress vector [Bazant and Chern, 1985] are considered for the approximation of the shrinkage and thermal expansion coefficient vectors

$$\boldsymbol{k} = \varepsilon^0_{sh}\psi(\boldsymbol{m} + \bar{\eta}\boldsymbol{\sigma}\mathrm{sign}(\dot{H})), \quad \boldsymbol{\alpha} = \alpha^0(\boldsymbol{m} + \bar{\zeta}\boldsymbol{\sigma}\mathrm{sign}(\dot{H})), \tag{6.57}$$

where $\dot{H} = \dot{\varphi} + c\dot{T}$ (with constant $c \geq 0$). Coefficients $\bar{\eta}$ and $\bar{\zeta}$ are determined empirically and their typical values are taken proportionally with respect to inverse value of the tensile strength, f_t, as $\bar{\eta} \in [0.1/f_t; 0.6/f_t]$ (MPa^{-1}) and $\bar{\zeta} \in [1/f_t; 2/f_t]$ (MPa^{-1}). Equations (6.57) may be simplified according to [Bažant, 1988] by considering $c \to 0$ in case of $\boldsymbol{k}$, to get $\mathrm{sign}(\dot{H}) = \mathrm{sign}(\dot{\varphi})$, and by setting $c \to \infty$ in case of $\boldsymbol{\alpha}$, thus yielding $\mathrm{sign}(\dot{H}) = \mathrm{sign}(\dot{T})$.

Generalization into 3D and including incremental form of shrinkage $\Delta\boldsymbol{\varepsilon}_{\mathrm{sh}} = \boldsymbol{k}\Delta\varphi$ and thermal dilatation $\Delta\boldsymbol{\varepsilon}_t = \boldsymbol{\alpha}\Delta T$, the incremental constitutive equation is obtained

$$\Delta\boldsymbol{\sigma} = J^{-1}(t, t_0)\widehat{\boldsymbol{D}}\big(\Delta\boldsymbol{\varepsilon} - \boldsymbol{k}\Delta\varphi - \boldsymbol{\alpha}\Delta T - \Delta\boldsymbol{\varepsilon}_{\mathrm{c}} - \Delta\boldsymbol{\varepsilon}_{\mathrm{d}}\big), \tag{6.58}$$

where $\widehat{\boldsymbol{D}}$ may be defined with the help of Poisson's ratio, ν, as

$$\widehat{\boldsymbol{D}} = \frac{1}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5-\nu & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5-\nu & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5-\nu \end{pmatrix}. \tag{6.59}$$

6.2.4 Damage Model

The scalar isotropic damage model belongs to popular models of continuum damage mechanics. It can be successfully used for problems of bending. The model is described in detail, e.g., in [Lemaitre and Chaboche, 1994] and [Skrzypek and Ganczarski, 1999]. Considering the one-dimensional case and the virgin state of the material, the area of the cross-section of a bar element is denoted by A. If the bar element is subjected to increasing uniaxial stress, defects start to evolve at a certain strain threshold. Let the area of these defects be denoted by A_d. The undamaged area can then be defined as $\tilde{A} = A - A_d$. The equivalence condition on the bar element between the nominal stress assumed on the original cross-section area, A, and stress acting on the undamaged cross-section area $\tilde{A}$ can be written in the form

$$\sigma A = \tilde{\sigma}\tilde{A}. \tag{6.60}$$

A dimensionless damage parameter ω can then be defined as

$$\omega = \frac{A_d}{A}. \tag{6.61}$$

The stress-strain relation of the scalar isotropic material reads

$$\sigma = (1-\omega)E\bar{\varepsilon}, \tag{6.62}$$

where $\bar{\varepsilon}$ represents the total strains decreased by the irreversible strain components

$$\bar{\varepsilon} = \varepsilon_{tot} - \varepsilon_p - \varepsilon_c - \varepsilon_{ag} - \varepsilon_{sh} - \varepsilon_t. \tag{6.63}$$

Equation (6.62) can be rewritten to the form

$$\sigma = E(\bar{\varepsilon} - \varepsilon_d) = E\varepsilon_e, \tag{6.64}$$

where

$$\varepsilon_d = \omega\bar{\varepsilon}, \tag{6.65}$$

with the definition of $\bar{\boldsymbol{\varepsilon}}$ corresponding to Equation (6.63).

For the three-dimensional case, the stress-strain relation can be written similarly

$$\boldsymbol{\sigma} = (1-\omega)\boldsymbol{D}_e\bar{\boldsymbol{\varepsilon}}. \tag{6.66}$$

A suitable evolution law for the damage parameter, ω, has to be defined for the given material type. In reference [Papa and Taliercio, 1996], the evolution law suitable for concrete was proposed in the form

$$\omega = \begin{cases} 0 & \text{for } \bar{\kappa} \leq \bar{\varepsilon_0} \\ \dfrac{a(\bar{\kappa} - \bar{\varepsilon}_0)^b}{1 + a(\bar{\kappa} - \bar{\varepsilon}_0)^b} & \text{for } \bar{\kappa} > \bar{\varepsilon_0}, \end{cases} \tag{6.67}$$

where $\bar{\varepsilon}_0$ is the strain threshold which distinguishes elastic ($\bar{\kappa} \leq \bar{\varepsilon}_0$) and inelastic behaviour ($\bar{\kappa} > \bar{\varepsilon}_0$) and $\bar{\kappa}$ is the maximum attained strain $\bar{\varepsilon}$ in the loading history. Material parameters a and b control the peak stress value and slope of the softening branch.

Generally, the problem of mesh sensitivity arises in FEM for materials with softening behaviour, e.g. [Lemaitre and Chaboche, 1994] and therefore it must be considered for damage models as well. The problem is connected with the dissipated energy which depends on the characteristic size of a damaged element and tends to zero with decreasing characteristic size of the element. For these instances, the method of the variable softening modulus was developed in [Pietruszczak and Mróz, 1981], which allows avoiding of the spurious mesh dependency. In this method, the characteristic element length is involved into the damage evolution law. In the one-dimensional case, the stress can be written in the form

$$\sigma = f_t \exp\left(-\frac{w_{cr}}{w_{cr0}}\right), \tag{6.68}$$

where f_t is the tensile strength in [Pa], w_{cr} is the crack opening in [m] and w_{cr0} is the material parameter controlling the initial slope of the softening branch in [m]. Consequently, the crack opening is considered to be smeared over the element as follows

$$\bar{\kappa} - \varepsilon_e = \frac{w_{cr}}{h}, \tag{6.69}$$

where h is the characteristic element length. Using Equations (6.62), (6.68) and (6.69) yields the resulting nonlinear equation for the damage parameter ω

$$(1-\omega)E\bar{\kappa} = f_t \exp\left(-\frac{\omega h\bar{\kappa}}{w_{cr0}}\right). \tag{6.70}$$

The nonlinear equation can be solved, e.g., with the help of the Newton method.

For the two-dimensional or three-dimensional stress states, the strain $\bar{\kappa}$ has to be substituted by equivalent strain κ_{eq}. In the case of concrete modelling, the equivalent strain, κ_{eq}, can be defined with the help of the Mazars' norm [Mazars and Pijaudier-Cabot, 1989]

$$\kappa_{eq} = \sqrt{\langle \bar{\varepsilon}_\alpha \rangle \langle \bar{\varepsilon}_\alpha \rangle}, \tag{6.71}$$

where $\bar{\varepsilon}_\alpha$ denotes the principal values of the strain tensor $\bar{\varepsilon}$ and the symbol $\langle\rangle$ denotes selection of positive components (Macaulay brackets).

6.3 Coupled heat and moisture transfer model

The model for the coupled heat and moisture transport proposed by Künzel and Kiessl [Künzel and Kiessl, 1996] is the phenomenological model developed for the description of the coupled heat and moisture transfer in porous materials such as concrete and masonry. The model is intended for the materials subjected to common climatic conditions, and its formulation allows for the determination of the material parameters from relatively simple laboratory measurements.

The model introduces two primary unknowns at the material - relative humidity φ [-] and temperature T [K]. Simultaneous water and water vapour transport is described by the relative humidity, φ, which is considered to be the only moisture potential for both hygroscopic and over-hygroscopic regions. Moreover, the over-hygroscopic region is divided into two subregions - capillary water region and supersaturated region, where different conditions for water and water vapour transport are considered.

6.3.1 Transport equations

The proposed model considers that water vapour diffusion and liquid transport [Künzel and Kiessl, 1996] are the only the moisture transport mechanisms relevant in the field of building physics. Liquid transport is linked with the small capillaries, while the vapour diffusion takes a place in large pores.

The liquid transport mechanism consists of liquid flow in the absorbed layer (surface diffusion) and water filled capillaries (capillary transport). For the both sources of flow, the driving potential may be considered to be either suction stress (capillary pressure) or relative humidity φ. The flux of liquid water in terms of relative humidity can be written as

$$\boldsymbol{J}_w = -D_\varphi \boldsymbol{\nabla} \varphi, \tag{6.72}$$

where the liquid conductivity D_φ [$\mathrm{kg\,m^{-1}\,s^{-1}}$] is the product of the liquid diffusivity D_w [$\mathrm{m^2\,s^{-1}}$] and the derivative of water retention function $D_\varphi = D_w \times \mathrm{d}w/\mathrm{d}\varphi$.

The Fick's law is used for the description of vapour diffusion. It relates the vapour flux, $\boldsymbol{J}_v$ and the relative humidity gradient as follows

$$\boldsymbol{J}_v = -\delta_p \boldsymbol{\nabla}(p_{sat}\,\varphi) = -\frac{\delta}{\mu} \boldsymbol{\nabla}(p_{sat}\,\varphi), \tag{6.73}$$

where δ_p [kg m s^{-1} Pa^{-1}] is the vapour permeability of the porous material, ($p_{sat}\,\varphi$) denotes vapour pressure [Pa], p_{sat} is the water vapour saturation pressure, δ [kg m s^{-1} Pa^{-1}] is the vapour diffusion coefficient in the air and μ is the vapour diffusion resistance.

The heat transfer is described by the Fourier's law, which states that heat flux is proportional to the temperature gradient with proportional factor called thermal conductivity. It can be written in the following form

$$\boldsymbol{q} = -\lambda \boldsymbol{\nabla} T, \tag{6.74}$$

where λ [W m^{-1} K^{-1}] is the thermal conductivity of the moist material. The enthalpy flows through moisture movement and phase transition is taken into account in the form of source terms in the heat balance equation.

6.3.2 Balance equations

Balance equations represent in the transfer processes some kind of conservation law. The energy conservation law is linked with heat transfer, while the mass conservation law is associated with moisture transfer. They are closely coupled because the moisture content depends on the total enthalpy and thermal conductivity, while the moisture flow influences the temperature.

Both laws can be described as a set of partial differential equations in terms of temperature T and relative humidity φ. They can be expressed on the domain Ω in the following form

$$\frac{\partial w}{\partial \varphi}\frac{\partial \varphi}{\partial t} = \boldsymbol{\nabla}^{\mathrm{T}}\big(D_\varphi \boldsymbol{\nabla}\varphi + \delta_p \boldsymbol{\nabla}(\varphi p_{\mathrm{sat}})\big), \qquad \boldsymbol{x} \in \Omega, \tag{6.75}$$

$$\Big(\rho C + \frac{\partial H_w}{\partial T}\Big)\frac{\partial T}{\partial t} = \boldsymbol{\nabla}^{\mathrm{T}}\big(\lambda\, \boldsymbol{\nabla} T\big) + h_v \boldsymbol{\nabla}^{\mathrm{T}}\big(\delta_p \boldsymbol{\nabla}(\varphi p_{\mathrm{sat}})\big), \qquad \boldsymbol{x} \in \Omega, \tag{6.76}$$

where H_w [J m^{-3}] is the enthalpy of the material moisture, w [kg m^{-3}] is the water content of the material, h_v [J kg^{-1}] is the evaporation enthalpy of the water, p_{sat} [Pa] is the water vapour saturation pressure, ρ [kg m^{-3}] is the material density, C [J kg^{-1} K^{-1}] is the specific heat capacity and t [s] denotes time.

Boundary of the domain Ω is split into parts Γ_T, Γ_φ, Γ_{qpT}, $\Gamma_{Jp\varphi}$, Γ_{qcT} and $\Gamma_{Jc\varphi}$ which are disjoint and their union is the whole boundary $\Gamma = \partial\Omega = \Gamma_T \cup \Gamma_\varphi \cup \Gamma_{qpT} \cup \Gamma_{Jp\varphi} \cup \Gamma_{qcT} \cup \Gamma_{Jc\varphi}$.

6.3.3 Boundary conditions

The following types of boundary conditions can be given for the system of equations (6.75) and (6.76):

- Dirichlet boundary conditions

$$T(\boldsymbol{x}, t) = \overline{T}(\boldsymbol{x}, t), \qquad \boldsymbol{x} \in \Gamma_T \tag{6.77}$$

$$\varphi(\boldsymbol{x}, t) = \overline{\varphi}(\boldsymbol{x}, t), \qquad \boldsymbol{x} \in \Gamma_\varphi \tag{6.78}$$

- Neumann boundary conditions

$$-\lambda\frac{\mathrm{d}T}{\mathrm{d}\boldsymbol{n}} = \boldsymbol{q}(\boldsymbol{x},t) = \overline{\boldsymbol{q}}(\boldsymbol{x},t), \qquad \boldsymbol{x}\in\Gamma_{qpT}, \tag{6.79}$$

$$-D_\varphi\frac{\mathrm{d}\varphi}{\mathrm{d}\boldsymbol{n}} = \boldsymbol{J}(\boldsymbol{x},t) = \overline{\boldsymbol{J}}(\boldsymbol{x},t), \qquad \boldsymbol{x}\in\Gamma_{Jp\varphi}, \tag{6.80}$$

- Cauchy boundary conditions

$$\boldsymbol{q}(\boldsymbol{x},t) = \alpha(T(\boldsymbol{x},t) - T_E(\boldsymbol{x},t)), \qquad \boldsymbol{x}\in\Gamma_{qcT}, \tag{6.81}$$

$$\boldsymbol{J}(\boldsymbol{x},t) = \beta(p(\boldsymbol{x},t) - p_E(\boldsymbol{x},t)), \qquad \boldsymbol{x}\in\Gamma_{Jc\varphi}, \tag{6.82}$$

where $\overline{T}(\boldsymbol{x},t)$ is the prescribed temperature, $\overline{\varphi}(\boldsymbol{x},t)$ is the prescribed relative humidity, $\overline{\boldsymbol{q}}(\boldsymbol{x},t)$ is the prescribed heat flux, $\overline{\boldsymbol{J}}(\boldsymbol{x},t)$ is the prescribed moisture flux, α [W m^{-2} K^{-1}] and β [kg s^{-1} Pa^{-1}] are the heat and mass transfer coefficient, T_E is the ambient temperature and p_E is the ambient water vapour pressure.

6.3.4 Discretization of the differential equations

The spatial discretization of the partial differential equations Equations (6.75) and (6.76) is performed with the help of finite element method. The weighted residual statement can be applied on both balance equation assuming $\delta T = 0$ on Γ_T and $\delta\varphi = 0$ on Γ_φ which lead to the form

$$\int_\Omega \delta\varphi\Big(\frac{\partial w}{\partial\varphi}\frac{\partial\varphi}{\partial t} - \boldsymbol{\nabla}^{\mathrm{T}}\big(D_\varphi\boldsymbol{\nabla}\varphi + \delta_p\boldsymbol{\nabla}(\varphi p_{\mathrm{sat}})\big)\Big)\mathrm{d}\Omega = 0 \tag{6.83}$$

$$\int_\Omega \delta T\left(\left(\rho C\ + \frac{\partial H_w}{\partial T}\right)\frac{\partial T}{\partial t} - \boldsymbol{\nabla}^{\mathrm{T}}\left(\lambda\ \boldsymbol{\nabla}T\right) - h_v\boldsymbol{\nabla}^{\mathrm{T}}\left(\delta_p\boldsymbol{\nabla}(\varphi p_{\mathrm{sat}})\right)\right)\mathrm{d}\Omega = 0. \tag{6.84}$$

Then the Green's theorem can be applied on the weak formulation of the mass balance equation which results in

$$\begin{aligned}\int_\Omega \delta\varphi\Big(\frac{\partial w}{\partial\varphi}\frac{\partial\varphi}{\partial t}\Big)\mathrm{d}\Omega + \int_\Omega \boldsymbol{\nabla}\delta\varphi\Big(D_w\frac{\mathrm{d}w}{\mathrm{d}\varphi} + \delta_p p_{\mathrm{sat}}\Big)\boldsymbol{\nabla}\varphi\mathrm{d}\Omega + \int_\Omega \boldsymbol{\nabla}\delta\varphi\Big(\delta_p\varphi\frac{\mathrm{d}p_{\mathrm{sat}}}{\mathrm{d}T}\Big)\boldsymbol{\nabla}T\mathrm{d}\Omega \\ - \int_{\Gamma_J}\delta\varphi\Big(D_w\frac{\mathrm{d}w}{\mathrm{d}\varphi} + \delta_p p_{\mathrm{sat}}\Big)\frac{\partial\varphi}{\partial\boldsymbol{n}}\mathrm{d}\Gamma - \int_{\Gamma_q}\delta\varphi\Big(\delta_p\varphi\frac{\mathrm{d}p_{\mathrm{sat}}}{\mathrm{d}T}\Big)\frac{\partial T}{\partial\boldsymbol{n}}\mathrm{d}\Gamma = 0.\end{aligned} \tag{6.85}$$

where $\Gamma_J = \Gamma_{Jc\varphi}\cup\Gamma_{Jp\varphi}$ and $\Gamma_q = \Gamma_{qcT}\cup\Gamma_{qpT}$. The similar approach can be used for the weak formulation for heat transfer which yields

$$\begin{aligned}\int_\Omega \delta T\Big(\rho C\ + \frac{\partial H_w}{\partial T}\Big)\frac{\partial T}{\partial t}\mathrm{d}\Omega + \int_\Omega \boldsymbol{\nabla}\delta T\Big(\lambda + h_v\delta_p\varphi\frac{\mathrm{d}p_{\mathrm{sat}}}{\mathrm{d}T}\Big)\boldsymbol{\nabla}T\mathrm{d}\Omega + \\ \int_\Omega \boldsymbol{\nabla}\delta T\Big(h_v\delta_p p_{\mathrm{sat}}\Big)\boldsymbol{\nabla}\varphi\mathrm{d}\Omega - \int_{\Gamma_J}\delta T\Big(h_v\delta_p p_{\mathrm{sat}}\Big)\frac{\partial\varphi}{\partial\boldsymbol{n}}\mathrm{d}\Gamma \\ - \int_{\Gamma_q}\delta T\Big(\lambda + h_v\delta_p\varphi\frac{\mathrm{d}p_{\mathrm{sat}}}{\mathrm{d}T}\Big)\frac{\partial T}{\partial\boldsymbol{n}}\mathrm{d}\Gamma \ = \ 0.\end{aligned} \tag{6.86}$$

The temperature field, T, and relative humidity filed, φ, may be approximated in the form

$$T = \boldsymbol{N}_T(\boldsymbol{x})\boldsymbol{d}_T, \qquad \varphi = \boldsymbol{N}_\varphi(\boldsymbol{x})\boldsymbol{d}_\varphi, \tag{6.87}$$

where $\boldsymbol{N}_\alpha(\boldsymbol{x})$ denotes the matrix of approximation functions of the α quantity, $\boldsymbol{d}_T$ denotes the vector of nodal temperatures and $\boldsymbol{d}_\varphi$ denotes the vector of nodal relative humidities. A similar approximation may be used for gradients of temperature and relative humidity

$$\boldsymbol{\nabla} T = \boldsymbol{B}(\boldsymbol{x})\boldsymbol{d}_T, \qquad \boldsymbol{\nabla}\varphi = \boldsymbol{B}(\boldsymbol{x})\boldsymbol{d}_\varphi. \tag{6.88}$$

where $\boldsymbol{B}(\boldsymbol{x})$ is the matrix of derivatives of approximation functions. Using approximations (6.87) and (6.88) in Equations (6.85) and (6.86) yields a set of the first order differential equations in the matrix form

$$\begin{pmatrix} \boldsymbol{K}_{\varphi\varphi} & \boldsymbol{K}_{\varphi T} \\ \boldsymbol{K}_{T\varphi} & \boldsymbol{K}_{TT} \end{pmatrix} \begin{pmatrix} \boldsymbol{d}_\varphi \\ \boldsymbol{d}_T \end{pmatrix} + \begin{pmatrix} \boldsymbol{C}_{\varphi\varphi} & \boldsymbol{C}_{\varphi T} \\ \boldsymbol{C}_{T\varphi} & \boldsymbol{C}_{TT} \end{pmatrix} \begin{pmatrix} \dot{\boldsymbol{d}}_\varphi \\ \dot{\boldsymbol{d}}_T \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_\varphi \\ \boldsymbol{f}_T \end{pmatrix}. \tag{6.89}$$

The matrices $\boldsymbol{K}_{\varphi\varphi}$, $\boldsymbol{K}_{\varphi T}$, $\boldsymbol{K}_{T\varphi}$ and $\boldsymbol{K}_{TT}$ represent blocks of the conductivity matrix of the problem and they are defined as

$$\boldsymbol{K}_{\varphi\varphi} = \int_\Omega \boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}_{\varphi\varphi}\boldsymbol{B}\mathrm{d}\Omega, \quad \boldsymbol{K}_{\varphi T} = \int_\Omega \boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}_{\varphi T}\boldsymbol{B}\mathrm{d}\Omega, \tag{6.90}$$

$$\boldsymbol{K}_{T\varphi} = \int_\Omega \boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}_{T\varphi}\boldsymbol{B}\mathrm{d}\Omega, \quad \boldsymbol{K}_{TT} = \int_\Omega \boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}_{TT}\boldsymbol{B}\mathrm{d}\Omega. \tag{6.91}$$

where $\boldsymbol{D}_{\varphi\varphi}$, $\boldsymbol{D}_{\varphi T}$, $\boldsymbol{D}_{T\varphi}$ and $\boldsymbol{D}_{TT}$ are the material conductivity matrices. These matrices have the diagonal form where the diagonal entries are equal to appropriate conductivities

$$k_{\varphi\varphi} = D_w\frac{\mathrm{d}w}{\mathrm{d}\varphi} + \delta_p p_{\mathrm{sat}}, \quad k_{\varphi T} = \delta_p\varphi\frac{\mathrm{d}p_{\mathrm{sat}}}{\mathrm{d}T}, \tag{6.92}$$

$$k_{T\varphi} = h_v\delta_p p_{\mathrm{sat}}, \quad k_{TT} = \lambda + h_v\delta_p\varphi\frac{\mathrm{d}p_{\mathrm{sat}}}{\mathrm{d}T}. \tag{6.93}$$

Similarly, the matrices $\boldsymbol{C}_{\varphi\varphi}$, $\boldsymbol{C}_{\varphi T}$, $\boldsymbol{C}_{T\varphi}$ and $\boldsymbol{C}_{TT}$ represent blocks of the capacity matrix of the problem and they can be expressed as

$$\boldsymbol{C}_{\varphi\varphi} = \int_\Omega \boldsymbol{N}_\varphi^{\mathrm{T}}\boldsymbol{H}_{\varphi\varphi}\boldsymbol{N}_\varphi\mathrm{d}\Omega, \quad \boldsymbol{C}_{\varphi T} = \int_\Omega \boldsymbol{N}_\varphi^{\mathrm{T}}\boldsymbol{H}_{\varphi T}\boldsymbol{N}_T\mathrm{d}\Omega, \tag{6.94}$$

$$\boldsymbol{C}_{T\varphi} = \int_\Omega \boldsymbol{N}_T^{\mathrm{T}}\boldsymbol{H}_{T\varphi}\boldsymbol{N}_\varphi\mathrm{d}\Omega, \quad \boldsymbol{C}_{TT} = \int_\Omega \boldsymbol{N}_T^{\mathrm{T}}\boldsymbol{H}_{TT}\boldsymbol{N}_T\mathrm{d}\Omega, \tag{6.95}$$

where $\boldsymbol{H}_{\varphi\varphi}$, $\boldsymbol{H}_{\varphi T}$, $\boldsymbol{H}_{T\varphi}$ and $\boldsymbol{H}_{TT}$ are the material capacity matrices which have diagonal form too. In this case, diagonal entries are equal to appropriate capacities

$$c_{\varphi\varphi} = \frac{\partial w}{\partial\varphi}, \quad c_{\varphi T} = 0, \tag{6.96}$$

$$c_{T\varphi} = 0, \quad c_{TT} = \rho C \ + \frac{\partial H_w}{\partial T}. \tag{6.97}$$

Finally, prescribed nodal fluxes yields the right hand side vectors $\boldsymbol{J}_\varphi$ and $\boldsymbol{q}_T$ which can be defined as follows

$$\boldsymbol{f}_\varphi = \int_{\Gamma_J} \boldsymbol{N}_\varphi^{\mathrm{T}}\, \hat{\boldsymbol{J}}_\varphi \mathrm{d}\Gamma, \quad \boldsymbol{f}_T = \int_{\Gamma_q} \boldsymbol{N}_T^{\mathrm{T}}\, \hat{\boldsymbol{q}}_T \mathrm{d}\Gamma \ , \tag{6.98}$$

where $\hat{J}_\varphi$ denotes the mass boundary fluxes and $\hat{\boldsymbol{q}}_T$ denotes the heat boundary fluxes at nodes defined as follows

$$\hat{\boldsymbol{J}}_\varphi = \boldsymbol{N}_\varphi\, \overline{\boldsymbol{J}}, \quad \hat{\boldsymbol{q}}_T = \boldsymbol{N}_T\, \overline{\boldsymbol{q}}. \tag{6.99}$$

6.4 Coupling transfer and mechanical models

The spatial discretization of the balance equations is done by the finite element method [Bittnar and Šejnoha, 1996] and a system of ordinary differential equations with time variables is obtained. In the case of hydro-thermo-mechanical problem, the system may have the form

$$\begin{pmatrix} \boldsymbol{C}_{uu} & \boldsymbol{C}_{uT} & \boldsymbol{C}_{u\varphi} \\ \boldsymbol{C}_{Tu} & \boldsymbol{C}_{TT} & \boldsymbol{C}_{T\varphi} \\ \boldsymbol{C}_{\varphi u} & \boldsymbol{C}_{\varphi T} & \boldsymbol{C}_{\varphi\varphi} \end{pmatrix} \begin{pmatrix} \dot{\boldsymbol{d}}_u \\ \dot{\boldsymbol{d}}_T \\ \dot{\boldsymbol{d}}_\varphi \end{pmatrix} + \begin{pmatrix} \boldsymbol{K}_{uu} & \boldsymbol{K}_{uT} & \boldsymbol{K}_{u\varphi} \\ \boldsymbol{K}_{Tu} & \boldsymbol{K}_{TT} & \boldsymbol{K}_{T\varphi} \\ \boldsymbol{K}_{\varphi u} & \boldsymbol{K}_{\varphi T} & \boldsymbol{K}_{\varphi\varphi} \end{pmatrix} \begin{pmatrix} \boldsymbol{d}_u \\ \boldsymbol{d}_T \\ \boldsymbol{d}_\varphi \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_u \\ \boldsymbol{f}_T \\ \boldsymbol{f}_\varphi \end{pmatrix}, \tag{6.100}$$

where the subscript u denotes the displacements, the subscripts φ denotes the relative humidity and the subscript T denotes the temperature. The vectors $\boldsymbol{d}_u$, $\boldsymbol{d}_T$ and $\boldsymbol{d}_\varphi$ denote unknown nodal variables, the vectors $\boldsymbol{f}_u$, $\boldsymbol{f}_T$ and $\boldsymbol{f}_\varphi$ denote prescribed nodal forces and fluxes, the matrices $\boldsymbol{K}$ with subscripts denote the stiffness, conductivity and coupling matrices and the matrices $\boldsymbol{C}$ with subscripts denote the capacity and coupling matrices.

The system of differential equations (6.100) can be written more compactly in the form

$$\boldsymbol{C}(\boldsymbol{d})\dot{\boldsymbol{d}} + \boldsymbol{K}(\boldsymbol{d})\boldsymbol{d} = \boldsymbol{f} \ , \tag{6.101}$$

where the dependency of the stiffness, conductivity, capacity and coupling matrices on the attained values of variables is explicitly denoted. $\boldsymbol{d}$ and $\dot{\boldsymbol{d}}$ denote nodal variables and their time derivatives.

A staggered approach can be used in the solution of system (6.100) because the transport processes are not considered to be influenced by the mechanical analysis. Thus the coupling matrices $\boldsymbol{C}_{Tu}$, $\boldsymbol{C}_{\varphi u}$, $\boldsymbol{K}_{Tu}$ and $\boldsymbol{K}_{\varphi u}$ forms zero blocks. However, it should be emphasized that the heat and moisture transfer are fully coupled.

The system of equations (6.101) has to be solved by an incremental method. Time discretization is based on the v-form of the generalized trapezoidal method [Hughes, 1987] defined by the relationships

$$\boldsymbol{d}_{n+1} = \boldsymbol{d}_n + \Delta t \boldsymbol{v}_{n+\alpha} \ , \tag{6.102}$$

$$\boldsymbol{v}_{n+\alpha} = (1-\alpha)\boldsymbol{v}_n + \alpha\boldsymbol{v}_{n+1} \ , \tag{6.103}$$

where $\boldsymbol{v}$ denotes the first derivatives of nodal values with respect to time. The subscript n denotes the time step and it serves also as an index in the incremental method, called the outer iteration loop. It is assumed that all variables are known at the time t_n and variables at the time t_{n+1} are searched.

Substitution of expressions defined in Equations (6.102) and (6.103) to the system of differential equations (6.101) leads to relationship

$$(\boldsymbol{C}_n + \Delta t\alpha\boldsymbol{K}_n)\,\boldsymbol{v}_{n+1} = \boldsymbol{f}_{n+1} - \boldsymbol{K}_n\,(\boldsymbol{d}_n + \Delta t(1-\alpha)\boldsymbol{v}_n) \ , \tag{6.104}$$

where $\boldsymbol{C}_n$ and $\boldsymbol{K}_n$ denote the capacity and stiffness/conductivity matrices evaluated with the help of values $\boldsymbol{d}_n$. The system of algebraic equations (6.104) is generally nonlinear and the Newton-Raphson method [Crisfield, 1991], [Bittnar and Šejnoha, 1996] has to be used at each time step.

The trial solution $\boldsymbol{v}_{n+1,0}$ to the system of equations (6.104) is used for computation of the trial nodal values $\boldsymbol{d}_{n+1,0}$ which are obtained from Equations (6.102) and (6.103). Substitution of the trial solution back to the system of equations (6.104) with modified matrices does not generally lead to equality. An iteration loop, called the inner iteration loop, in every time step is based on residual which is computed from the relationship

$$\begin{aligned} \boldsymbol{r}_{n,j+1} &= \boldsymbol{f}_{n+1} - \boldsymbol{K}_n\,(\boldsymbol{d}_n + \Delta t(1-\alpha)\boldsymbol{v}_n) \\ &- (\boldsymbol{C}_{n,j} + \Delta t\alpha\boldsymbol{K}_{n,j})\,\boldsymbol{v}_{n,j} \ , \end{aligned} \tag{6.105}$$

where $\boldsymbol{C}_{n,j}$ and $\boldsymbol{K}_{n,j}$ denote the matrices evaluated for $\boldsymbol{d}_{n,j+1}$ and j is the index in the inner loop. Correction of nodal time derivatives are computed from the equation

$$(\boldsymbol{C}_{n,j} + \Delta t\alpha\boldsymbol{K}_{n,j})\,\Delta\boldsymbol{v}_{n,j+1} = \boldsymbol{r}_{n,j} \tag{6.106}$$

and new time derivatives are in the form

$$\boldsymbol{v}_{n,j+1} = \boldsymbol{v}_{n,j} + \Delta\boldsymbol{v}_{n,j+1}. \tag{6.107}$$

It has to be noted that the permanent recalculation of matrices $\boldsymbol{K}$ and $\boldsymbol{C}$ with respect to actual nodal values is very computationally demanding. In such a case, the matrix of the system of equations $\boldsymbol{C}(\boldsymbol{d}) + \Delta t\alpha\boldsymbol{K}(\boldsymbol{d})$ has to be always factorized and it requires additional computational time. The numerical examples show that the modified Newton method, which changes the system matrix only at the beginning of a new time step is the best choice.

6.5 Implementation of material models in SIFEL

In SIFEL, material models are connected with the integration points on elements. The computation of actual stiffness, conductivity and capacity matrices, as well as actual stress and flux computation proceeds at the level of integration points. The results from the integration points are passed to the element level where the numerical quadrature is used to obtain the resulting element matrices or vectors.

The material model for the transport part is fully coupled. It is not easy to separate such models into independent parts according to block scheme in (6.100) to be reused in the other model formulation. It led to the implementation of the models for transport processes as a single material model represented by a single class.

The different situation is in the case of mechanical models where advanced material models can often be composed of the material models describing single effect. Moreover, exploitation of the additive strain decomposition assumption is widely used. Therefore each basic material model describing one effect or group of related effects is implemented as a single class. Advanced models which take into account several effects are implemented as an artificial material model which manages contributions from the basic ones. In the case of mechanical material model for the simulation of concrete, there are class for basic material models such as elastic isotropic model, thermal expansion model, creep and shrinkage model and scalar isotropic damage model. Additionally, there is a class for the material model which is combination of creep and damage models. All models share the isotropic elastic models which can provide stiffness matrix for the actual elastic modulus and Poisson's ratio. Similarly, the strain passed to particular material models is decremented by the thermal strain component calculated at the thermal expansion model.

Each finite element in the mesh has array of the material model types and other similar array with identifiers of the instance of the given material model type with the given values of material model parameters. The first material model type is considered to be the master material model which knows the sequence of the basic material models from which is composed and it controls the all computations at the integration point level.

This decomposition principle is advantageous because particular basic models in the scope of advanced (master) material model can be changed simply at the runtime level and it requires no or minimum changes at the source code level if the new basic material model is added.

6.6 Modelling of the construction phases

Simulation of construction phases of concrete structures consists mainly of adding new structural parts. Each newly added structural part is usually stress-free, meaning that the deformed state at the interfaces between the current and new parts does not exert stress on the newly added parts. It is common to assume zero values of primary unknowns, i.e. the displacement vector components, in time-dependent mechanical problems because it is not easy to determine initial displacements by in situ measurements, and thus the zero values are usually a reasonable choice. Suppose added parts are in contact with the current ones. In such case, it is necessary to determine initial values of strains $\boldsymbol{\varepsilon}_i = \boldsymbol{\varepsilon}(t_0)$ at new elements on the interface, which must be subtracted from the total strains to obtain stress-free state at the time t_0 of insertion of new elements.

$$\boldsymbol{\varepsilon}(t) = \boldsymbol{B}\,\boldsymbol{d}(t), \tag{6.108}$$

$$\boldsymbol{\sigma}(t) = \boldsymbol{D}(t)\,(\boldsymbol{\varepsilon}(t) - \boldsymbol{\varepsilon}_i)\,. \tag{6.109}$$

There are several approaches how to attain a stress-free state on newly added parts of the structure in the FE code implementation, but the following two are the most used

- Lifetime function of elements and nodes - each element has time function controlling element birth and death, similar function is defined for nodes which controls DOFs defined at the given node.
- Time-changing material models - special master material model which allows for the switching among the given material models.

6.6.1 Lifetime functions of elements

The problem is defined by the finite element mesh where each element has a defined lifetime function controlling its birth and death, i.e. addition and removal. Similar function can be defined for nodes or the node lifetime may be controlled by the time functions of the adjacent elements. At the element birth, the displacement vector at element nodes contains either zero (initial) values at new nodes or values attained from the previous computation in the case of nodes on interface between old and new elements. These nodal displacement values are collected in vector $\boldsymbol{d}_i$ and stored for further usage. In the case of strain computation on that element, these stored displacements are considered to be initial ones and computation of strains and stresses changes as follows

$$\boldsymbol{\varepsilon}(t) = \boldsymbol{B}\,(\boldsymbol{d}(t) - \boldsymbol{d}_i)\,, \tag{6.110}$$

$$\boldsymbol{\sigma}(t) = \boldsymbol{D}(t)\,(\boldsymbol{\varepsilon}(t))\,. \tag{6.111}$$

The storage of initial displacements is advantageous over the storage of initial strains because strains are stored at the level of element integration points whose number may be significantly greater than the number of element nodes. Also the number of displacement vector components is usually lower than the ones of strains.

Moreover, the lifetime function is be involved in procedures assembling the stiffness matrix and internal force vector where only the active (live) elements can be taken into account. It results in significantly better computational performance because of the fewer active DOFs, and calls of computational procedures on elements are also reduced.

A weak point of this strategy is that the implementation requires modification of all procedures containing loops over elements where the conditional statements have to be added for the loop to process only active elements. Also, initial displacements of the newly added nodes are not known and thus set to zero usually which results in the incorrect deformed shape of the added part of the structure. If the deformed shape is not the objective of the analysis, this need not be a problem because the correct stress state is preserved, and similarly, this flaw does not influence the determination of the damage/crack pattern. The correct determination of the initial deformed shape of the added part is dealt with in Section 7.

6.6.2 Time-changing material models

Another approach how to handle the construction phases is the modelling of the structure as a whole where the parts which are not yet active have assigned an almost zero stiffness. The stiffness cannot be true zero because this would lead to zero rows in the system stiffness matrix. Usually, the elastic modulus is reduced significantly by a factor $10^{-4} \sim 10^{-6}$ but it may lead to the issues in the material models such as creep B3 model. More safe

approach is the using simple elastic isotropic model with reduced stiffness for non-active parts and at the moment of the part activation switch the material model to the desired one while the attained strains at activation time are stored and subtracted according to Equations (6.108)–(6.109).

The newly activated structure parts have automatically initial displacements at all nodes but the deformed shape is not correct usually and it is influenced by the neighbour non-active parts. It can be illustrated with the help of simple example of a beam gradual construction starting from the two opposite supports. The beam is assumed to be loaded by the dead weight load. The construction is divided into four phases and its progress is depicted in Figure 6.3 where the stages at the left column were calculated with correctly determined initial displacements and stages in the right column were calculated with the help of time switched material models. Figure 6.4 captures a diagram of the vertical

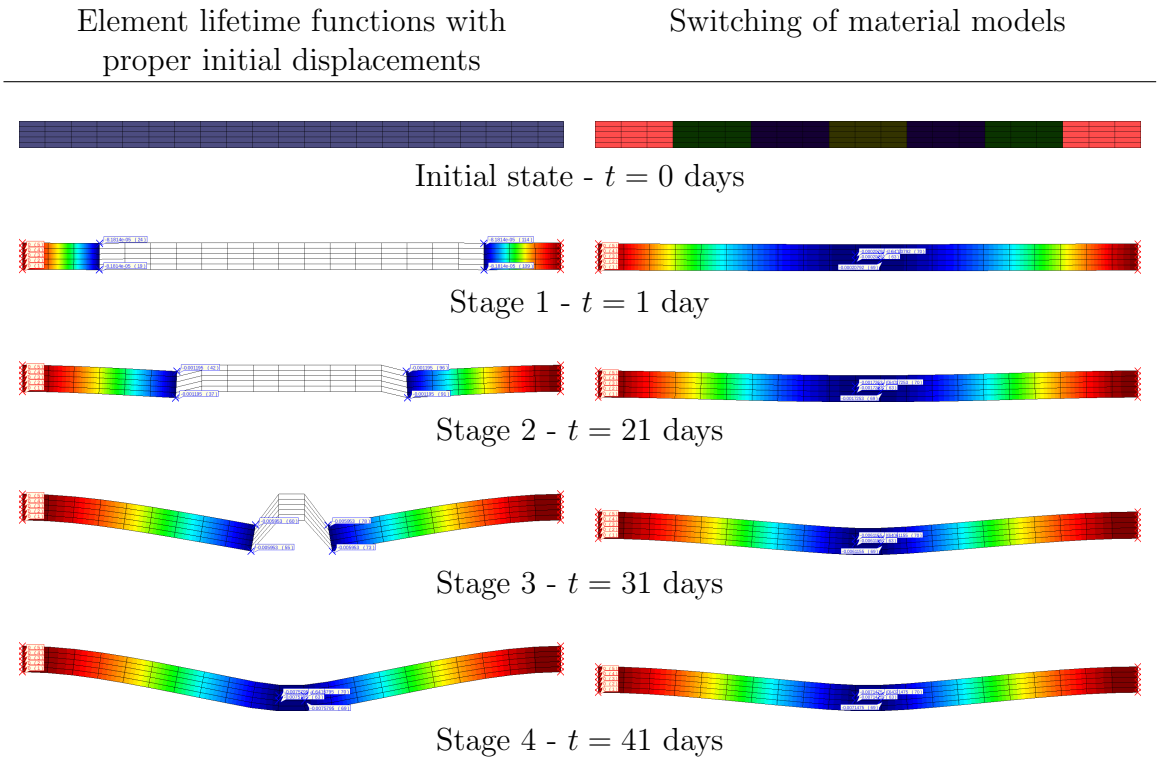


Figure 6.3: Time evolution of deformed shape of the gradually constructed beam.

displacement evolution at top surface nodes located at the ends of particular construction segments where the difference in vertical displacement of the final segment in the middle of the beam can be clearly distinguished.

From the implementation point of view, construction phases modelled by the time-changing material models require implementing a new material model, which usually requires the modification of fewer source codes than the implementation of lifetime functions. The performance is somewhat lower because the whole structure is always considered, and therefore the dimension of the system stiffness matrix is constant but at the maximum.

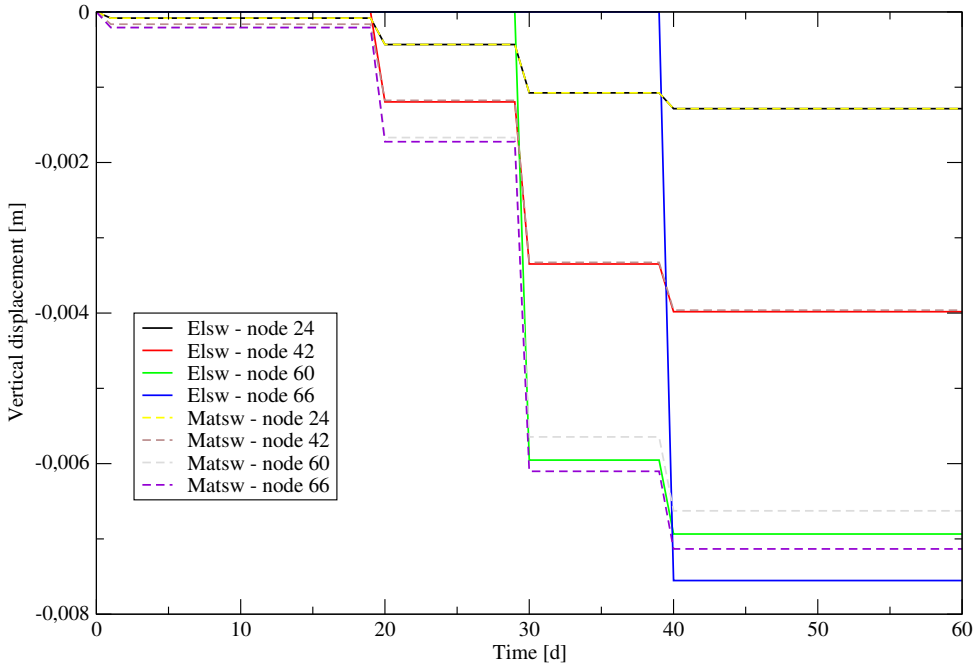


Figure 6.4: Time evolution of the vertical displacements. Solid lines denotes results from element life functions, dashed lines denotes results from switching of material models.

6.7 Problem description and analysis results

The simulation of the casting procedure of the foundation slab was addressed to the modelling of drying processes in young concrete. It demanded the choice of a physically correct model describing transport processes, real material parameters for heat and moisture transport (including the model of heat hydration evolution), a suitable damage model for concrete, and a set of correct initial and boundary conditions (climatic conditions).

The slab was created from two parts which were mutually shifted 1.3 m. The thickness of the slab was 1 m, and the spans were 15.0 and 15.8 m. On the boundaries were left shrinkage bands whose width was 1.5 m. The scheme of the slab is depicted in Figure 6.5. The slab was reinforced by 12 bars of reinforcement V25 per meter in longitudinal and transversal directions. There were also ties made from reinforcement V16, whose density was 9 pieces per square meter.

The slab was cast in three layers of a thickness of approximately 33 cm to avoid damage due to the generated hydration heat. The cast sections were watered for three days, and they were covered by polyethylene sheets after casting. The generation of hydration heat during the first several hours and damage evolution due to shrinkage and thermal strains were the reasons for the small time steps used at the beginning of the analysis. Damage evolution caused the increase of the norm of the unbalanced force vector in the Newton-Raphson method, and if the time step was not chosen carefully, convergence problems arose.

The computer simulation began 1 hour after the casting of the first layer. In the performed thermo-hydro-mechanical analysis, the Künzel-Kiessl's model was used for the modelling of transport processes, the B3 creep model and the scalar isotropic damage

model were used for the description of the mechanical behaviour. These material models were in SIFEL computer code complemented by a hydration heat model for concrete and a model of climatic condition effect based on statistically processed data of climatic conditions for the Prague region. Details about the climatic condition model can be found in [Grunewald, 2000] and [Maděra and Černý, 2005].

The slab was supported by springs at the bottom. The stiffness of the springs near the corners was increased to capture subsoil behaviour. A deadweight load was applied on the whole slab. An important role was played by the thermal boundary conditions used in the heat transfer analysis. In this case, thermal boundary conditions simulated the average aerial daily temperatures in June in the region.

The given thermo-hydro-mechanical coupled problems had very high demands on computational power. Because many material models were coupled together, extraordinarily high numbers of internal variables were stored in each integration point. The stored internal variables and the large matrix of the system of algebraic equations led to extremely high demands on computer memory. In this case, the memory space used for internal variables was comparable to that used for the system matrix. Considering the memory requirements, the 2D model of the problem was created even though the program can solve 3D problems, and the material models are derived for 3D too. Using 2D elements reduced the number of internal variables and unknowns. The reduced number of unknowns was also important for the factorisation speed of the equation system Equation (6.104). The factorization had to be performed once or several times at each time step depending on the results from the Newton-Raphson method that had to be used for the solution of the system (6.100) due to nonlinearities involved in the scalar isotropic damage model.

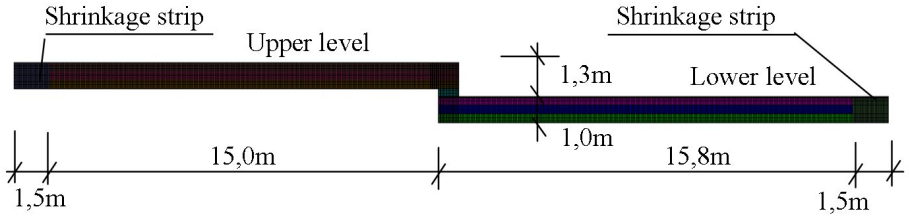


Figure 6.5: Dimensions of the 2D model and finite element mesh.

The sizes of finite elements used were about 4 cm in both directions except of the thin bottom and top layers where the mesh was twice finer in the transversal direction. The necessity of the finer mesh was given by increased temperature and humidity gradients in these layers and the consequent damage occurrence. Details of mesh decomposition were captured in Figure 6.6. The generated mesh considered the sequential casting procedure and the particular concrete layers were generated with different material properties. In Figures 6.5 and 6.6, these layers were filled by various colours.

B3 model was used for the creep and shrinkage description, which involved evolution of Young's modulus with respect to age of concrete while the scalar isotropic damage model assumed the material parameters to be constant. It was especially necessary to introduce a time dependent evolution of the tensile strength. In this case, the strength was assumed proportionally to the actual value of the elastic modulus

$$f_t(t) = cE(t), \tag{6.112}$$

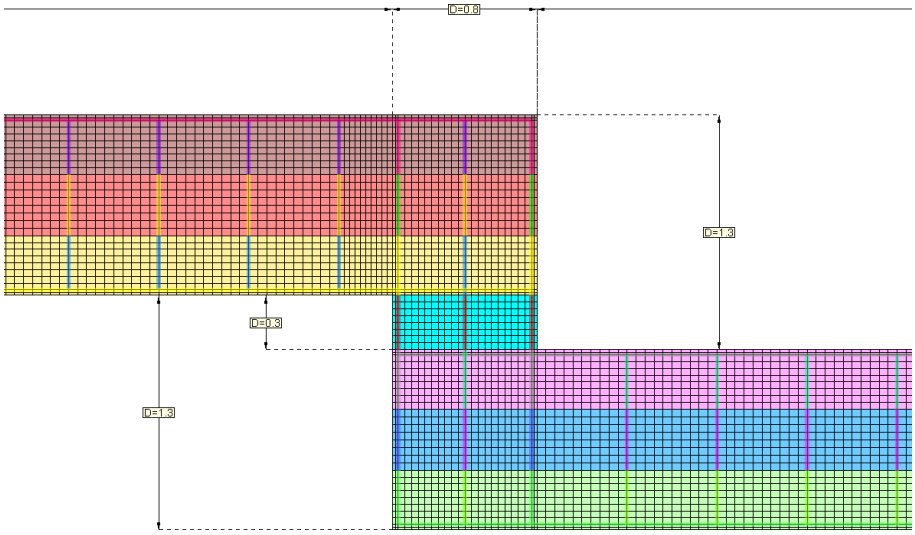


Figure 6.6: Detail of FE mesh near drop.

where c was the proportional factor and $E(t)$ was the value of time-dependent elastic modulus, which was calculated by the B3 model. The material parameters for concrete class C35/45 were used in the B3 model and c was set to 10^{-4}.

Two conclusions followed from the results of the coupled heat and moisture transfer analysis and from the simultaneous mechanical analysis. The first was that the accumulated hydration heat expired approximately after 7 days (Figure 6.7) simultaneously with autogeneous shrinkage phase. The second conclusion was that during the process of drying, the drop of moisture content and temperature occurred first in the surface layers and much later in the core. The effect of the diffusion process of drying (shrinkage of concrete) on the stress development and micro-cracks distribution was rather extensive and smeared cracks could cause the initialization of main cracks.

The following figures depict the resulting course of the normal stresses σ_x (Fig. 6.8), the shear stresses τ_{xy} (Fig. 6.9). The deformed shape of the structure (Figs. 6.10–6.12) and the damage parameter ω (Figs. 6.13–6.15) were captured for particular construction stages of the lower slab. The first two stages were captured shortly before the next layer addition. Results at the third stage corresponded to the state at the time 15 hours after the casting of the first layer. Detailed views of the damaged areas were captured in Figures 6.16 and Fig. 6.17.

Results of the presented analysis showed that damage evolved in three zones. The first zone was on the top of the slab, and it resulted from the drying processes. The second zone was on the slab side. The shear stresses caused this damage, and it was also observed in the realized structure. The third zone appeared at the bottom. It stemmed from the non-uniform distribution of temperature across the slab due to the sequential casting procedure. It should be noted that the computed analysis captured only the initial stage of the construction process, and it did not consider the upper structure's load. This load may also cause propagation of damage and cracks.

The results also confirmed that the correct modelling of the sequential construction influences the evolution of the damage parameter significantly. The distribution of nonzero

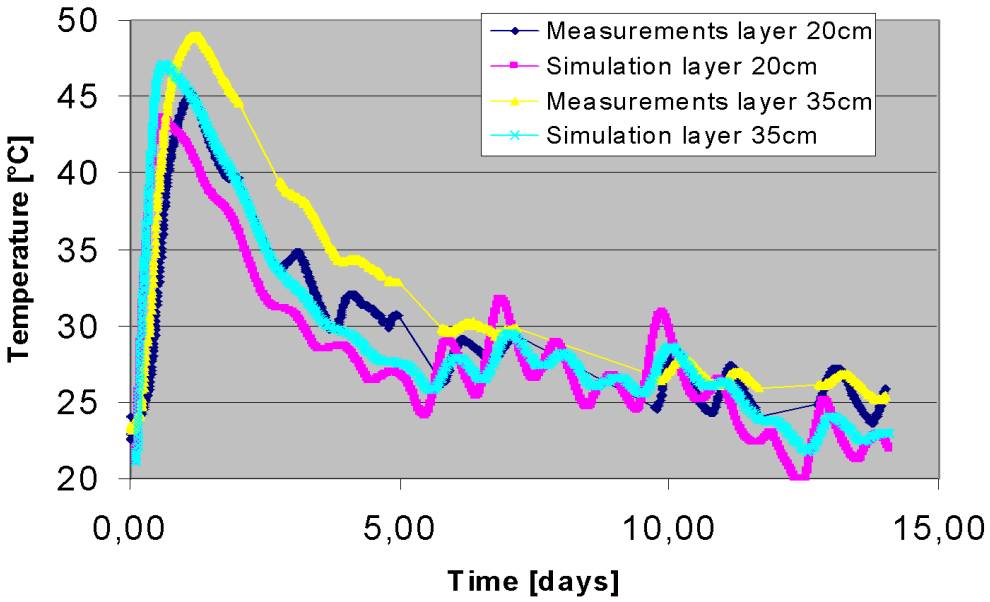


Figure 6.7: Temperature history.

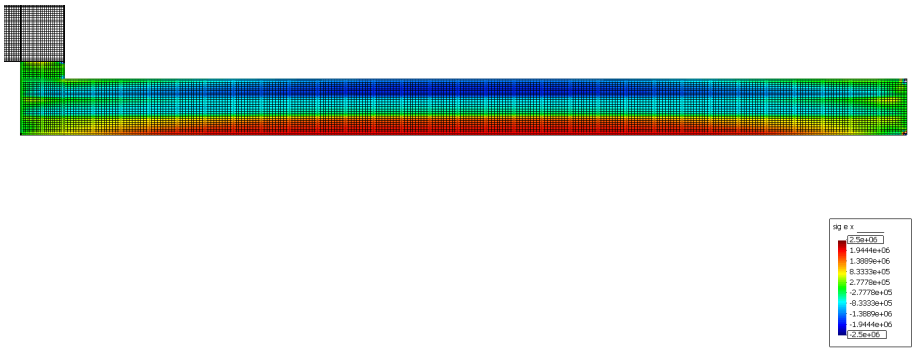


Figure 6.8: Distribution of stresses σ_x.

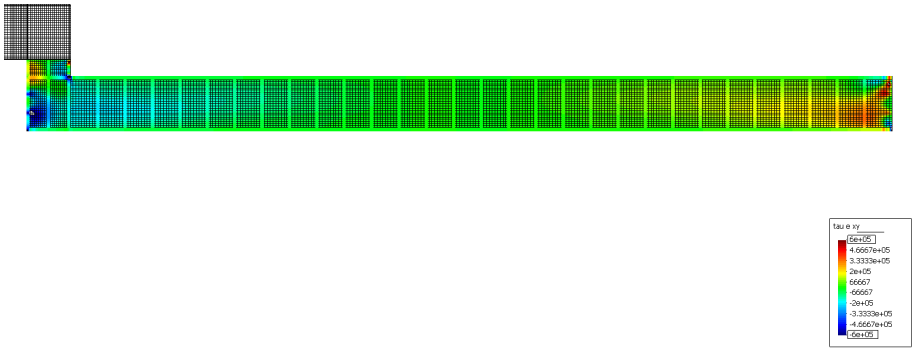


Figure 6.9: Distribution of stresses τ_{xy}.

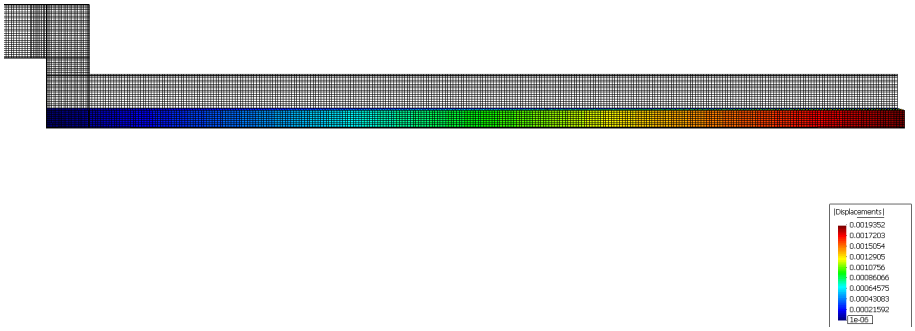


Figure 6.10: Deformed shape of the first layer of concrete.

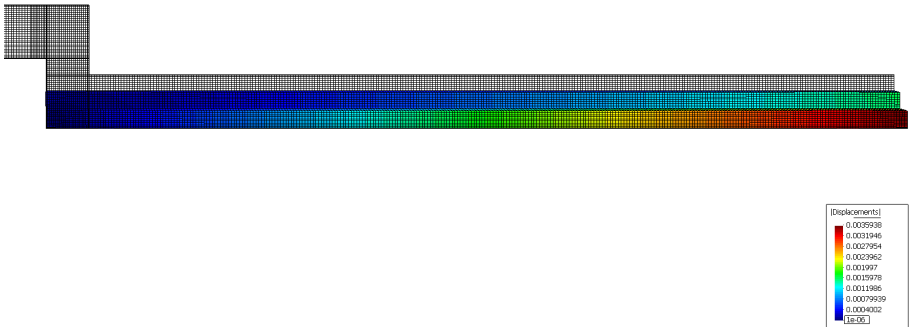


Figure 6.11: Deformed shape of the second layer of concrete.

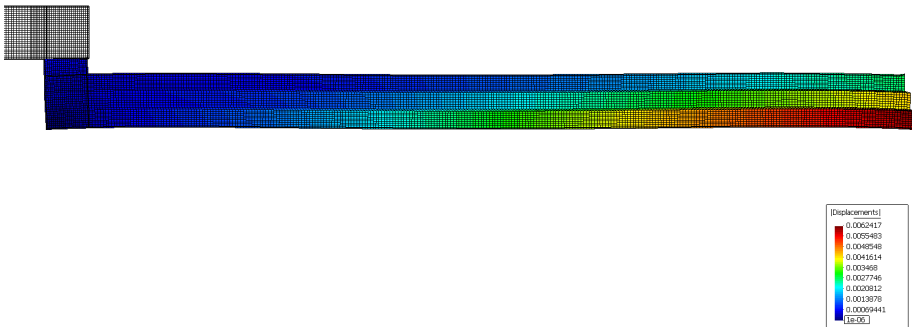


Figure 6.12: Deformed shape of the third layer of concrete.

values of the damage parameter could be seen on the bottom layer, which extended to 20 cm of its thickness. The maximum value of the damage parameter was 0.4. The damage was caused by hydration heat generation of the top layer, which was delayed when compared to the bottom layers. The peak of hydration heat generation in the top layer caused the nonuniform distribution of thermal strains, and consequently, the slab tended to deflect upward. In the middle of the slab, the influence of dead weight load dominated, and it led to damage to the bottom layer. The resulting deformed shape of the structure was captured in Figure 6.12.

The climate conditions are another factor causing damage. It can be observed in Figure 6.16 that the whole top surface is damaged but only to a shallow depth. The

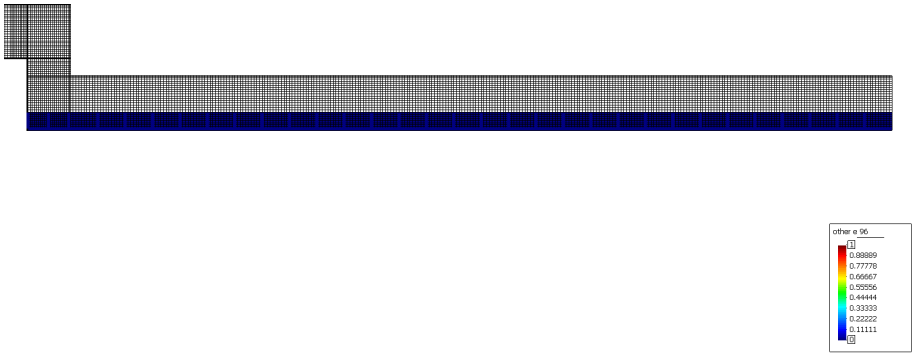


Figure 6.13: Distribution of the damage parameter ω in the first layer of concrete.

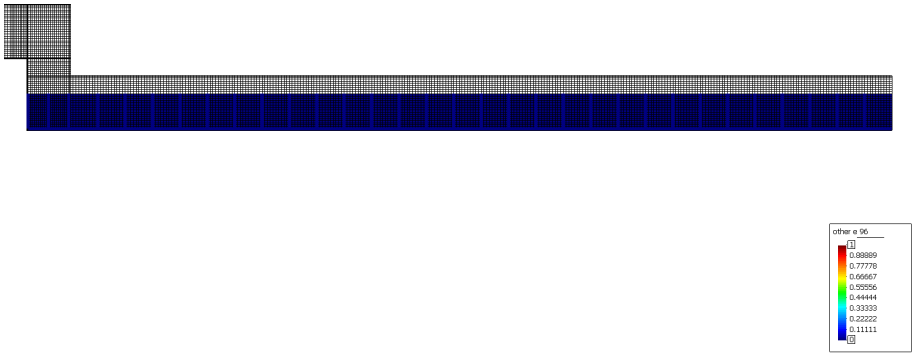


Figure 6.14: Distribution of the damage parameter ω in the second layer of concrete.

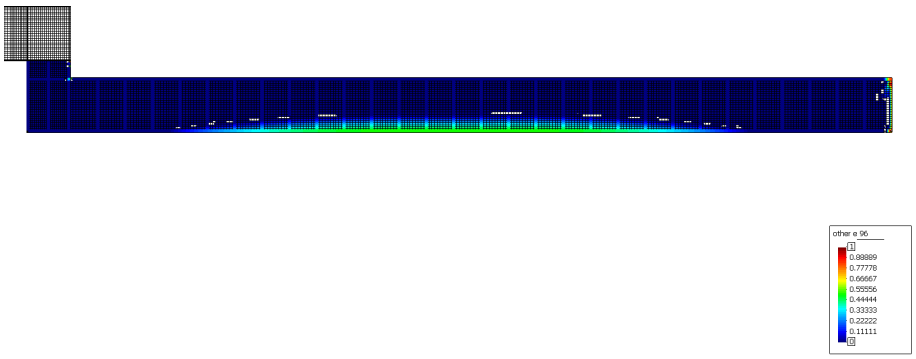


Figure 6.15: Distribution of the damage parameter ω in the third layer of concrete.

damage was caused by drying shrinkage which was intensified by the applied climatic conditions. The last area with significant damage evolution is at the top right corner of the slab (see Figure 6.17). In this case, the damage was caused by shear stresses whose concentration at the corner can be observed in Figure 6.9.

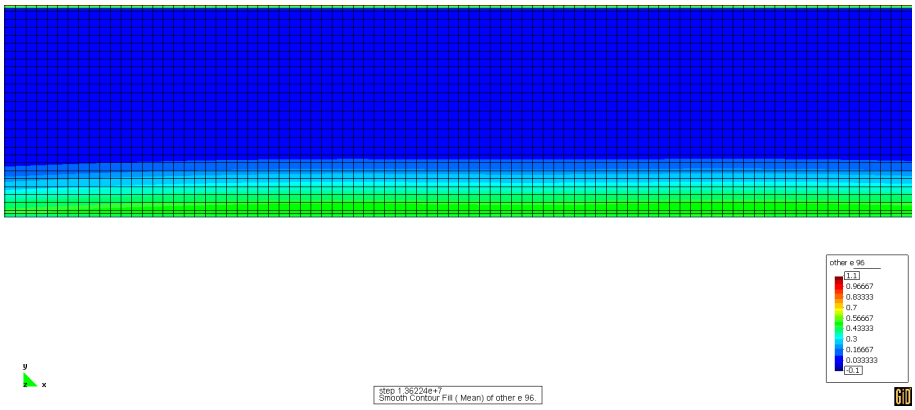


Figure 6.16: Distribution of the damage parameter ω in the middle of the slab.

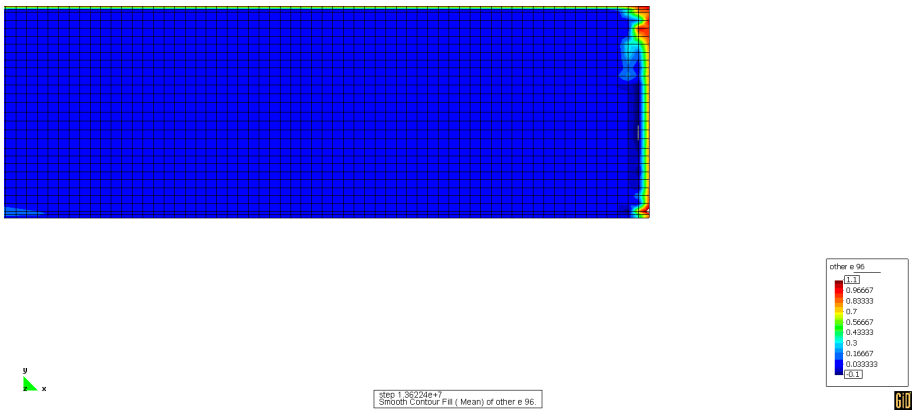


Figure 6.17: Distribution of the damage parameter ω in the right corner of the slab.

Chapter 7

Creep analysis of the bridge in Mělník

Project TA01030733 of the Czech Technology Agency was focused on concrete bridges, and it had the following objectives:

- Development of a methodology for the designing of new road bridges,
- monitoring existing bridge structures,
- predicting the behaviour of bridges showing excessive deflections due to creep and relaxation.

The author's team was involved in the last objective, where the creep behaviour of a selected existing bridge structure should be analysed, and the result of the objective should be a software tool allowing for the bridge analysis. The bridge model should be three-dimensional, and a detailed creep model for the concrete and relaxation model of the tendons should considered. At the beginning of the project solution, it was decided to analyse the road bridge at Mělník. It was a box girder bridge built at the beginning of the 90s using the balanced cantilever construction method with cast-in-place segments supported by form travellers. The bridge exhibited higher deflections than expected, and thus the extension of the SIFEL code was focused on the simulation and prediction of the bridge deflection. Non-mechanical effects such as temperature or moisture transfer were neglected because they influence creep evolution in the early stage rather, while the model was focused on long-term behaviour.

Structural models of the bridge structures used in common engineering practice are frequently based on the beam theory assumption of the cross-section planarity after deformation, which cannot capture many important phenomena violating this assumption. For example, it is known that there is a shear lag effect on box girder cross-sections which is manifested by the section warping and non-uniform normal stress distribution. This effect can be observed on the flanges (decks) of the box girder sections, and it is caused by the shear strains in their plane. Similarly, the effect of a tendon anchorage may lead to the warping of a cross-section.

The proposed tools were based on detailed creep and relaxation models whose calibration was considered with the help of measured data from the existing bridge structures

that were reconstructed in the past. Modern optimization methods and stochastic sensitivity analysis tools were used to calibrate and identify the parameters of the numerical models.

The 3D model of the bridge was solved by the finite element method, where hexahedron 3D elements with linear approximation were used for the concrete body of the bridge. Tendons were modelled by 3D bar elements with a linear approximation.

7.1 Mechanical material models

Prestressed concrete bridges with large spans and carrying huge loads represent structures, where the issue of permanent growth of deformations over time is highly relevant in terms of serviceability, long-term reliability and durability. Experience shows that there are often greater deflection values compared to the predicted/computed ones and that they are increasing over time. These deflections are significantly influenced by the prestressing losses due to creep and shrinkage. Another significant influence represents tendon relaxation. Both these effects were taken into account in the proposed mechanical model.

7.1.1 Creep model for concrete

The Bazant's B3 creep model seemed the most plausible at that time and therefore it was used in the performed analysis. Alternatively, the model proposed in ČSN EN was considered. The B3 creep model was described in details in Section 6.2.1. The effects of temperature and moisture transfer were not considered in the analysis and thus they were assumed to be constant in time. This assumption led to the zero stress induced contributions to the shrinkage and thermal strains and Equation (6.57) was transformed into (6.56). The mean drying shrinkage was defined according to [Bazant and Chern, 1985] by the following empirical relations

$$\varepsilon_{sh}(t, t_0) = -\varepsilon_{sh\infty}\, k_h\, S(t - t_0), \tag{7.1}$$

where $\varepsilon_{sh\infty}$ is the ultimate shrinkage, and k_h is the factor of humidity dependence and $S(t)$ is the time curve which depends on the duration of the drying $t - t_0$. The ultimate shrinkage is given by the relation

$$\varepsilon_{sh\infty} = \alpha_1\, \alpha_2 \left[\frac{0.019\ (w_c\, c_s)^{2.1}}{(f_c)^{0.28}} + 270.0 \right] \frac{E(7 + 600)}{E(t_0 + \tau_{sh})}. \tag{7.2}$$

In the above equation, α_1 (–) is the coefficient of cement type, α_2 (–) is the coefficient for curing, w_c represent water-cement ratio (–), c_s is the cement content (kg/m^3), f_c is the compressive strength of concrete (MPa) and t_0 is the time when drying begins. In this case, coefficients $\alpha_1 = 1.05$ and $\alpha_2 = 1.2$ were considered with respect to cement type and normal curing conditions. Time dependent elastic modulus denoted by $E(t)$ can be obtained either as

$$E(t) = 1/J(t, \tau), \tag{7.3}$$

or approximately with the help of ACI relation

$$E(t) = E(28)\sqrt{\frac{t}{4+0.85t}}, \tag{7.4}$$

where t is the concrete age in days. Quantity τ_{sh} is called shrinkage halftime and indicates time when ε_{sh} attains a half of final value approximately. It can be defined as

$$\tau_{sh} = k_t \left(k_s D\right)^2, \tag{7.5}$$

where k_t (d/m^2) is the factor dependent on concrete diffusivity, k_s () is the the cross-sectional shape factor and D represents the effective cross-sectional thickness. The effective cross-section area can be defined with the help of ratio between structure volume V_S and the exposed structure surface S_E as

$$D = \frac{2V_S}{S_E}. \tag{7.6}$$

Factor k_t can be estimated by the formula

$$k_t = \frac{0.085}{t^{0.08}\,(f_c)^{0.25}}. \tag{7.7}$$

Shape factor, k_s, ranges in [1.0; 1.55] where the lowest value is considered for slabs and the highest value for cubes. With respect to the box girder cross-section, $k_s = 1.0$ was applied in the analysis. The factor of humidity, k_h, can be defined as

$$k_h = \begin{cases} 1-\varphi^3 \text{ for } \varphi \leq 0.98 \\ -0.2 \text{ for } \varphi = 1 \text{ (swelling in water)} \\ \text{linear interpolation for } 0.98 \leq \varphi \leq 1. \end{cases} \tag{7.8}$$

The time curve, $S(t)$, is an increasing function which describes evolution of the normalized shrinkage strain in a perfectly dry environment. The function is defined according [Baweja and Bažant, 1995] as

$$S(t) = \tanh\left(\sqrt{\frac{t-t_0}{\tau_{sh}}}\right). \tag{7.9}$$

It should be noted that time t is assumed in days in the above mentioned creep relations. The compliance function is extended by the additional term due to drying, J_d, that can be defined as

$$J_d(t,\tau) = q_5\sqrt{\exp(-g(t) - \exp(-g(\tau)))}, \tag{7.10}$$

$$g(t) = 8\left[1-(1-\varphi)S(t)\right]. \tag{7.11}$$

and the resulting compliance function can be written as

$$J(t,\tau) = q_1 + q_2 Q(t,\tau) + q_3 \ln\left[1+\left(\frac{t-\tau}{\lambda_0}\right)^n\right] + q_4 \ln\left(\frac{t}{\tau}\right) + J_d(t,\tau). \tag{7.12}$$

7.1.2 Tendon relaxation model

The influence of prestressing losses on the stress state and deformation may be very significant because the magnitude of the resulting internal forces due to the prestressing is large and comparable with the magnitude of forces due to load. Therefore determination of correct prestressing losses plays an important role. Two basic kinds of prestressing losses can be distinguished:

- Technological (short-term) losses - change of the prestressing on post-tensioned structures, which can be observed between the tensioning and jacking anchorage. The losses due to tendon friction and anchorage slip represent the most significant ones. Generally, they are influenced by the used tensioning system. These values should be determined by either producer of the given tensioning system or/and measurements of a tendon elongation during the tensioning. These losses are usually well-defined.

- Long-term losses - these losses are manifested after the tendon anchorage until the time under consideration or end of structure service life. These losses depend on the rheological properties of the given materials – creep and shrinkage in the case of concrete and steel relaxation.

Generally, the relaxation effect is manifested as a gradual decrease of stress in the specimen subjected to constant strain. Magnitude and time evolution depend on the material properties of used steel, the tendons' production process and anchorage technology. There is also a dependency of the relaxation on temperature, but this effect was not considered in the analysis.

In this case, the model introduced in EN 1992-1-1 was adopted. The model allows for determining the prestressing losses with respect to the tendon types and duration of the relaxation process.

Prestressing losses, $\Delta\sigma_{p,r}$ for the normal relaxation of tendons can be defined as

$$\Delta\sigma_{p,r} = \sigma_{p0}\, 43.12\, \exp\left(\frac{6.7\,\sigma_{p0}}{f_{pk}}\right)\left(\frac{t}{1000}\right)^{0.75\left(1-\frac{\sigma_{p0}}{f_{pk}}\right)}\, 10^{-5}, \tag{7.13}$$

where σ_{p0} is the initial presstresing, f_{pk} is the characteristic strength of tendon material. Similarly, prestressing losses for low relaxation tendons are given as

$$\Delta\sigma_{p,r} = \sigma_{p0}\, 1.65\, \exp\left(\frac{9.1\,\sigma_{p0}}{f_{pk}}\right)\left(\frac{t}{1000}\right)^{0.75\left(1-\frac{\sigma_{p0}}{f_{pk}}\right)}\, 10^{-5}. \tag{7.14}$$

Figure 7.1 captures a time evolution of the ratio of prestressing losses to the initial prestressing for low relaxation tendons for different rations of the initial prestressing, σ_{p0} and tendon material strength f_{pk}. The model was implemented in generalized form in terms of coefficients a_p, b_p and c_p

$$\Delta\sigma_{p,r} = \sigma_{p0}\, a_p\, \exp\left(b_p\, c_p\right)\left(\frac{t}{1000}\right)^{0.75(1-c_p)}. \tag{7.15}$$

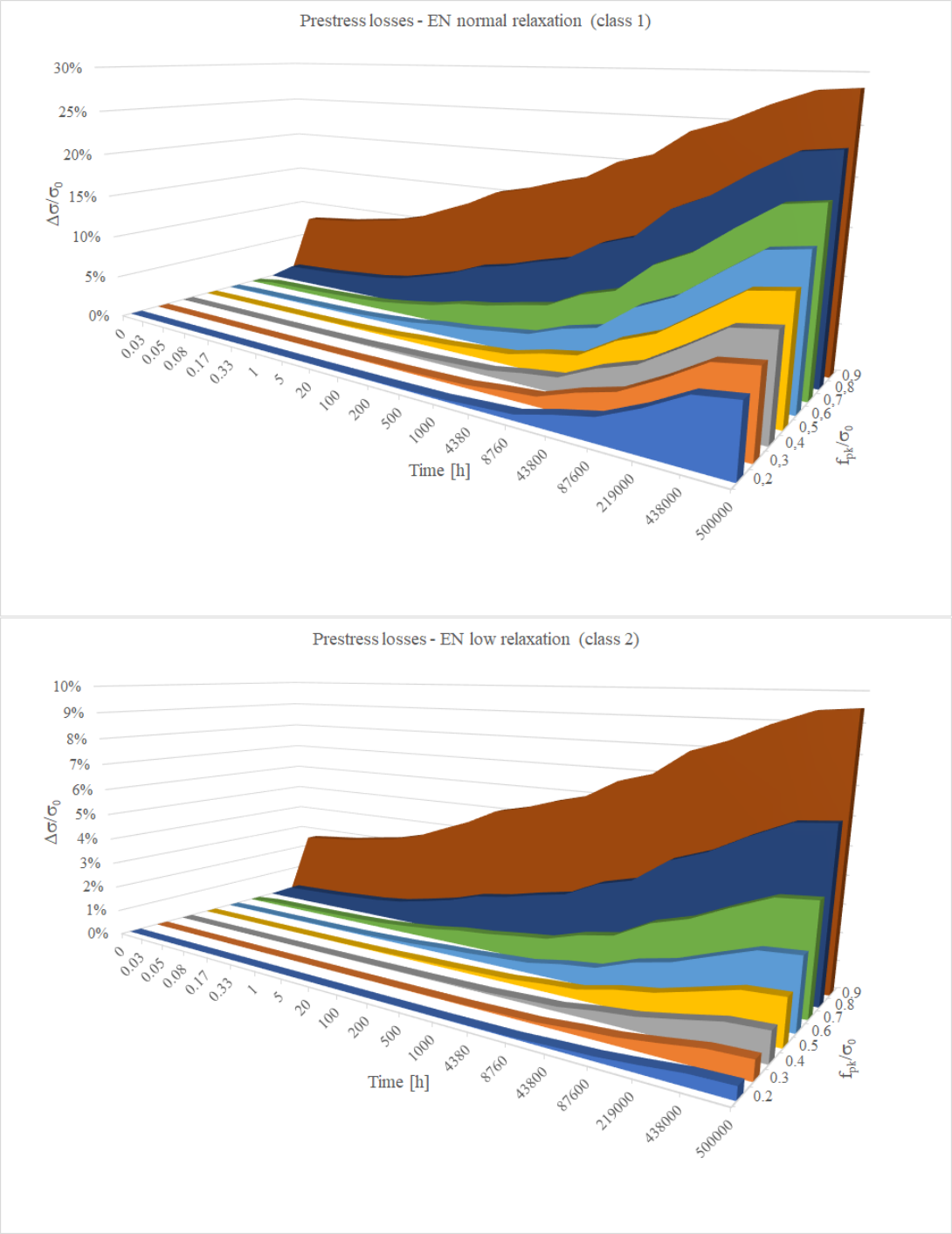


Figure 7.1: Time evolution of prestressing losses ratio $\Delta\sigma_{p,r}/\sigma_{p0}$ for normal relaxation (top) and low relaxation (bottom) tendons.

Coefficients a_p and b_p can be set either according to an in-situ measurement or set to the corresponding values that in Equations (7.13) and (7.14). Coefficient c_p represents a ratio between initial prestressing and tendon material strength. It should be noted that time, t, should be considered in hours in Equations (7.13)-(7.15).

The resulting stress-strain relation for the tendon relaxation model in a 1D stress-strain state can be written as

$$\sigma = \sigma_{p0} - \Delta\sigma_{p,r} + E_p\varepsilon, \tag{7.16}$$

where E_p is the elastic modulus of the tendon material.

7.2 Modelling of tendons

A direct approach to tendon modelling, where bar elements are directly connected to the 3D element nodes, would lead to a very high FE mesh density and, thus, to high computational demands. It stems from the fact that tendons are placed close to each other in section webs and flanges. For this reason, the hanging node concept was adopted to model tendons.

Generally, displacements can be approximated in arbitrary points of a hexahedron element by the following relation

$$\boldsymbol{u}(\boldsymbol{x}_n) = \boldsymbol{N}_u(\boldsymbol{x}_n)\boldsymbol{d}_e, \tag{7.17}$$

$$\boldsymbol{N}_u(\boldsymbol{x}_n) = (\boldsymbol{N}_{u1}(\boldsymbol{x}_n), \boldsymbol{N}_{u2}(\boldsymbol{x}_n), \boldsymbol{N}_{u3}(\boldsymbol{x}_n), \ldots, \boldsymbol{N}_{u8}(\boldsymbol{x}_n)) \tag{7.18}$$

$$\boldsymbol{N}_{ui}(\boldsymbol{x}_n) = \begin{pmatrix} N_i(\boldsymbol{x}_n) & 0 & 0 \\ 0 & N_i(\boldsymbol{x}_n) & 0 \\ 0 & 0 & N_i(\boldsymbol{x}_n) \end{pmatrix} \tag{7.19}$$

$$\tag{7.20}$$

where $\boldsymbol{x}_n = (\xi, \eta, \zeta)$ stands for natural coordinates of the given point, $\boldsymbol{N}_u$ is the matrix of shape functions, and $\boldsymbol{d}_e$ is the nodal displacement vector of the given element. N_i represents the shape functions of the i-th element node, which can be defined as

$$N_i(\boldsymbol{x}_n) = \frac{1}{8}(1 + \xi_i\,\xi)(1 + \eta_i\,\eta)(1 + \zeta_i\,\zeta). \tag{7.21}$$

In the hanging node concept, nodes of rebar or tendon elements are not defined directly with the help of standalone nodes that constitute independent DOFs. However, these nodes can be defined arbitrarily in the domain of surrounding 3D elements. Their displacements can be expressed in terms of element nodal displacements, where the given node lays, according to Equation (7.17). Thus the rebar node is 'hung' on nodes of its master element.

Equation (7.17) requires the natural coordinates, $\boldsymbol{x}_n$ of the given hanging node to be determined. Let the hanging node be defined in the global coordinate system, and then

the same approximation for that spatial coordinates may be used

$$x_g = f_{gx}(\boldsymbol{x}_n) = \sum_{i=1}^{8} N_i(\boldsymbol{x}_n) x_{e,i}, \tag{7.22}$$

$$y_g = f_{gy}(\boldsymbol{x}_n) = \sum_{i=1}^{8} N_i(\boldsymbol{x}_n) y_{e,i}, \tag{7.23}$$

$$z_g = f_{gz}(\boldsymbol{x}_n) = \sum_{i=1}^{8} N_i(\boldsymbol{x}_n) z_{e,i}, \tag{7.24}$$

where x_g are known coordinates of the hanging in the global coordinate system and $\boldsymbol{x}_e$ are known global nodal coordinates of the master element. The unknown natural coordinates $\boldsymbol{x}_n$ can be determined from the inverse relation, which leads to the system of 3 nonlinear equations, see definition of N_i. The system can be solved with the help of the Newton-Raphson method, whose $k+1$-th step can be expressed as

$$\boldsymbol{x}_n^{(k+1)} = \boldsymbol{x}_n^{(k)} + \left(\boldsymbol{J}^{(k)}\right)^{-1} {\boldsymbol{x}_r}^k, \tag{7.25}$$

where $\boldsymbol{J}$ is the Jacobi's matrix defined as follows

$$\boldsymbol{J} = \begin{pmatrix} \frac{\partial f_{gx}}{\partial \xi}\left(\boldsymbol{x}_n^{(k)}\right) & \frac{\partial f_{gx}}{\partial \eta}\left(\boldsymbol{x}_n^{(k)}\right) & \frac{\partial f_{gx}}{\partial \zeta}\left(\boldsymbol{x}_n^{(k)}\right) \\ \frac{\partial f_{gy}}{\partial \xi}\left(\boldsymbol{x}_n^{(k)}\right) & \frac{\partial f_{gy}}{\partial \eta}\left(\boldsymbol{x}_n^{(k)}\right) & \frac{\partial f_{gy}}{\partial \zeta}\left(\boldsymbol{x}_n^{(k)}\right) \\ \frac{\partial f_{gz}}{\partial \xi}\left(\boldsymbol{x}_n^{(k)}\right) & \frac{\partial f_{gz}}{\partial \eta}\left(\boldsymbol{x}_n^{(k)}\right) & \frac{\partial f_{gz}}{\partial \zeta}\left(\boldsymbol{x}_n^{(k)}\right) \end{pmatrix}, \tag{7.26}$$

and $\boldsymbol{x}_r$ is the residual vector defined as

$$\boldsymbol{x}_r = \boldsymbol{x}_g - \boldsymbol{f}_g\left(\boldsymbol{x}_n^{(k)}\right), \tag{7.27}$$

$$\boldsymbol{x}_g^{\mathrm{T}} = (x_g, y_g, z_g), \tag{7.28}$$

$$\boldsymbol{f}_g^{\mathrm{T}}\left(\boldsymbol{x}_n^{(k)}\right) = \left(f_{gx}\left(\boldsymbol{x}_n^{(k)}\right), f_{gy}\left(\boldsymbol{x}_n^{(k)}\right), f_{gz}\left(\boldsymbol{x}_n^{(k)}\right)\right). \tag{7.29}$$

For user convenience, the tendons are defined by the minimum points needed for capturing the tendon's geometrical shape. There is a tool MIDAS [Svoboda, 2012] for an automatic discretization of tendons with respect to their intersections with elements of a 3D mesh of the bridge concrete body. The input for this tool represents the 3D mesh of the bridge and the coarse geometry of tendons. A new fine mesh of 3D bar elements is generated, whose end nodes are represented by the calculated intersections. For each node of the new fine mesh, the corresponding 3D element is determined together with the natural coordinates of the node on the detected element, and thus the new nodes can be defined as hanging nodes.

7.3 Gradual construction process of the girder beam bridge

The balanced cantilever construction method with cast-in-place segments supported by form travellers represents a very complex problem from the numerical analysis point of

view. The static scheme at particular construction stages changes along the casting procedure and the tendon prestressing process. The construction process significantly influences the resulting deflections, and therefore, it must be considered. Contrary to the simulation of the casting procedure of foundation slab in Těšnov, the resulting displacements play a significant role, and thus the correct determination of initial displacements is the crucial point.

Each new cast segment of the bridge represents a set of finite elements in the mesh, and initial displacements at their nodes have to be determined. Some new elements may be connected with the current ones by shared nodes on the interface between the current and new segments. If such interface nodes exist, their attained displacements are stored as initial ones as proposed in Section 6. Initial values of displacements of the remaining nodes on the new segment can be calculated by the solution to the substructure consisting of the new segment where the Dirichlet boundary conditions are applied in the form of attained displacements at the interface nodes. The elements in the new active substructure can be identified with the help of element life functions. Let the life function of the i-th element be denoted by l_i then it can be defined as follows

$$l_i(t) = \begin{cases} 1 & \text{if } i\text{-th element is active} \\ 0 & \text{if } i\text{-th element is not active.} \end{cases} \tag{7.30}$$

Table 7.1 summarizes the algorithm of initial displacement calculation at the beginning of new time step at time $t + \Delta t$. In Table 7.1, i-th element of the whole domain solved Ω is denoted by e_i while the i-th node of the domain Ω is denoted by the n_i.

The progress of balanced cantilever construction method is accompanied by the deflections of cast segments that increase gradually toward the free ends. These deflections can be avoided or mitigated either by increase of tendon prestressing or by deck elevation of new segment which is realized via traveller tilting. The deck elevation is another effect which should be taken into account in the construction process modelling.

Bridge deck elevation can be simulated by the setting of initial displacements of the new bridge segment where the initial displacement calculation is enhanced by the application of vertical displacements at set of nodes on the bridge deck denoted by Γ_θ. Their values are determined from the given axis of tilting and the tilt angle θ where axis is given by two nodes A and B on the interface between current and new segment. Rotation of the node N about the axis AB form its original position to the rotated one denoted by N_θ is depicted in Figure 7.2. Calculation of the initial displacements at one selected node of the new segment deck is summarized in Table 7.2 where $\boldsymbol{a} \cdot \boldsymbol{b}$ denotes the inner product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, $\|.\|$ denotes the Euclidean norm and $\boldsymbol{a} \times \boldsymbol{b}$ denotes the vector product of $\boldsymbol{a}$ and $\boldsymbol{b}$.

Complete algorithm of the initial displacement determination is summarized in Table 7.3. The influence of the initial displacement determination can be demonstrated on a simple example of gradual construction process inspired by the balanced cantilever method. Figure 7.3 captures FE whole mesh of the problem. The structure is divided into segments denoted by indices 1–8. The construction process starts at the starter segment 1 and proceeds by the building of segments simultaneously on the left and right from the starter in particular stages of the construction process. Initial displacements of segments 2–4 were were calculated according to the algorithm in Table 7.1 with no tilting. Initial displacements of segments 5–7 were calculated according to the algorithm in Table 7.3

1	Identify set of new elements, Ω_N $\Omega_N = \{e_i \in \Omega \mid l_i(t+\Delta t) > 0 \ \wedge \ l_i(t) = 0\}$
2	Identify set of current elements, Ω_o $\Omega_o = \{e_i \in \Omega \mid l_i(t+\Delta t) > 0 \ \wedge \ l_i(t) > 0\}$
3	Identify set of nodes on interface between current elements and new elements, Γ_I $\Gamma_I = \{n_i \in \Omega \mid n_i \in \{e_i \in \Omega_o\} \ \wedge \ n_i \in \{e_i \in \Omega_N\}\}$
4	Save initial displacements at nodes $n_i \in \Gamma_I$
5	Generate DOF numbers for all nodes $n_i \in \{e_i \in \Omega_N\} \wedge n_i \notin \Gamma_I$
6	Assemble vector of prescribed initial displacements on Ω_N, $\boldsymbol{d}_i^{\mathrm{T}} = \left(\boldsymbol{d}_z, \bar{\boldsymbol{d}}_i\right)$, where $\bar{\boldsymbol{d}}_i$ is the vector of saved initial displacements at Γ_I, $\boldsymbol{d}_z = \boldsymbol{0}$, $\boldsymbol{d}_z \in \mathbb{R}^{nd}$ where nd is the maximum DOF number on the domain Ω_n.
7	Initialize material models on Ω_N.
8	Compute right hand side vector $\boldsymbol{f}_i = -\int_{\Omega_N} \boldsymbol{B}^{\mathrm{T}} \boldsymbol{\sigma}_i \mathrm{d}\Omega_N$, where $\boldsymbol{\sigma}_i = \boldsymbol{D}_i \, \boldsymbol{\varepsilon}_i$, $\boldsymbol{\varepsilon}_i = \boldsymbol{B} \, \boldsymbol{d}_i$ and $\boldsymbol{D}_i$ is the initial material stiffness matrix.
9	Assemble stiffness matrix of the reduced problem, $\boldsymbol{K}_n = \int_{\Omega_N} \boldsymbol{B}^{\mathrm{T}} \boldsymbol{D}_i \boldsymbol{B} \mathrm{d}\Omega_n$
10	Solve problem $\boldsymbol{K}_n \, \boldsymbol{d}_0 = \boldsymbol{f}_i$ for unknown vector $\boldsymbol{d}_0$
11	Save initial displacements from vectors $\boldsymbol{d}_0$ and $\bar{\boldsymbol{d}}_i$ on elements of Ω_N.

Table 7.1: Algorithm of initial displacement calculation.

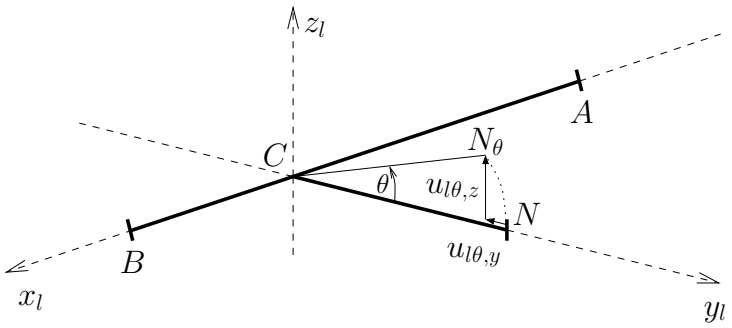


Figure 7.2: Scheme of the tilting of the node N.

1	Calculate origin of the local coordinate system, C, for the rotation of node N $C = \dfrac{\overrightarrow{AB} \cdot \overrightarrow{AN}}{\left\|\overrightarrow{AB}\right\|^2}\overrightarrow{AB} + A$
2	Assemble transformation matrix $\boldsymbol{T}_\theta$ $\boldsymbol{l}_x = \dfrac{\overrightarrow{AB}}{\left\|\overrightarrow{AB}\right\|}$, $\boldsymbol{l}_y = \dfrac{\overrightarrow{CN}}{\left\|\overrightarrow{CN}\right\|}$, $\boldsymbol{l}_z = \dfrac{\boldsymbol{l}_x \times \boldsymbol{l}_y}{\|\boldsymbol{l}_x \times \boldsymbol{l}_y\|}$ $\boldsymbol{T}_\theta^{\mathrm{T}} = (\boldsymbol{l}_x, \boldsymbol{l}_y, \boldsymbol{l}_z)$
3	Calculate initial displacements due to rotation in the local coordinate system $\boldsymbol{u}_{l\theta}^{\mathrm{T}} = \left(0,\ (\cos\theta - 1)\left\|\overrightarrow{CN}\right\|,\ \sin\theta\left\|\overrightarrow{CN}\right\|\right)$
4	Transformation of the initial displacement vector, $\boldsymbol{u}_{l\theta}$, to the global coordinate system $\boldsymbol{u}_{g\theta} = \boldsymbol{T}_\theta^{\mathrm{T}}\boldsymbol{u}_{l\theta}$
5	Add linear approximation of the initial displacements $\boldsymbol{u}_A$ and $\boldsymbol{u}_B$ at the interface nodes A and B to $\boldsymbol{u}_{g\theta}$ $\Longrightarrow$ resulting initial displacement vector,$\boldsymbol{u}_i$, at node N $\boldsymbol{u}_i = \boldsymbol{u}_{g\theta} + (\boldsymbol{u}_B - \boldsymbol{u}_A)\left(\boldsymbol{l}_x \cdot \overrightarrow{AC}\right)\dfrac{1}{\left\|\overrightarrow{AB}\right\|} + \boldsymbol{u}_A$

Table 7.2: Algorithm of initial displacement calculation due to the traveller tilting at one node N.

Figure 7.3: Simpilfied example of the balanced cantilever method - FE mesh with segment description.

with gradually increased tilting angle θ. A dead weight load was applied on particular segments but its application time was delayed one step after the given segment birth in order to visualize the effect of tilting. Table 7.4 summarizes particular stages of the construction process.

For the simplification, a linear elastic material model was assumed on all segments. Elastic modulus was considered to be 30 GPa and Poisson's ratio was 0.3. Dead weight load was calculated considering value 25 kN/m^3. The resulting deflections of the structure can be seen in Figure 7.4. Results from the stage 3L clearly shows that the deflections of the right segments were decreased due to traveller tilting. The maximum value of deflection attained value 6.04 mm on the left end while the maximum deflection on the right end was 4.2 mm.

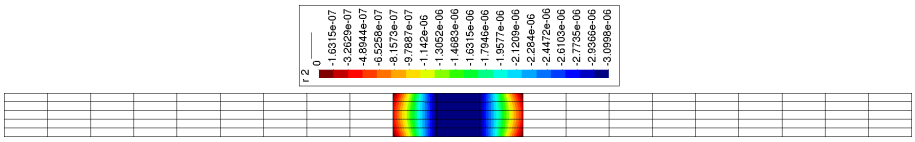
1	Identify set of new elements, Ω_N $\Omega_N = \{e_i \in \Omega \mid l_i(t+\Delta t) > 0 \ \wedge \ l_i(t) = 0\}$
2	Identify set of current elements, Ω_o $\Omega_o = \{e_i \in \Omega \mid l_i(t+\Delta t) > 0 \ \wedge \ l_i(t) > 0\}$
3	Identify set of nodes on interface between current elements and new elements, Γ_I $\Gamma_I = \{n_i \in \Omega \mid n_i \in \{e_i \in \Omega_o\} \ \wedge \ n_i \in \{e_i \in \Omega_N\}\}$
4	Save initial displacements at nodes $n_i \in \Gamma_I$
5	Generate DOF numbers for all nodes $n_i \in \{e_i \in \Omega_N\} \wedge n_i \notin \Gamma_I$
6	Determine initial displacements due to tilting, $\bar{\boldsymbol{d}}_\theta$ at all nodes of set Γ_θ according to algorithm in Table 7.2
7	Assemble vector of prescribed initial displacements on Ω_N, $\boldsymbol{d}_i^{\mathrm{T}} = \left(\boldsymbol{d}_z, \bar{\boldsymbol{d}}_\theta, \bar{\boldsymbol{d}}_i\right)$, where $\bar{\boldsymbol{d}}_i$ is the vector of saved initial displacements at Γ_I, $\boldsymbol{d}_z = \boldsymbol{0}$, $\boldsymbol{d}_z \in \mathbb{R}^{nd}$ where nd is the maximum DOF number on the domain Ω_n.
8	Initialize material models on Ω_N.
9	Compute right hand side vector $\boldsymbol{f}_i = -\int_{\Omega_N} \boldsymbol{B}^{\mathrm{T}} \boldsymbol{\sigma}_i \mathrm{d}\Omega_n$, where $\boldsymbol{\sigma}_i = \boldsymbol{D}_i \, \boldsymbol{\varepsilon}_i$, $\boldsymbol{\varepsilon}_i = \boldsymbol{B} \, \boldsymbol{d}_i$ and $\boldsymbol{D}_i$ is the initial material stiffness matrix.
10	Assemble stiffness matrix of the reduced problem, $\boldsymbol{K}_n = \int_{\Omega_N} \boldsymbol{B}^{\mathrm{T}} \boldsymbol{D}_i \boldsymbol{B} \mathrm{d}\Omega_n$
11	Solve problem $\boldsymbol{K}_n \, \boldsymbol{d}_0 = \boldsymbol{f}_i$ for unknown vector $\boldsymbol{d}_0$
12	Save initial displacements from vectors $\boldsymbol{d}_0$, $\bar{\boldsymbol{d}}_\theta$, and $\bar{\boldsymbol{d}}_i$ on elements of Ω_N.

Table 7.3: Algorithm of initial displacement calculation with the traveller tilting.

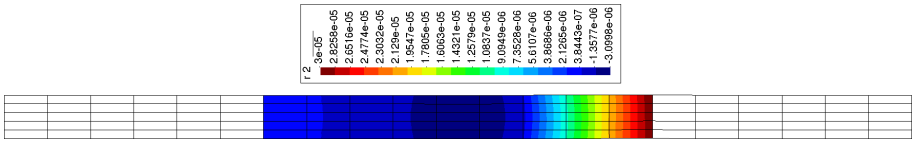
Stage	Indices of new segments and tilting angle θ	Indices of loaded segments
0	1, $\theta = \mathrm{NA}$	1
1	2, $\theta = \mathrm{NA}$ 5, $\theta = 1 \cdot 10^{-5}$	1
1L		1, 2, 5
2	3, $\theta = \mathrm{NA}$ 6, $\theta = 1 \cdot 10^{-4}$	1, 2, 5
2L		1, 2, 3, 5, 6
3	4, $\theta = \mathrm{NA}$ 7, $\theta = 2 \cdot 10^{-4}$	1, 2, 3, 5, 6
3L		1, 2, 3, 4, 5, 6, 7

Table 7.4: Description of construction stages for the simple example of balanced cantilever method.

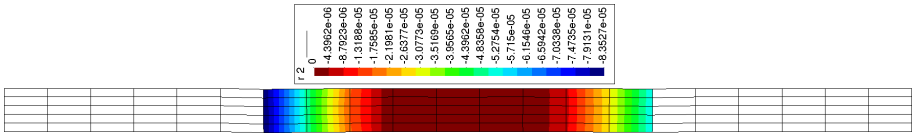
Stage 0:



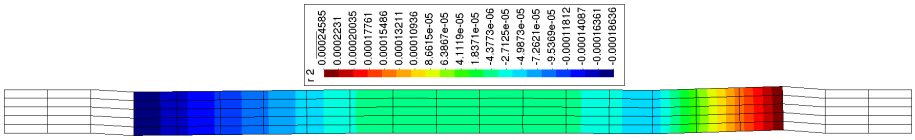
Stage 1:



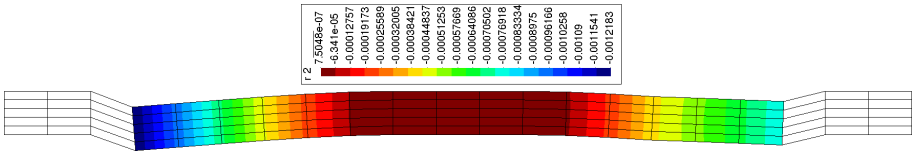
Stage 1L:



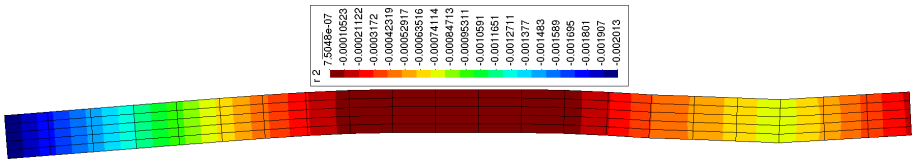
Stage 2:



Stage 2L:



Stage 3:



Stage 3L:

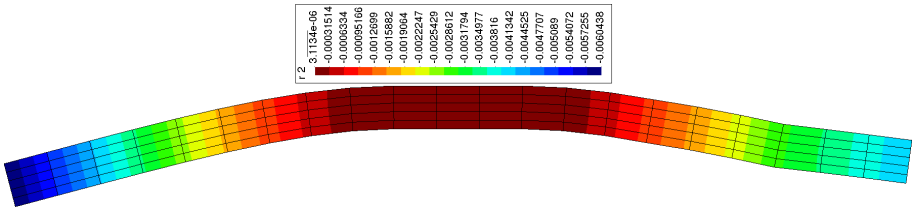


Figure 7.4: Simpilfied example of the balanced cantilever method - resulting deflections.

Figure 7.5: Location of the bridge in Mělník city.

7.4 Analysis of the girder beam bridge in Mělník

The developed tools were tested on the real bridge structure which is located at the Mělník city on road I/16, see map in Figure 7.5. The structure is central part of the bridge system which traverse Labe river and industrial railway on the exit road I/16. It is a three-span bridge where the main span traverse the river and one side span traverse the railroad lines, see Figure 7.6 and 7.7. The length of the main span is 146.2 m, side spans

Figure 7.6: Side views on the bridge from the right Labe bank, industrial railway is located on the right side.

Figure 7.7: Side views on the bridge from the left Labe bank, industrial railway is located on the right side.

have equal length 72.05 m. The bridge represents prestressed concrete structure that was constructed by the cantilever casting method. The construction started in January 1991 and finished in October 1992, the tendon prestressing finished in June 1993. Particular stages are depicted in Figure 7.8.

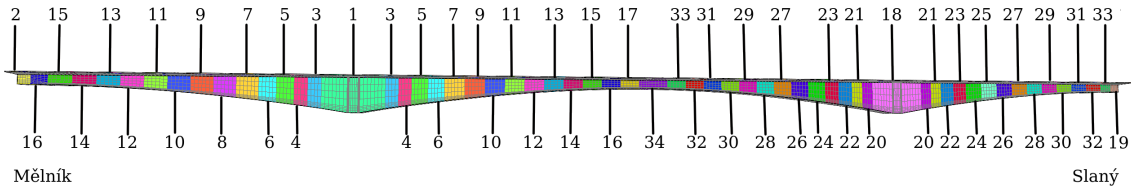


Figure 7.8: Side view on the bridge with FE mesh and concrete segment numbering.

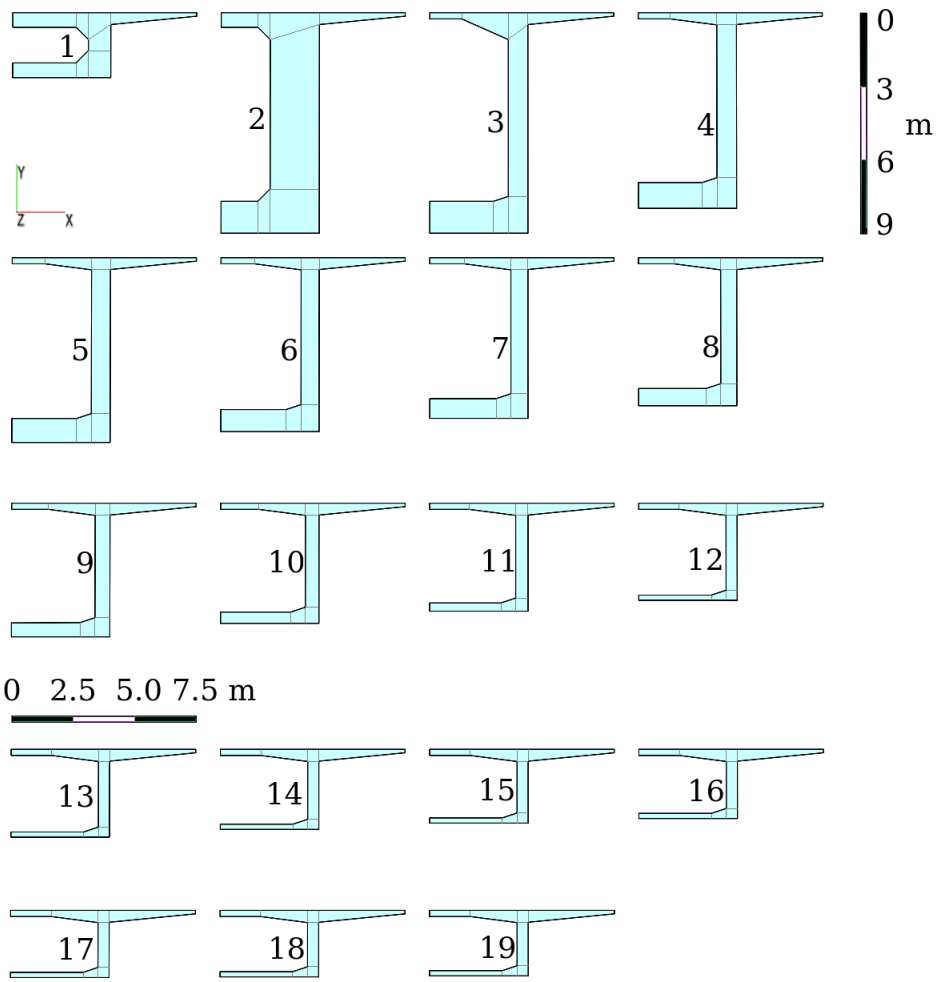


Figure 7.9: Set of cross-sections used for the 3D mesh generation - only halves were considered because of symmetry along the vertical axis.

Time schedule of the casting and prestressing procedures is given in Tables 7.5 and 7.6. The bridge has variable cross-section because of stiffeners in cross-sections at piers and parabolic haunches at spans. Therefore, the 2D mesh for the beginning and final cross-sections of particular segments were created and finally 3D structured mesh was generated by extrusion of 2D mesh between beginning and final cross-sections of the segments. A special set of automatic generators for 2D cross-section meshes and 3D bridge mesh was created, see BRIDGEN tool [Fiedler and Koudelka, 2011]. Particular cross-sections are depicted in Figure 7.9. The bridge was considered to be symmetric along the plane defined by longitudinal and vertical axes and therefore, this assumption was exploited in order to reduce the mesh size. The resulting mesh contained 73,765 nodes, 48,896 hexahedral elements and 10,448 3D bar elements. Resulting 3D mesh of finite elements is captured in Figures 7.10 The bridge contains 94 groups of tendons which were gradually activate in 40 stages. Positions of particular tendons are captured in Figure 7.11.

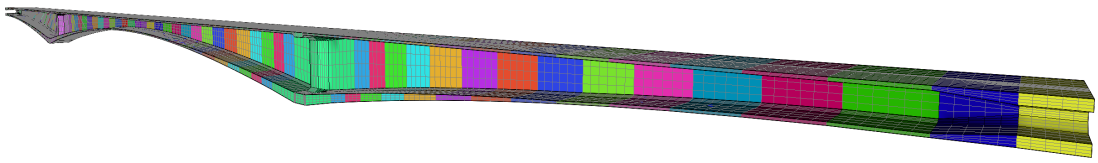
Unfortunately, data from the construction process regarding the traveller tilting were not known. Therefore, the deflections during the construction process were only partially mitigated by adjustment of the tendon pre-stressing. The calculation of the initial dis-

Segment	Series of section numbers used for the mesh generation	Segment length [m]	Time of activation from the construction beginning [days]
Construction of the starter segments of the first saddle			
1	4–3–2–2–3–4	15	7
2	1–1	2.65	66
Construction of segments from the first starter segment			
3	5–4	3.125	97
4	4–5	3.125	97
5	6–5	3.125	111
6	5–6	3.125	111
7	7–6	4	127
8	6–7	4	127
9	8–7	4	141
10	7–8	4	141
11	9–8	5	158
12	8–9	5	158
13	10–9	5	172
14	11–10	5	188
15	9–10	5	188
16	12–11	5	202
17	10–11	5	202
18	13–12	5	219
19	11–12	5	219
Construction of the starter segment of the second saddle			
20	4–3–2–2–3–4	15	219
Construction of segments of the first saddle continued			
21	14–13	5	233
22	12–13	5	233
23	15–14	5	250
24	13–14	5	250
25	16–15	5	264
26	14–15	5	264
27	17–16	5	280
27	15–16	5	280
28	1–17	3.5	294
29	16–18	5	294
30	18–19	5	310

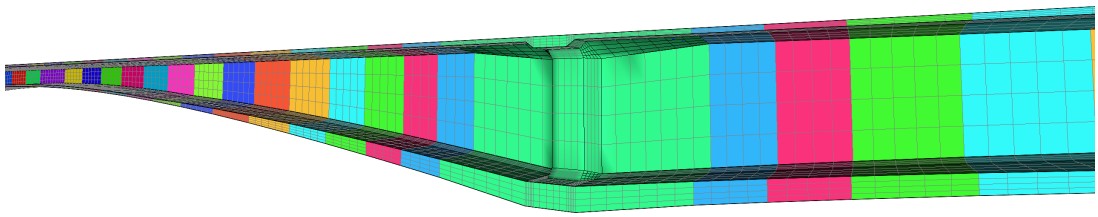
Table 7.5: Time schedule of concrete segment construction, segment lengths and cross-section definition for segments - part I.

Segment	Series of section numbers used for the mesh generation	Segment length [m]	Time of activation from the construction beginning [days]
	Construction of segments of the second saddle		
31	5–4	3.125	432
32	4–5	3.125	432
	Construction of the starter segment for the second saddle		
33	1–1	2.65	432
	Construction of segments of the second saddle		
34	6–5	3.125	446
35	5–6	3.125	446
36	7–6	4	463
37	6–7	4	463
38	8–7	4	477
39	7–8	4	477
40	9–8	5	493
41	8–9	5	493
42	9–10	5	507
43	10–9	5	524
44	10–11	5	524
45	11–10	5	538
46	11–12	5	538
47	12–11	5	554
48	12–13	5	554
49	13–12	5	568
50	13–14	5	568
51	14–13	5	585
51	14–15	5	585
52	15–14	5	599
53	15–16	5	599
54	16–15	5	616
55	16–17	5	616
56	18–16	5	630
57	17–1	3.5	630
	Connection of saddles		
58	19–19–18	5.4	646

Table 7.6: Time schedule of concrete segment construction, segment lengths and cross-section definition for segments - part II.



(a) Isometric view on the whole bridge.



(b) Detailed isometric view on the stiffener at pier's section.

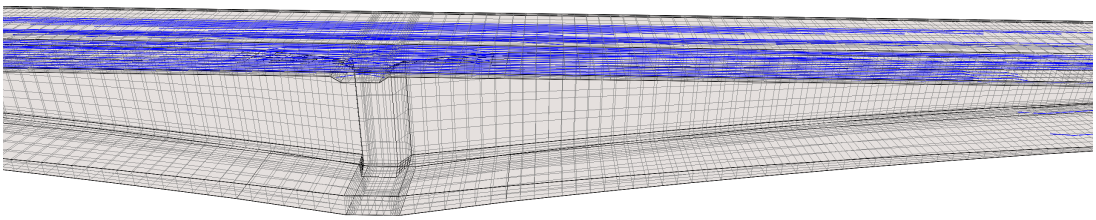
(c) Top view on the bridge deck.

Figure 7.10: 3D bridge mesh.

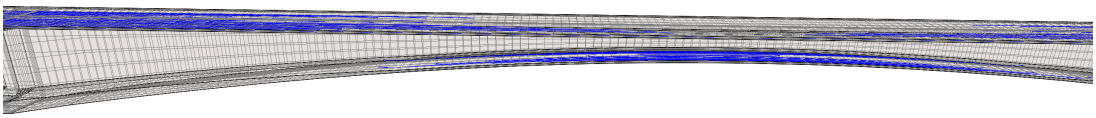
placements of the new segments was therefore performed according to the algorithm in Table 7.1. Another issue was connected with the conditions under which the deflection measurement was performed. Mostly, the measurements were performed at the same temperatures, but there were also exceptions. Exact initial values of tendon pre-stressing were also unknown, only their range was known to be 1,200–1,500 MPa. Initial values of pre-stressing were therefore involved as an additional parameter for the model calibration.

Because of unknown tilting and initial pre-stressing values, the calibration of the model parameters could not be performed on the total deflection. However, the parameters were fitted concerning deflection increment counted after 1,000 days.

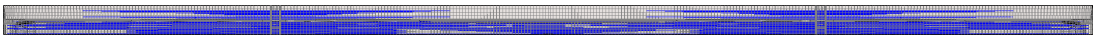
Parameters of B3 and relaxation model were fitted to capture long-term deflection in the midspan. Assessment of the influence of particular model parameters was performed in terms of global sensitivity analysis. Individual parameters were varied within the estimated bounds, and their combinations formed the so-called Design of Experiments (DoE); see, e.g. [Myšáková and Lepš, 2012] for more details. The resulting DoE was composed of 3,000 combinations where the evaluation of one combination took 4 hours of CPU time on average. The grid computing in MetaCentrum on a Linux platform was used to reduce the time demands; see [Koudelka et al., 2014] for further details. Resulting set of model parameters and used range of parameter values are summarized in Table 7.7. The given set of parameters represents the best fit of the midspan and side span deflection on the Mělník side.



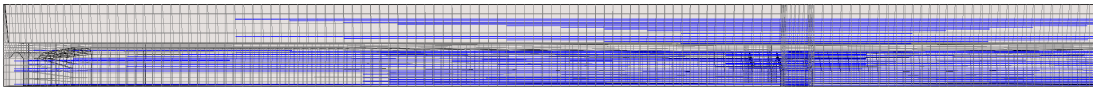
(a) Side view on the bridge - top tendons in the vicinity of the pier.



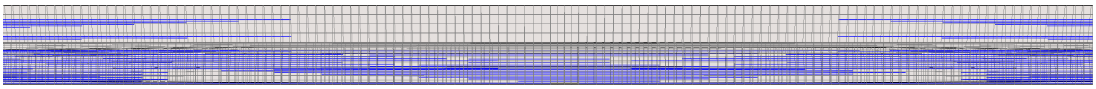
(b) Side view on the bridge - bottom tendons in the midspan.



(c) Top view on the bridge deck with tendons.



(d) Top view on the bridge deck with tendons - left support.



(e) Top view on the bridge deck with tendons - midspan.

Figure 7.11: 3D bridge mesh - views on tendons.

Parameter	q_1	q_2	q_3	q_4	q_5	σ_{p0}	a_p	b_p
Parameter	14.504	43.511	0.435	7.252	0.0	1200	1.65	9.1
value range	29.008	188.549	4.351	18.855	1450.377	1500	6.7	44
Units	10^{-6}/MPa					MPa	–	–
Fitted values	28.079	78.162	1.809	18.507	918.274	1267	40.5	8.707

Table 7.7: Fitted parameters of B3 and relaxation models.

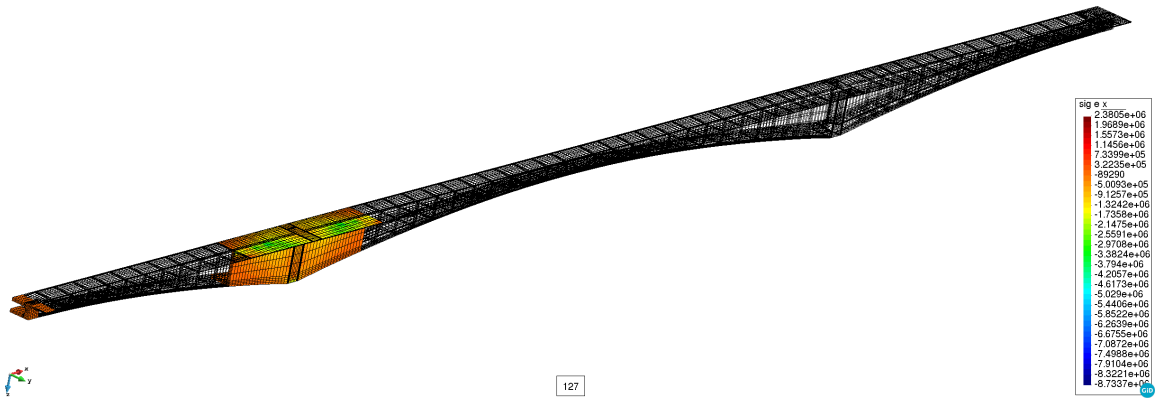
Distribution of the resulting normal stresses, σ_x, is depicted in Figures 7.12–7.15. The x ordinate is oriented along the horizontal bridge axis. The first picture series in Figure 7.12 captures stress states after 7, 66 and 127 days from the beginning of the bridge construction. It starts by the building of the starter segments at days 7 and 66 and the last state shows three segments added symmetrically to the starter one and thus creating a saddle. Stress concentrations due to tendon anchorage can be observed on the bridge deck.

The series of pictures in Figure 7.13 captures the intermediate state in the construction of the first saddle (day 219) and the connection of the first saddle with the starter segment on the Mělník bank after 294 days. Consequently, the construction process moved to the second saddle, which is captured after 432 days, where the first segments are connected to the starter one, and the starter segment on the Slaný bank was built. Again, there can be distinguished stress concentrations due to tendon anchorage on the deck of the first saddle.

The third picture series in Figure 7.14 captures states just before the connection of both saddles (day 630), after the building of the last segment (day 646) and the end of the bridge construction phase when the remaining continuous tendons were activated (day 660). The last picture series in Figure 7.15 captures two stress states. Figure 7.15a captures the stress state after 6003 days, representing the end of the period where deflection measurements were available. Figure 7.15b shows the stress distribution from the last time step (day 8400) of the performed analysis.

(a) Stress state after 7 days.

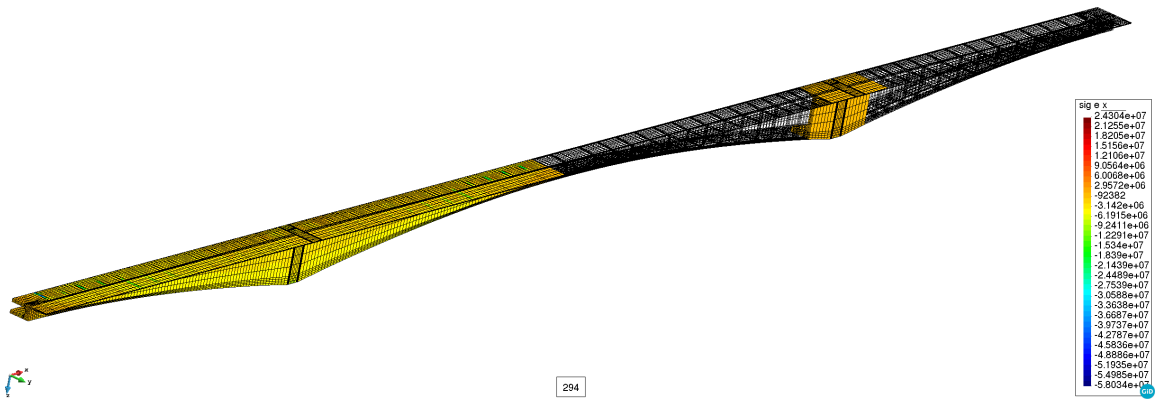
(b) Stress state after 66 days.



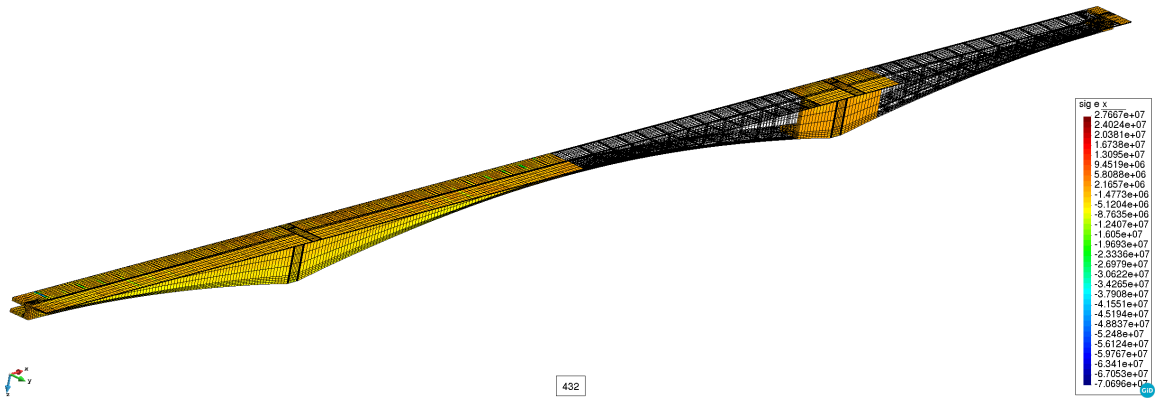
(c) Stress state after 127 days.

Figure 7.12: 3D bridge mesh - evolution of the normal stress, σ_x, after 7, 66 and 127 days.

(a) Stress state after 219 days.



(b) Stress state after 294 days.



(c) Stress state after 432 days.

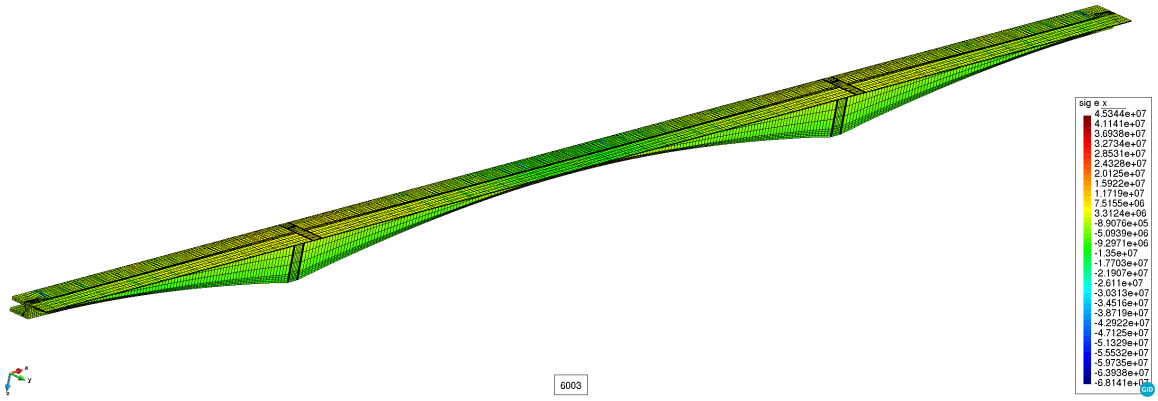
Figure 7.13: 3D bridge mesh - evolution of the normal stress, σ_x, after 219, 294 and 432 days.

(a) Stress state after 630 days.

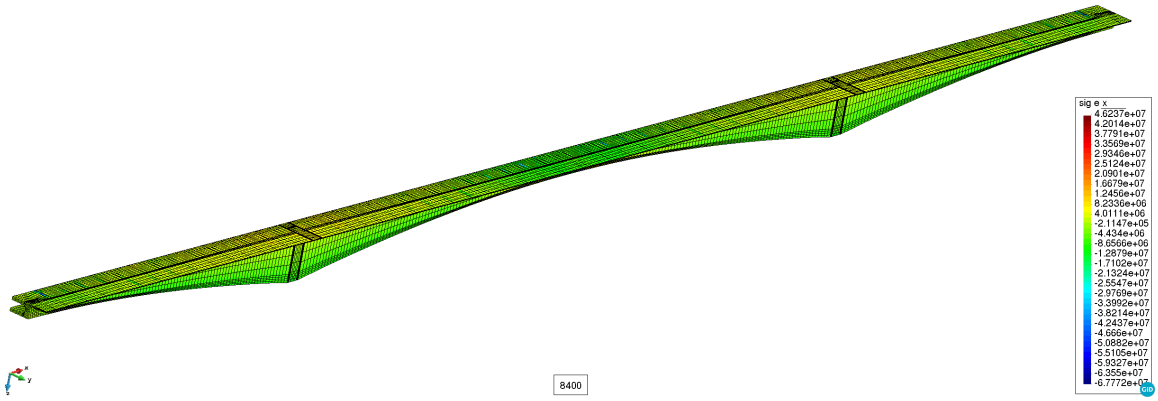
(b) Stress state after 646 days.

(c) Stress state after 660 days.

Figure 7.14: 3D bridge mesh - evolution of the normal stress, σ_x, after 630, 646 and 660 days.



(a) Stress state after 6003 days.



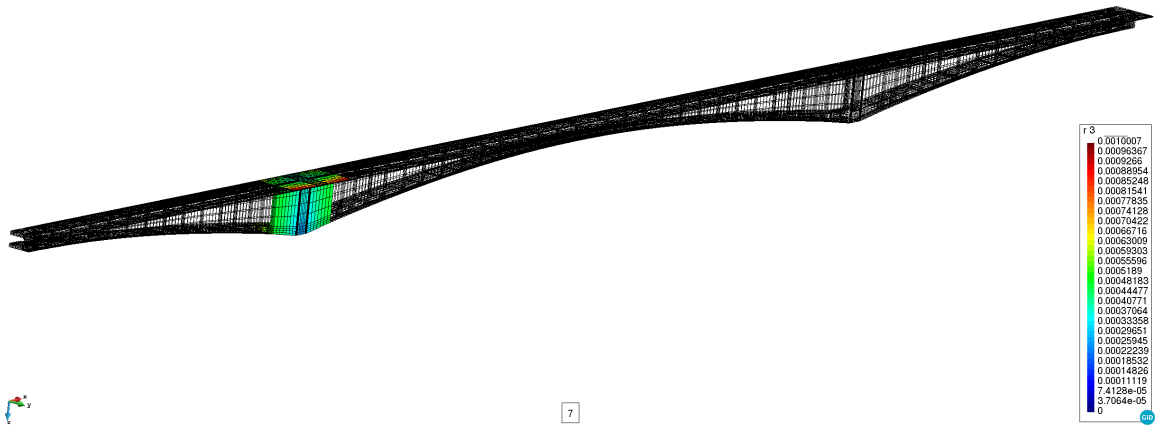
(b) Stress state after 8400 days.

Figure 7.15: 3D bridge mesh - evolution of the normal stress, σ_x, after 6003 and 8400 days.

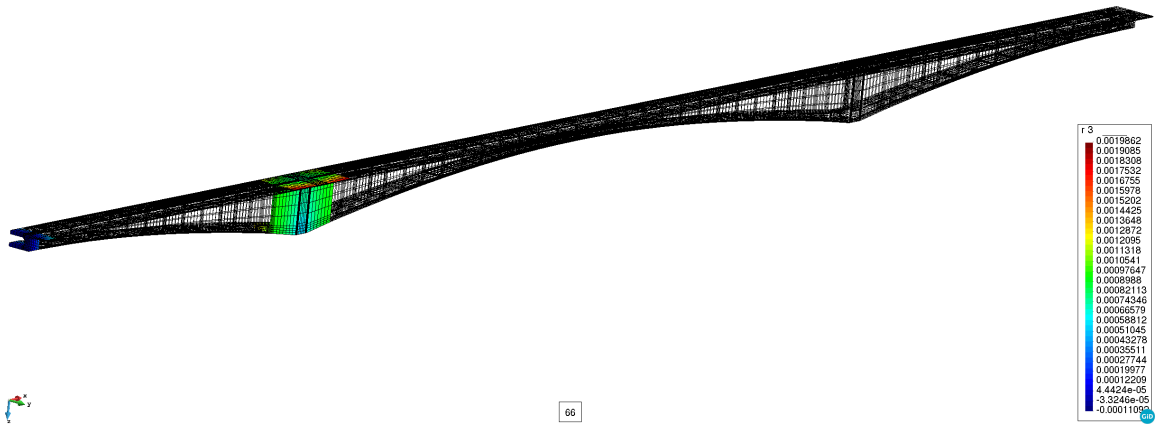
Distribution of the resulting deflections is depicted in Figures 7.16–7.19. The deflection represents vertical component, w, of the displacement vector. The positive values of the deflection are considered in the direction of positive z-axis, i.e. downward. The first picture series in Figure 7.16 captures distribution of the vertical displacement, w, after 7, 66 and 127 days from the beginning of the bridge construction. It starts by the building of the starter segments at days 7 and 66 and the last state shows three segments added symmetrically to the starter one and thus creating a saddle. Negative deflections at the saddle ends are caused by the tendon pre-stressing.

The series of pictures in Figure 7.17 captures the intermediate state in the construction of the first saddle (day 219) and the connection of the first saddle with the starter segment on the Mělník bank after 294 days. Consequently, the construction process moved to the second saddle, which is captured after 432 days, where the first segments are connected to the starter one, and the starter segment on the Slaný bank was built. At the same time the temporary support of the first saddle was removed, and thus the positive deflection appeared on the midspan end.

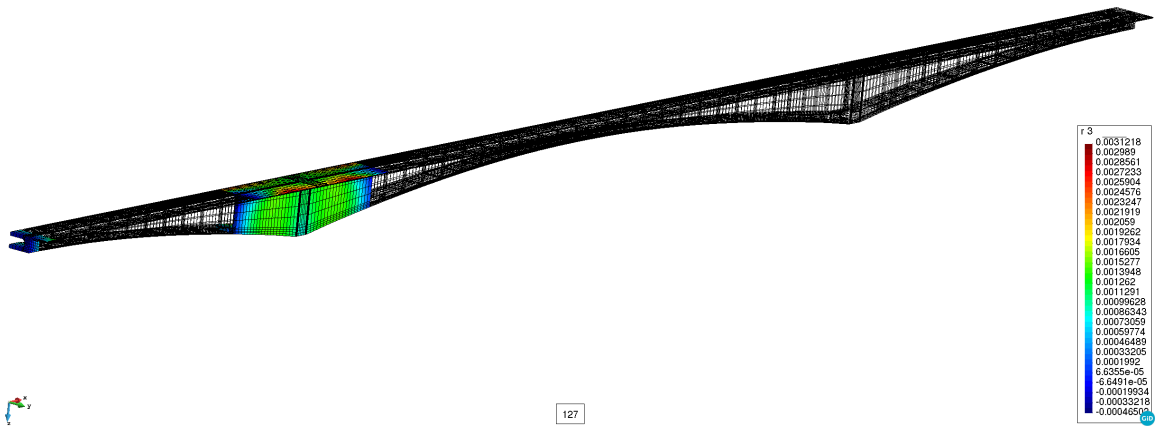
The third picture series in Figure 7.18 captures states just before the connection of both saddles (day 630), after the building of the last segment (day 646) and the end of the bridge construction phase when the remaining continuous tendons were activated (day 660). The last picture series in Figure 7.19 captures two distributions of the vertical displacement. Figure 7.19a captures the state after 6,003 days, representing the end of the period where deflection measurements were available. Figure 7.19b shows the deflection distribution from the last time step (day 8400) of the performed analysis.



(a) Distribution of the deflection, w, after 7 days.

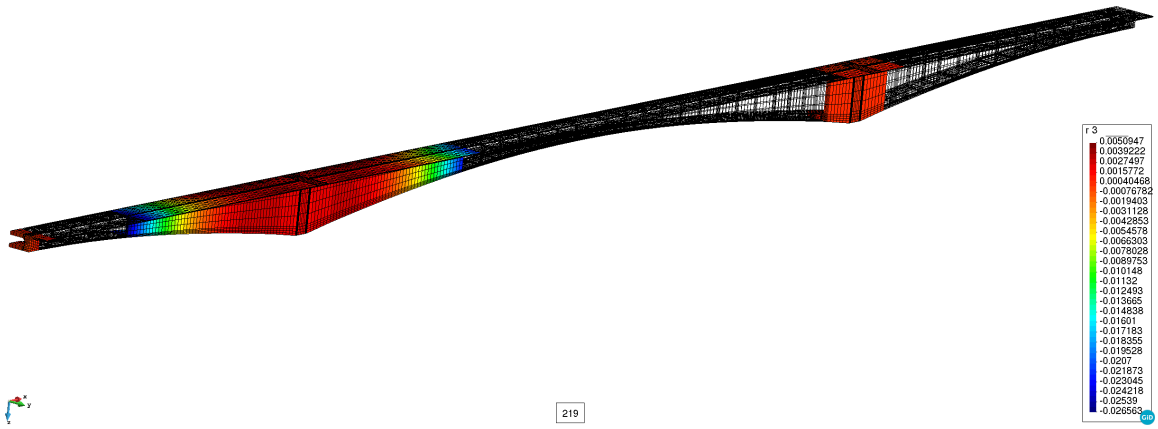


(b) Distribution of the deflection, w, after 66 days.

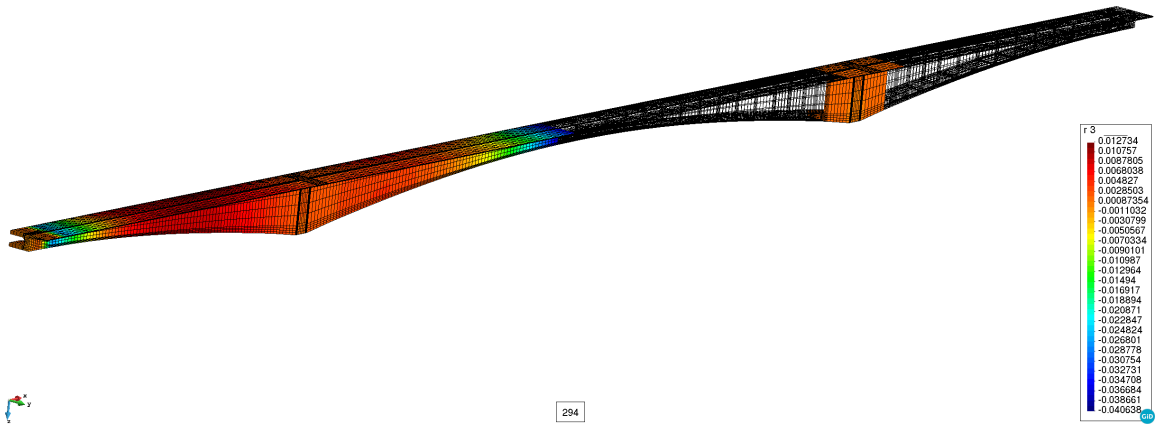


(c) Distribution of the deflection, w, after 127 days.

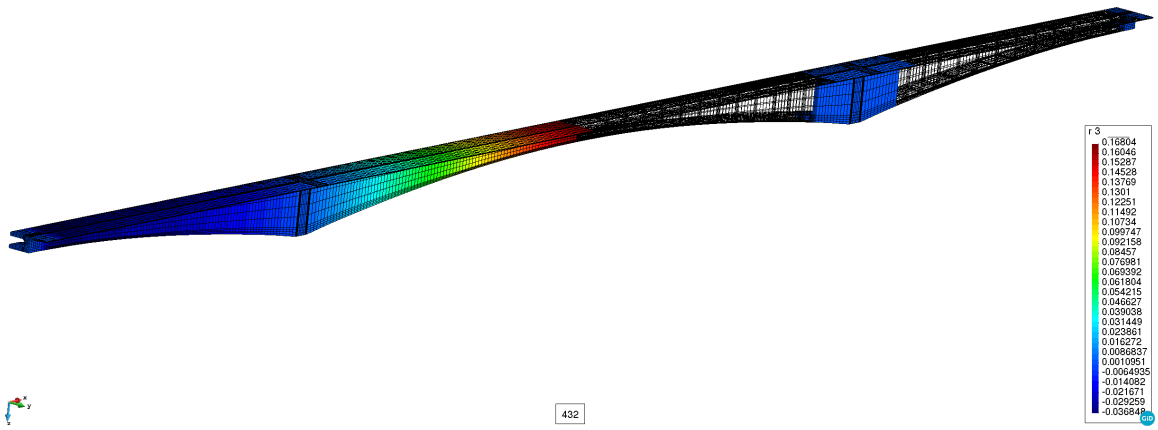
Figure 7.16: 3D bridge mesh - vertical deflection evolution after 7, 66 and 127 days.



(a) Distribution of the deflection, w, after 219 days.

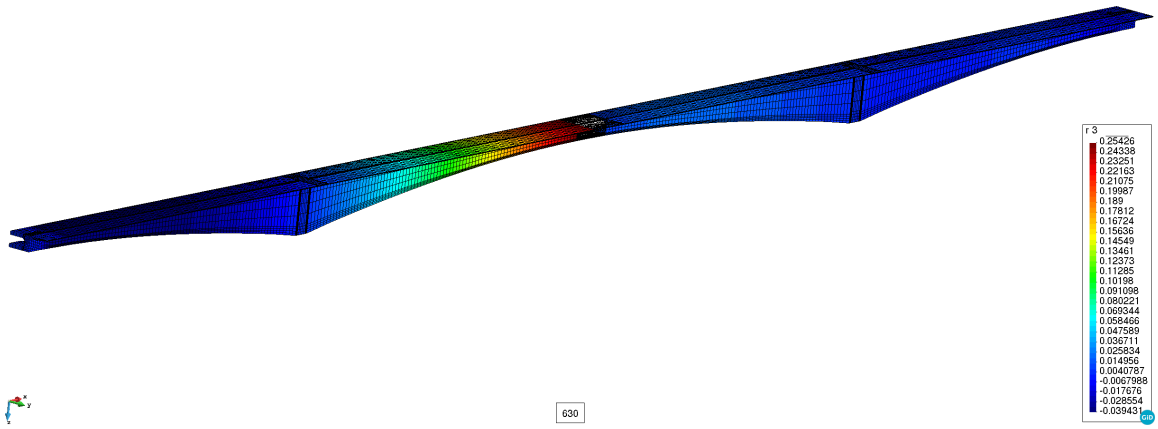


(b) Distribution of the deflection, w, after 294 days.

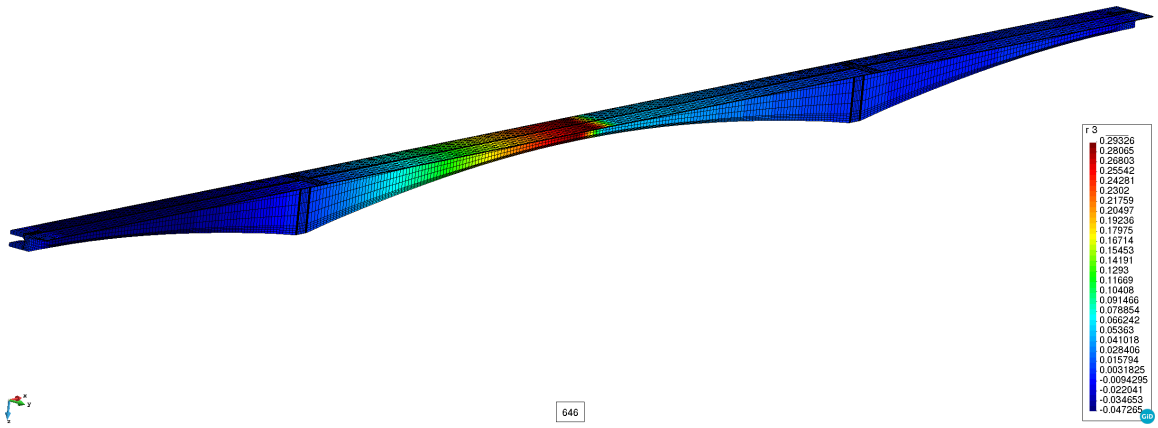


(c) Distribution of the deflection, w, after 432 days.

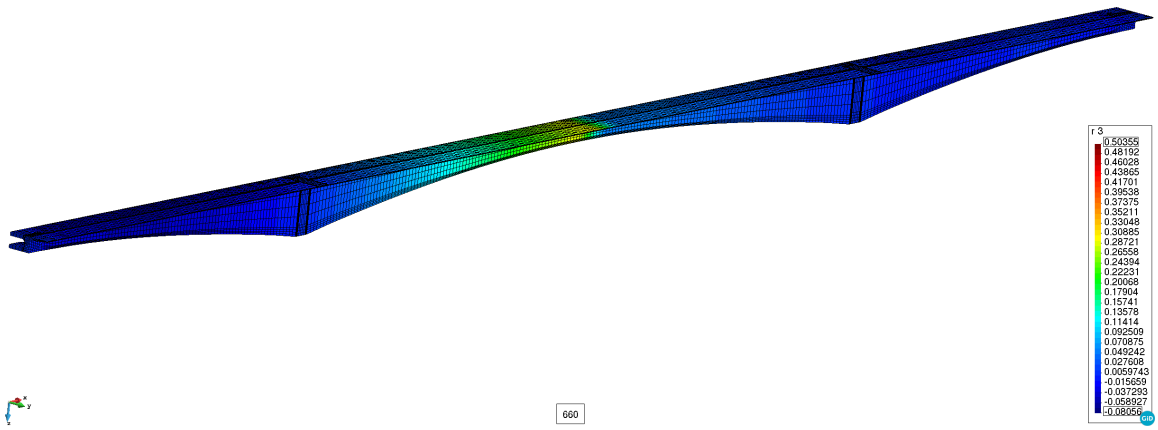
Figure 7.17: 3D bridge mesh - vertical deflection evolution after 219, 294 and 432 days.



(a) Distribution of the deflection, w, after 630 days.

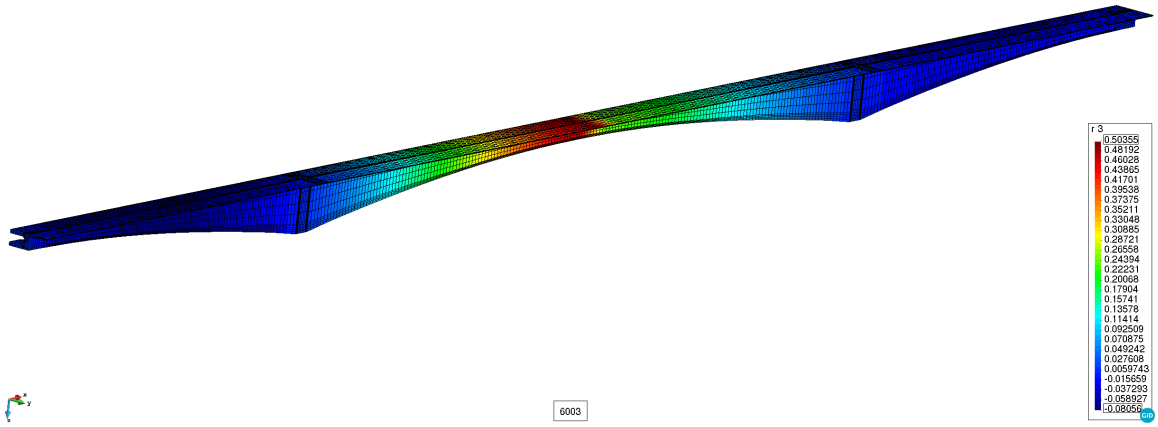


(b) Distribution of the deflection, w, after 646 days.

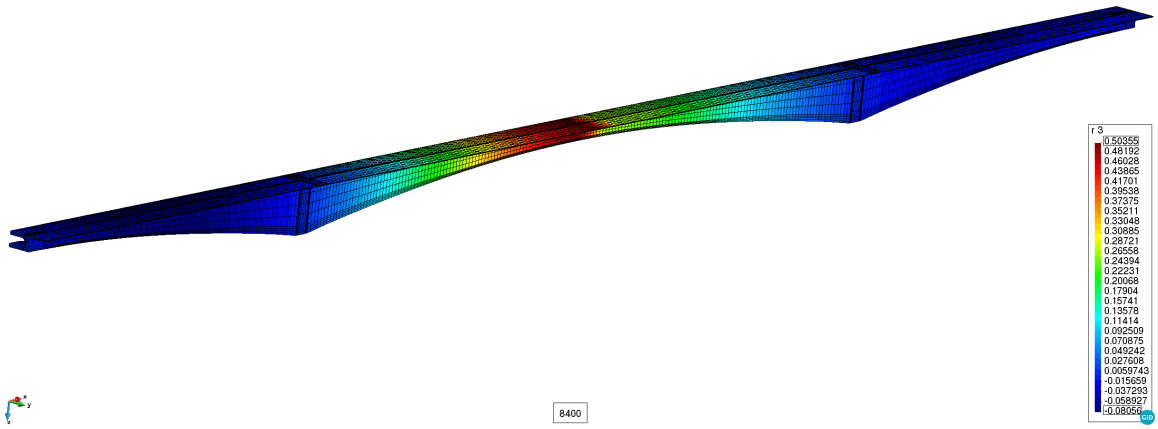


(c) Distribution of the deflection, w, after 660 days.

Figure 7.18: 3D bridge mesh - vertical deflection evolution after 630, 646 and 660 days.



(a) Distribution of the deflection, w, after 6003 days.



(b) Distribution of the deflection, w, after 8400 days.

Figure 7.19: 3D bridge mesh - vertical deflection evolution after 6003 and 8400 days.

The fitted deflection curve can be found in diagram in Figure 7.20. It can be concluded that the model parameters collected in Table 7.7 yielded a good agreement of the deflection evolution for the left side span (Mělník direction) and for the midspan, while the the deflection evolution for the right side span differs. The difference between ranged from -20 to $+20$ %. It should be noted that there is a jump in the deflection measurement at the end of measurement period. The jump was caused by the increased temperature at the time of the measurement when deflections were mostly measured at the beginning of autumn while in this case, the measurement proceeded in July. More details about the processing of deflection measurements can be found in [Vráblík et al., 2008].

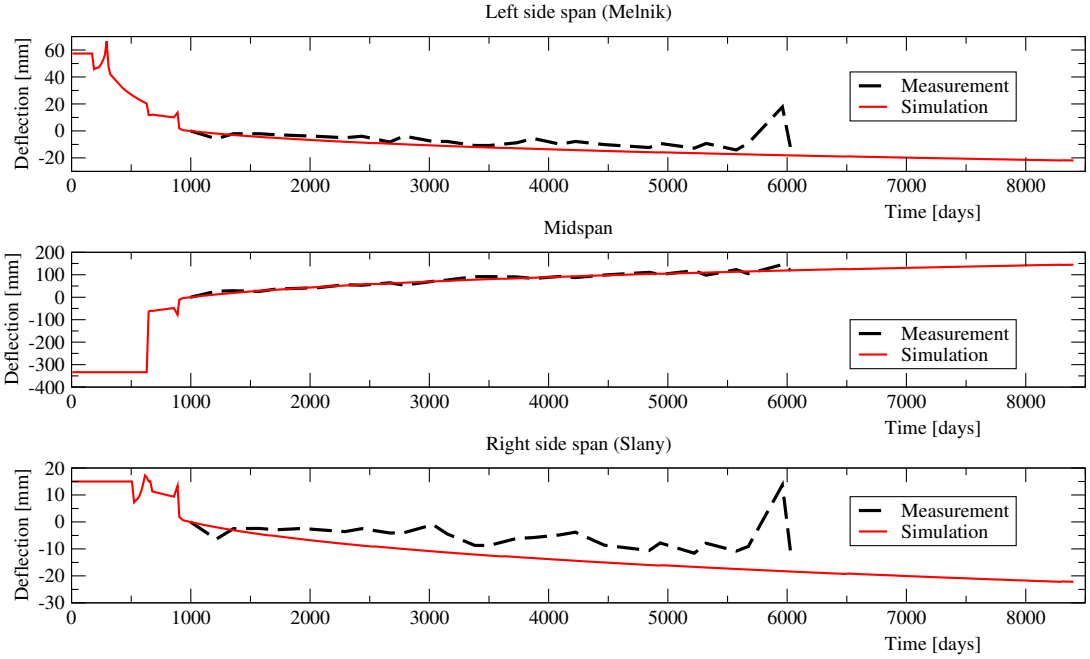


Figure 7.20: Diagram of the midspan deflection - comparison of measurement and numerical analysis.

7.5 Conclusions

A set of software tools was created for the advanced creep analysis of the girder box bridges. The development started with the generator BRIDGEN which significantly simplified the definition of 3D FE mesh. The 3D mesh had to involve tendons modelled as 3D bar elements. Software tool MIDAS was created to discretize coarse tendon geometry according to the resulting 3D mesh obtained from the BRIDGEN generator. The concept of hanging nodes was exploited both in MIDAS and SIFEL for the connection between concrete elements and tendon bars.

Additionally, the SIFEL code was extended by the proper determination of the initial displacements, which allows for the simulation of bridge construction phases in the case of casting by the cantilever method. The functionality of the proposed algorithms was demonstrated in the simple example as well as in the real bridge construction in Mělník.

Another extension of the SIFEL code was concerned with the tendon relaxation model based on the generalized model from EN 1992-1-1. The creep model was based on Bazant's B3 model implemented in SIFEL, which was adjusted for the isothermal conditions. The model parameters were determined with the help of the last link of the software tool chain CONVECTOR.

Further investigation of the real bridge construction process revealed that there were bridge deck level adjustments. They were attained either by increasing tendon pre-stressing or tilting of the casting traveller. Tendon pre-stressing level can be adjusted with the help of the relaxation model parameters, but the tilting of the traveller required modification of the proposed algorithm for the correct initial displacement determination.

The performance of the developed tools was demonstrated in the creep analysis of the Mělník bridge. Even though there were missing data about the pre-stressing level and traveller tilting, the model was able to capture the development of the deflection increments quite well.

Chapter 8

Modelling of expansive clays

Expansive clays are known for their large swelling/shrinkage capacity which is influenced by the water content namely. The most expansive clays are bentonites that are composed from montmorillonite and some additional elements (K, Na, Al, Ca, Mg). The swelling or shrinkage may lead to excessive increase or decrease in pressures on the retaining structures or differential settlement which can cause severe damage in superstructure as well as substructure. Bentonites are also known for their very low permeability which is expressed by the permeability coefficient with magnitude ranging typically from 10^{-12} to 10^{-14} m.s^{-1} according to void ratio [Villar and Lloret, 2007]. Swelling accompanied with low permeability comprises selfsealing properties of bentonites that are exploited in the sealing of dams for example.

Actually, the bentonites are also assumed to be a part of engineering barrier at deep geological repositories in high level radioactive waste disposals which is produced in the nuclear power plants namely. These repositories are complex engineering structures with very high demands on the safety and reliability and they are equipped by the multi-level barrier system. They should be placed in the stable rock host environment which represents natural barrier providing protection against disruptive natural events, water flow, human intrusion and the radionuclides migration. In the rock host environment, the system of galleries with chambers should be created for placement of radioactive waste surrounded by so called engineering barrier. The engineering barrier is composed from the special containers for spent nuclear fuel sealed by the bentonite layer which should able to stop the radionuclides migration in the case of container failure. Obviously, it is crucial for the design of engineering barrier to use the proper model of bentonite behaviour.

There are two distinguished groups of models for the clayey soils. One large (older) group of models is based on theory of elasto-plasticity [Simo and Hughes, 2000], [Jirásek and Bazant, 2002] or [Neto et al., 2008] where the model is described by the stress-strain relation in the form

$$\boldsymbol{\sigma} = \boldsymbol{D}_e : (\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_p), \tag{8.1}$$

where $\boldsymbol{\sigma}$ is the second-order stress tensor, $\boldsymbol{D}_e$ is the fourth-order elastic stiffness tensor, $\boldsymbol{\varepsilon}$ is the second-order total strain tensor and $\boldsymbol{\varepsilon}_p$ is the second-order plastic strain tensor. Note that operator (:) represents a double contraction. The plastic strains $\boldsymbol{\varepsilon}_p$ represent an irreversible part of the total strains $\boldsymbol{\varepsilon}$ which can be defined by the associated plastic flow

rule given by

$$\dot{\boldsymbol{\varepsilon}}_p = \dot{\gamma}\frac{\partial f(\boldsymbol{\sigma}, \boldsymbol{\eta})}{\partial \boldsymbol{\sigma}}, \tag{8.2}$$

where $f(\boldsymbol{\sigma}, \boldsymbol{\eta})$ represents selected yield function, $\boldsymbol{\eta}$ is the vector of hardening parameters and γ is the consistency parameter. Cam-Clay model [Roscoe and Burland, 1968] belongs to most popular models for clays based on elasto-plasticity. The Cam-Clay model involves pressure dependent modulus for loading and associated flow rule with isotropic hardening. It was intended for the fully saturated soils originally but there are extensions of the model for the partially saturated states proposed in [Alonso et al., 2011], [Gens et al., 2011], [Gallipoli et al., 2003], [Borja, 2004]. In these extensions, there is additional hardening parameter depending on the suction pressure which shifts the yield surface along the hydrostatic axis. Advantages of the mentioned elasto-plastic models is the pressure dependent loading, direct incorporation of the state boundary surface and, in the case of extended model, taking into account the influence of suction pressure. But they have also important shortcoming related to the elastic unloading which is not in agreement with the observed soil behaviour.

To avoid of this shortcoming, a relatively new group of hypoplastic models was developed by Kolymbas [Kolymbas, 1991], Gudehus [Gudehus, 1996], Bauer [Bauer, 1996], Niemunis [Niemunis, 2002] and Herle [Herle and Kolymbas, 2004]. They introduced different loading/unloading moduli directly in the rate form of stress-strain relation

$$\dot{\boldsymbol{\sigma}} = \boldsymbol{\mathcal{M}}(\boldsymbol{\sigma}, \dot{\boldsymbol{\varepsilon}}, \varkappa) : \dot{\boldsymbol{\varepsilon}}, \tag{8.3}$$

where $\boldsymbol{\mathcal{M}}$ is the fourth-order generalized stiffness tensor which depends on the actual stresses, $\boldsymbol{\sigma}$, strain rate, $\dot{\boldsymbol{\varepsilon}}$, and other state variables denoted by the vector $\varkappa$. The rate form of stress-strain relation of the hypoplastic models thus constitutes the system of ordinary differential equations. The total stress needed at the equilibrium conditions have to be obtained by the integration of (8.3) in time. Additionally, the state variables are also given in the rate form and, therefore, they have to be integrated too.

There is a promising model for the description of expansive partially saturated clays proposed in [Mašín, 2013] and [Mašín, 2017]. The model is composed from mechanical part based on hypoplasticity which is coupled with the hydraulic part. This model should be accompanied by the water flow description in the porous medium where the model proposed in [Lewis and Schrefler, 1998] originally and simplified in [Krejčí et al., 2014] can exploited. This chapter deals with the time integration of the Mašín model and implementation of the coupled hydro-mechanical model to the finite element (FE) package SIFEL [SIFEL, 2022]. Additionally, the excavation problem is addressed where an extension of the algorithms from chapter 7 due to excavation stages has to be established.

The structure of the chapter is as follows. Section 8.1 deals with the brief description of the hypoplastic model while the model of water flow is being described in Section 8.2. Coupling of the hypoplastic model with water flow is described in Section 8.3. The numerical methods based on the Runge-Kutta-Fehlberg approach and their application to time integration of the rate form of hypoplastic model are described in Section 8.4. There is also performance evaluation of these methods on several benchmark examples.

8.1 Hypoplastic model for expansive clays

In the field of soil mechanics, recent rapid development of hypoplastic models resulted in many popular models for cohesionless soils [Herle and Kolymbas, 2004], [Niemunis, 2002] as well as for cohesive soils (see e.g. [Mašín, 2012]). The advanced hydro-mechanical model based on hypoplasticity was proposed in [Mašín, 2013] and [Mašín, 2017]. The model takes into account the double structure of the aggregated clayey soils (see [Miao et al., 2007]) and it exploits separated formulation of behaviour on macro and micro level according to well established models [Alonso et al., 2011], [Sánchez et al., 2005] and [Romero et al., 2011] but there is added dependence of water retention on volumetric deformation. Coupling between macro and microstructure levels depends on size of macropores (interaggregate pores), the shear strength of soil is assumed to depend on the macrostructure and it is given by effective stress measure independent on microstructural quantities. Hydraulic equilibrium is assumed between both structure levels. The model is quite complex and comprehensive and therefore only brief summary of the model is given in Subsection 8.1.1 while the remaining definitions are given in Appendix B.

8.1.1 Brief description of hypoplastic model

This subsection is addressed to the brief description of the hydro-mechanical model based on hypoplasticity proposed in [Mašín, 2013] and [Mašín, 2017], and the following notation is used in further text. The second-order tensors or matrices are denoted by bold italic font (e.g. $\boldsymbol{\sigma}$, $\boldsymbol{N}$), while the fourth-order tensors are written in capital calligraphic bold font (e.g. $\boldsymbol{\mathcal{M}}$, $\boldsymbol{\mathcal{I}}$). There is also used symbol ":" between tensors of various orders for the double contraction. Additionally, $\boldsymbol{I}$ is the second-order identity tensor and $\|\boldsymbol{\sigma}\|$ denotes the Euclidean norm of tensor.

In the model, the particular quantities related to the macro-structural behaviour are denoted by superscript index ^{M} while superscript index ^{m} denotes quantities related with the micro-structural level. The dot $\dot{()}$ denotes the time derivative. The model assumes additive decomposition of the total strain rate $\dot{\boldsymbol{\varepsilon}}$ in the form

$$\dot{\boldsymbol{\varepsilon}} = \dot{\boldsymbol{\varepsilon}}^M + f_m \dot{\boldsymbol{\varepsilon}}^m, \qquad 0 \leq f_m \leq 1, \tag{8.4}$$

where f_m stands for the factor that quantifies the level of occlusion of macro-porosity by aggregates ranging from 0 to 1. The void ratios for particular structural levels are defined by

$$\begin{aligned} e &= (V_p^m + V_p^M)/V_s, && (8.5)\\ e^m &= V_p^m/V_s, && (8.6)\\ e_p^M &= V_p^M/V_A, && (8.7)\\ & && (8.8) \end{aligned}$$

where e is the total void ratio, e^m is the microstructural void ratio, e^M is the macrostructural void ration, V_p^m is the micropore volume, V_p^M is the macropore volume, V_s is the total solid volume and V_A is the volume of aggregates.

The consistent definitions of the total void ratio together with other porosity measures for particular structural levels are defined by

$$e = e^M + e^m + e^M e^m, \tag{8.9}$$

$$\frac{\dot{e}}{1+e} = \dot{\varepsilon}_V, \tag{8.10}$$

$$\frac{\dot{e}^M}{1+e^M} = \dot{\varepsilon}_V^M + (f_m - 1)\dot{\varepsilon}_V^m, \tag{8.11}$$

$$\frac{\dot{e}^m}{1+e^m} = \dot{\varepsilon}_V^m. \tag{8.12}$$

In Eqs. (8.9)-(8.12), the volumetric strains for macro and micro levels are denoted by ε_V^M and ε_V^m respectively while the total volumetric strain is denoted by ε_V ($\varepsilon_V = \mathrm{tr}(\boldsymbol{\varepsilon})$). The total degree of saturation S_r of the two pore system can be described in terms of degree of saturation of macro and micro structures by

$$S_r = S_r^M + \frac{e^m}{e}(S_r^m - S_r^M). \tag{8.13}$$

Two different mechanical models for macro and microstructure level are defined in the model. Assuming local hydraulic equilibrium $s^m = s^M$ and $\boldsymbol{\sigma}^{netM} = \boldsymbol{\sigma}^{netm}$ and the Bishop's effective stresses concept [Bishop, 1959], the following terms for the effective stresses at macro and micro levels are given

$$\boldsymbol{\sigma}^M = \boldsymbol{\sigma}^{net} - \boldsymbol{I} s \chi^M, \tag{8.14}$$

$$\boldsymbol{\sigma}^m = \boldsymbol{\sigma}^{net} - \boldsymbol{I} s \chi^m, \tag{8.15}$$

where $\boldsymbol{I}$ is the second-order identity tensor, χ is being the effective stress parameter and the net stress $\boldsymbol{\sigma}^{net}$ and suction s are defined as follows

$$\boldsymbol{\sigma}^{net} = \boldsymbol{\sigma} + u_a, \tag{8.16}$$

$$s = u_a - u_w. \tag{8.17}$$

In Eqs. (8.16) and (8.17), $\boldsymbol{\sigma}$ is the total stress while u_a and u_w are the air and water pressures respectively. It should be noted that u_a and u_w are assumed to be positive in compression. The water retention models for macro level reads

$$\chi^M = S_r^M = \chi = \begin{cases} 1, & s < s_e \\ \left(\dfrac{s_e}{s}\right)^\gamma, & s \geq s_e, \end{cases} \tag{8.18}$$

where s_e is the water retention model variable defined by the following expression

$$s_e = s_{en}(a_e + a_{sc} - a_e\, a_{sc}), \tag{8.19}$$

and parameter γ represents the slope of macrostructural water retention curve which is usually assumed by the value 0.55. In the above equation, a_e stands for the model parameter and the state variable a_{sc} is given as

$$a_{sc}(s) = \begin{cases} 0 & \text{if } s \leq s_W \\ \dfrac{s - s_W}{s_D - s_W} & \text{if } s_W < s < s_D, \\ 1 & \text{if } s \geq s_D \end{cases} \tag{8.20}$$

where s_D is the suction at the main drying curve corresponding to the actual S_r^M which is defined as

$$s_D = \frac{s_{en}}{s_e} s, \tag{8.21}$$

and $s_W = a_e s_D$ is the suction at the main wetting curve. The quantity s_{en} is defined by

$$s_{en} = s_{e0} \frac{e_0^M}{e^M}, \tag{8.22}$$

where s_{e0} and e_0^M are model parameters. The rate equation for a_{sc} reads

$$\dot{a}_{sc} = \begin{cases} 0, & s \leq s_W \text{ or } s \geq s_D, \\ \dfrac{1}{s_D - s_W} \dot{s} & \text{otherwise.} \end{cases} \tag{8.23}$$

On the micro level, the model is being defined by

$$\chi^m = S_r^m = 1, \tag{8.24}$$

and thus the fully saturated state in the micropore system is being assumed.

The rate form of the macrostructural microstructural effective stresses defined by Eqs. (8.14) and (8.15) reads

$$\dot{\boldsymbol{\sigma}}^M = \dot{\boldsymbol{\sigma}}^{net} + \boldsymbol{I} \chi^M \left[-\dot{s} + \gamma \, s \, \frac{\dot{e}^M}{e^M} \right], \tag{8.25}$$

$$\dot{\boldsymbol{\sigma}}^m = \dot{\boldsymbol{\sigma}}^{net} + \boldsymbol{I} \dot{s}. \tag{8.26}$$

The macrostructure effective stress rate is governed by the hypoplastic model given by

$$\dot{\boldsymbol{\sigma}}^M = f_s \left(\boldsymbol{\mathcal{L}} : \dot{\boldsymbol{\varepsilon}}^M + f_d \boldsymbol{N} \| \dot{\boldsymbol{\varepsilon}}^M \| \right) + f_u \boldsymbol{H}, \tag{8.27}$$

where f_s is the barotropy factor, $\boldsymbol{\mathcal{L}}$ is the hypoelastic fourth-order tensor, f_d is the pyknotropy factor, $\boldsymbol{N}$ is the second-order tensor introducing the failure condition, f_u and $\boldsymbol{H}$ are factor and second-order tensor introducing the wetting-induced collapse. Definition these quantities can be found in Appendix B.

The microstructure effective stress rate is given by the following relation

$$\dot{\boldsymbol{\sigma}^m} = \boldsymbol{I} \frac{p^m}{\kappa_m} \dot{\varepsilon}_V^m, \tag{8.28}$$

where p^m denotes the mean stress at micro level and κ_m is the model parameter. There is an explicit formulation of void ratio on the micro structural level given by term

$$e^m = \exp \left[\kappa_m \ln \frac{s_r}{p^m} + \ln(1 + e_r^m) \right] - 1, \tag{8.29}$$

where e_r^m and s_r are the material parameters representing an arbitrary reference value of void ration at micro level for the reference value of suction.

For the convenience, state variables of the model may be collected in vector $\varkappa^{\mathrm{T}} = \{e, S_r, e^M, e^m, S_r^M, a_{sc}, r_e\}$ where all components of state variable vector are defined in the rate form similarly to the stress-strain relation.

Originally, the model was implemented D. Mašín in the rate form and used for the simulations on the material point level. His implementation was transformed to the finite element code SIFEL and extended by the integration schemes proposed in Subsection 8.1.2 for better performance.

Solution to the involved hypoplastic model defined by Equation (8.27) is not straightforward for the loading input obtained from a finite element code where $\dot{\boldsymbol{\varepsilon}}$ and $\dot{s}$ are provided usually. Difficulties are associated with the fact that the total stress rate appears both in the formulation of $\dot{\boldsymbol{\varepsilon}}^m$ and in the formulation of $\dot{\boldsymbol{\sigma}}^M$, thus on both the right- and left-hand side of Equation (8.27) (see (8.12), (8.29), (8.15) and (8.16)). A numerical iterative procedure was proposed to solve this equation in [Mašín, 2017] but it leads to the problems with the material stiffness matrix formulation. In this model, only the 'elastic' stiffness $\mathcal{L}$ is assumed for the mechanical part of the generalized stiffness matrix while the coupling blocks with water transport are calculated numerically by the perturbation of model input values.

8.1.2 Modification of hypoplastic model formulation

The original definition of the a_{sc} state variable as a non-smooth function can lead to numerical difficulties with the time integration of the model and therefore approximation by a smooth function can be useful. Many approximation formulae are based on logarithmic-exponential approximation functions. An approximation formula of canonical piecewise linear function into smooth representation can be found in [Jimenez-Fernandez et al., 2016]. Generally, every piecewise linear function can be rewritten in the following canonical form

$$y(x) = a + b\,x + \sum_{i=1}^{\delta-1} c_i |x - \beta_i|, \tag{8.30}$$

where $b = 0.5\,(j_1 + j_\delta)$, $c_i = 0.5\,(j_{i+1} - j_i)$ and $a = y(0) - \sum_{i=1}^{\delta-1} c_i |\beta_i|$. In Eq. (8.30), index i denotes the i-th segment of the original function while its slope is denoted by j_i and δ stands for the number of segments. Particular segments are connected in break-points with x coordinates equal to β_i. The logarithmic-exponential approximation, $\hat{y}(x)$, of the function $y(x)$ can be written in the form

$$\hat{y}(x) = a + b\,x + \sum_{i=1}^{\delta-1} c_i\,(x - \beta_i) + \frac{2}{c_\alpha} \sum_{i=1}^{\delta-1} \ln\left(1 + \mathrm{e}^{-c_\alpha (x-\beta_i)}\right), \tag{8.31}$$

where the parameter c_α controls the smoothness of the function approximation in the break-point vicinity. Application of the formula to the definition of a_{sc} yields

$$\begin{aligned}
j_1 &= 0, \qquad j_2 = (s_D - s_W)^{-1}, \qquad j_3 = 0, && (8.32)\\
b &= 0.5(j_1 + j_3), && (8.33)\\
c_1 &= 0.5(j_2 - j_1), \qquad c_2 = 0.5(j_3 - j_2), && (8.34)\\
a &= 0 - c_1|s_W| - c_2|s_D|, && (8.35)\\
\hat{a}_{sc} &= a + bs - c_1(s - s_W) - c_2(s - s_D) + \\
&\quad \frac{2}{c_\alpha}\left[\ln\left(1 + \mathrm{e}^{c_\alpha (s - s_W)}\right) + \ln\left(1 + \mathrm{e}^{c_\alpha (s - s_D)}\right)\right]. && (8.36)
\end{aligned}$$

The meaning of particular coefficients and parameters used in the approximation can be seen in Figure 8.1. The approximation of the rate of a_{sc} can be obtained by the time

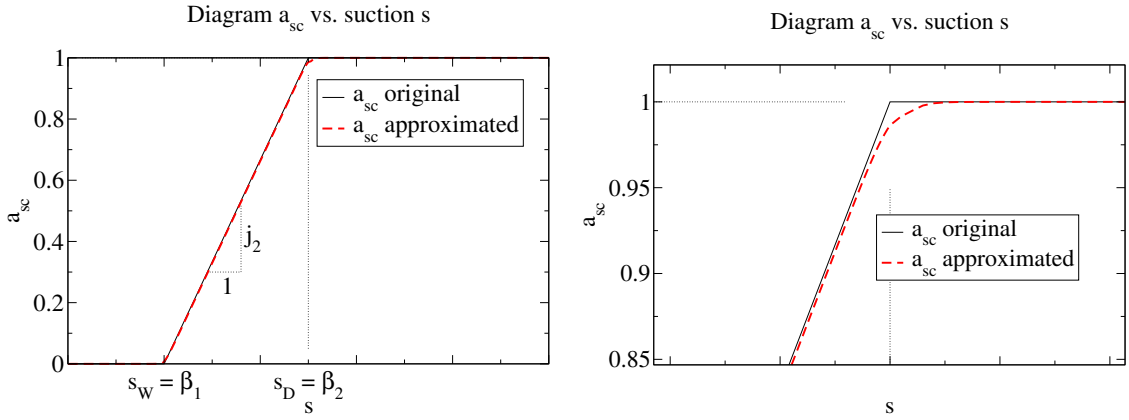


Figure 8.1: Diagram of a_{sc} state variable evolution with respect to the suction s - full view (left) and detailed view in the vicinity of the s_D point (right).

differentiation of (8.36) in the following form

$$\dot{\hat{a}}_{sc} = b - c_1 - c_2 + 2\left[c_1 \frac{\mathrm{e}^{c_\alpha(s-s_W)}}{1+\mathrm{e}^{c_\alpha(s-s_W)}} + c_2 \frac{\mathrm{e}^{c_\alpha(s-s_D)}}{1+\mathrm{e}^{c_\alpha(s-s_D)}}\right]\dot{s}. \tag{8.37}$$

8.2 Model of water flow in deforming porous medium

For the preliminary investigations, the water flow is assumed to be the only driving mechanism of moisture transfer (see [Krejčí et al., 2001]) and the flow process is considered to be at the isothermal conditions. In this case, the assumption of isothermal one-phase flow is adopted for the model based on Lewis and Schrefler‘s approach [Lewis and Schrefler, 1998]. Vector of mass flux density of moisture $\boldsymbol{J}_w$ (kg.m^{-2}.s^{-1}) is given by generalized Darcy's law [Lewis and Schrefler, 1998] in the form

$$\boldsymbol{J}_w = \rho^w n S_r \left(\boldsymbol{v}^w - \dot{\boldsymbol{u}}\right) = \rho^w \frac{k^{rw}\boldsymbol{k}_{\mathrm{sat}}}{\mu^w}\left(-\mathrm{grad}\, p_w + \rho^w \boldsymbol{g}\right), \tag{8.38}$$

where the term $\boldsymbol{v}^w - \dot{\boldsymbol{u}}$ is the mass averaged relative velocity with the water velocity $\boldsymbol{v}^w$. Additionally, $k^{rw} \in [0;1]$ stands for the relative permeability which is a function of degree of saturation $k^{rw} = k^{rw}(S_r)$ and $\boldsymbol{g}$ is the gravity acceleration vector. $\boldsymbol{k}_{\mathrm{sat}}$ [m^2] is the intrinsic permeability matrix, μ_w is the dynamic viscosity [kg m^{-1}s^{-1}] and $p_w = -u_w$ is the pore pressure. The intrinsic mass density ρ^w is related to the volume averaged mass density ρ_w according to the following relation

$$\rho_w = n\, S_r\, \rho^w, \tag{8.39}$$

where n is the porosity defined as $n = e/(1+e)$.

The constitutive equation has to be accompanied with the mass balance equations for particular phase which can be defined as follows

$$\frac{\partial(n\,S_r\,\rho^w)}{\partial t} + \mathrm{div}\,(n\,S_r\,\rho^w\,\boldsymbol{v}^w) = \pm\dot{m} \quad \text{for liquid phase,} \tag{8.40}$$

$$\frac{\partial\left((1-n)\,\rho^s\right)}{\partial t} + \mathrm{div}\,((1-n)\,\rho^s\,\dot{\boldsymbol{u}}) = 0 \quad \text{for solid phase,} \tag{8.41}$$

where $\dot{m}$ is the mass rate of evaporation. Equation (8.41) can be rewritten to express the term $\dfrac{\partial n}{\partial t}$ and by neglecting the gradient $\nabla\left[(1-n)\rho^s\right]$ to the form

$$\frac{\partial n}{\partial t} = \frac{1-n}{\rho^s}\frac{\partial \rho^s}{\partial t} + (1-n)\mathrm{div}\,\dot{\boldsymbol{u}}. \tag{8.42}$$

Substitution of Equation (8.42) to (8.40) and neglecting the term $\nabla(nS_r\rho^w)$ results in continuity equation for fluid phase

$$\frac{1-n}{\rho^s}\frac{\partial \rho^s}{\partial t} + \frac{n}{\rho^w}\frac{\partial \rho^w}{\partial t} + (1-n)\mathrm{div}\,\dot{\boldsymbol{u}} + \frac{n}{S_r}\frac{\partial S_r}{\partial t} + n\mathrm{div}\boldsymbol{v}^w = \pm\frac{\dot{m}}{S_r\,\rho^w} \tag{8.43}$$

The continuity equation can be rewritten with assumptions of small strains, small displacements, isothermal conditions and mass conservation law by introduction of the following terms for densities ρ^s and ρ^w

$$\frac{1}{\rho^w}\frac{\partial \rho^w}{\partial t} = \frac{1}{K_w}\frac{\partial p_w}{\partial t}, \tag{8.44}$$

$$\frac{1}{\rho^s}\frac{\partial \rho^s}{\partial t} = \frac{1}{1-n}\left((\alpha-n)\frac{1}{K_g}\frac{\partial S_r\,p_w}{\partial t} - (1-\alpha)\mathrm{div}\,\dot{\boldsymbol{u}}\right), \tag{8.45}$$

where K_w is the bulk modulus of water [Pa], K_g is the bulk modulus of the grains [Pa] and $\alpha = 1 - \dfrac{K_s}{K_g}$ is the Biot's constant, where K_s is the bulk modulus of the solid skeleton [Pa]. The continuity condition then reads

$$\left(\frac{\alpha-n}{K_g}S_r^2 + \frac{n\,S_r}{K_w} + \frac{\alpha-n}{K_g}S_r\,p_w\frac{\partial S_r}{\partial p_w} + n\frac{\partial S_r}{\partial p_w}\right)\frac{\partial p_w}{\partial t} + \\ +\alpha\,S_r\,\mathrm{div}\,\dot{\boldsymbol{u}} + \frac{1}{\rho^w}\mathrm{div}\,(nS_r\rho^w\,(\boldsymbol{v}^w - \dot{\boldsymbol{u}})) = \pm\frac{\dot{m}}{\rho^w}, \tag{8.46}$$

where the dependency of the degree of saturation on the pore water pressure $\dfrac{\partial S_r}{\partial t} = \dfrac{\partial S_r}{\partial p_w}\dfrac{\partial p_w}{\partial t}$ was used. If there is no exchange of water vapour the term of the right hand side of Equation (8.46) becomes zero. Substitution of the Darcy's law (8.38) into Equation (8.46) results in continuity equation for one-phase (liquid water) flow in deforming medium

$$\left(\frac{\alpha-n}{K_g}S_r^2 + \frac{n\,S_r}{K_w} + \frac{\alpha-n}{K_g}S_r\,p_w\frac{\partial S_r}{\partial p_w} + n\frac{\partial S_r}{\partial p_w}\right)\frac{\partial p_w}{\partial t} + \\ +\alpha\,S_r\,\mathrm{div}\,\dot{\boldsymbol{u}} + \frac{1}{\rho^w}\mathrm{div}\left(\rho^w\frac{k^{rw}\boldsymbol{k}_{sat}}{\mu^w}\left(-\mathrm{grad}\,p_w + \rho^w\,\boldsymbol{g}\right)\right) = 0. \tag{8.47}$$

8.3 Numerical solution to coupled hydromechanical model

The coupled hydromechanical model is defined on domain $\Omega \subset \mathbb{R}^d$, where d is the space dimension. The boundary of the domain Ω is split with respect to transport processes to parts Γ_p and Γ_N, where pore pressure, $\bar{p}_w$ is prescribed on the part Γ_p while water flux vector, $\bar{\boldsymbol{J}}_w$ is prescribed on the part Γ_N. The following relationships $\Gamma_p \cup \Gamma_N = \Gamma$ and $\Gamma_p \cap \Gamma_N = \emptyset$ have to be satisfied. Boundary conditions can be written in the form

$$p_w = \bar{p}_w \quad \forall \boldsymbol{x} \in \Gamma_p, \tag{8.48}$$

$$\boldsymbol{J}_w = \bar{\boldsymbol{J}}_w \quad \forall \boldsymbol{x} \in \Gamma_N. \tag{8.49}$$

The boundary of the domain Ω is split with respect to mechanical problem to parts Γ_u and Γ_t, where displacement, $\boldsymbol{u}$, is prescribed on the part Γ_u while surface traction, $\bar{\boldsymbol{t}}$, is prescribed on the part Γ_t. Similarly to the previous decomposition, the relationships $\Gamma_u \cup \Gamma_t = \Gamma$ and $\Gamma_u \cap \Gamma_t = \emptyset$ have to be satisfied. Boundary conditions can be summarized as

$$\dot{\boldsymbol{u}} = \dot{\bar{\boldsymbol{u}}} \quad \forall \boldsymbol{x} \in \Gamma_u, \tag{8.50}$$

$$\boldsymbol{S}\dot{\boldsymbol{\sigma}} = \dot{\bar{\boldsymbol{t}}} \quad \forall \boldsymbol{x} \in \Gamma_t, \tag{8.51}$$

where $\boldsymbol{S}$ is the matrix defined in (6.9).

From now on, stresses and strains are stored in vectors and stiffness tensor is transformed to matrix. The total stress $\boldsymbol{\sigma}$ which is defined by equations (8.4), (8.14), (8.15), (8.16), (8.27) and (8.28) can be expressed in the form of vector function

$$\boldsymbol{\sigma} = \boldsymbol{g}(\boldsymbol{\varepsilon}(\boldsymbol{u}), p_w), \tag{8.52}$$

where $\boldsymbol{\varepsilon} = \boldsymbol{\partial u}$ is the strain vector. Time derivative of the stress vector has the form

$$\dot{\boldsymbol{\sigma}} = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{\varepsilon}}\dot{\boldsymbol{\varepsilon}} + \frac{\partial \boldsymbol{g}}{\partial p_w}\dot{p}_w = \boldsymbol{D}\dot{\boldsymbol{\varepsilon}} + \boldsymbol{h}\dot{p}_w, \tag{8.53}$$

where the new matrix $\boldsymbol{D}$ and new vector $\boldsymbol{h}$ were defined. The rate of change of the total stress has to satisfy the equilibrium equation in the form

$$\boldsymbol{\partial}^{\mathrm{T}}\left(\boldsymbol{D}\dot{\boldsymbol{\varepsilon}} + \boldsymbol{h}\dot{p}_w\right) + \dot{\boldsymbol{b}} = \boldsymbol{0} \qquad \forall \boldsymbol{x} \in \Omega, \tag{8.54}$$

where $\dot{\boldsymbol{b}}$ is the time derivative of the body force vector. Finally, the initial conditions are in the form

$$p_w(\boldsymbol{x}, t=0) = p_{w,0}(\boldsymbol{x}) \qquad \forall \boldsymbol{x} \in \Omega, \tag{8.55}$$

$$\boldsymbol{u}(\boldsymbol{x}, t=0) = \boldsymbol{u}_0(\boldsymbol{x}) \qquad \forall \boldsymbol{x} \in \Omega, \tag{8.56}$$

where $p_{w,0}(\boldsymbol{x})$ and $\boldsymbol{u}_0(\boldsymbol{x})$ are given functions.

The unknown functions $\boldsymbol{u}$ and p_w are approximated in the classical form

$$\boldsymbol{u} = \boldsymbol{N}_u\boldsymbol{d}, \tag{8.57}$$

$$p_w = \boldsymbol{N}_p\boldsymbol{p}, \tag{8.58}$$

where $\boldsymbol{d}$ and $\boldsymbol{p}$ are vectors of the appropriate nodal values and $\boldsymbol{N}_u$ and $\boldsymbol{N}_p$ are matrices of shape functions. Two additional matrices will be needed

$$\boldsymbol{B}_u = \boldsymbol{\partial} \boldsymbol{N}_u, \tag{8.59}$$

$$\boldsymbol{B}_p = \nabla \boldsymbol{N}_p. \tag{8.60}$$

Applying Galerkin's method, the Gauss theorem and using the spatial discretization, the system of discretized equations for the hydro-mechanical problem (cf. [Lewis and Schrefler, 1998], [Bittnar and Šejnoha, 1996]) is obtained in the form

$$\begin{pmatrix} \boldsymbol{K}_{uu} & \boldsymbol{C}_{up} \\ \boldsymbol{C}_{pu} & \boldsymbol{C}_{pp} \end{pmatrix} \begin{pmatrix} \dot{\boldsymbol{d}}_u \\ \dot{\boldsymbol{d}}_p \end{pmatrix} + \begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{K}_{pp} \end{pmatrix} \begin{pmatrix} \boldsymbol{d}_u \\ \boldsymbol{d}_p \end{pmatrix} = \begin{pmatrix} \dot{\boldsymbol{f}}_u \\ \boldsymbol{f}_p \end{pmatrix}, \tag{8.61}$$

where $\boldsymbol{K}_{uu}$ is the stiffness matrix in the form

$$\boldsymbol{K}_{uu} = \int_{\Omega} \boldsymbol{B}_u^{\mathrm{T}} \boldsymbol{D} \boldsymbol{B}_u \, \mathrm{d}\Omega, \tag{8.62}$$

$\boldsymbol{K}_{pp}$ is the permeability matrix

$$\boldsymbol{K}_{pp} = \int_{\Omega} \boldsymbol{B}_p^{\mathrm{T}} \frac{k^{rw}}{\mu^w} \boldsymbol{k}_{\mathrm{sat}} \boldsymbol{B}_p \, \mathrm{d}\Omega, \tag{8.63}$$

$\boldsymbol{C}_{pp}$ is the compressibility matrix

$$\boldsymbol{C}_{pp} = \int_{\Omega} \boldsymbol{N}_p^{\mathrm{T}} \left(\frac{\alpha - n}{K_g} S_r \left(S_r + \frac{\partial S_r}{\partial p_w} p_w \right) + n \left(\frac{\partial S_r}{\partial p_w} + \frac{S_r}{K_w} \right) \right) \boldsymbol{N}_p \, \mathrm{d}\Omega, \tag{8.64}$$

and $\boldsymbol{C}_{up}$ and $\boldsymbol{C}_{pu}$ are the coupling matrices

$$\boldsymbol{C}_{up} = \int_{\Omega} \boldsymbol{B}_u^{\mathrm{T}} \boldsymbol{h} \boldsymbol{N}_p \, \mathrm{d}\Omega, \tag{8.65}$$

$$\tag{8.66}$$

$$\boldsymbol{C}_{pu} = -\int_{\Omega} \boldsymbol{N}_p^{\mathrm{T}} \alpha S_r \boldsymbol{m}^{\mathrm{T}} \boldsymbol{B}_u \, \mathrm{d}\Omega, \tag{8.67}$$

where $\boldsymbol{m}^{\mathrm{T}} = (1, 1, 1, 0, 0, 0)$. $\dot{\boldsymbol{f}}_u$ is the time derivative of the load vector

$$\dot{\boldsymbol{f}}_u = \int_{\Omega} \boldsymbol{N}_u^{\mathrm{T}} \dot{\boldsymbol{b}} \, \mathrm{d}\Omega + \int_{\Gamma_t} \boldsymbol{N}_u^{\mathrm{T}} \dot{\bar{\boldsymbol{t}}} \, \mathrm{d}\Gamma, \tag{8.68}$$

$\boldsymbol{f}_p$ is the flux vector

$$\boldsymbol{f}_p = \int_{\Omega} \boldsymbol{B}_p^{\mathrm{T}} \frac{k^{rw} \boldsymbol{k}_{\mathrm{sat}}}{\mu^w} \rho \boldsymbol{g} \, \mathrm{d}\Omega - \int_{\Gamma_N} \boldsymbol{N}_p^{\mathrm{T}} \frac{\boldsymbol{n}^{\mathrm{T}} \bar{\boldsymbol{J}}_w}{\rho^w} \, \mathrm{d}\Gamma. \tag{8.69}$$

For the sake of consistency, it is convenient to approximate the strains $\boldsymbol{\varepsilon}$ and the pore water pressure p_w by polynomials of the same degree. Strain-displacement equations then imply that the displacements should be approximated by a polynomial one order higher than the pore pressure ([Bathe, 1996], [Krejčí et al., 2014] and [Koudelka et al., 2011]). The system of differential equations (8.61) has to be integrated numerically. Time discretization is based on the v-form of the generalised trapezoidal method [Hughes, 1987]. The resulting system of algebraic equations is non-linear and the Newton-Raphson method [Bittnar and Šejnoha, 1996] has to be used at each time step. The hydro-mechanical model was implemented in SIFEL (source files can be found on the web page [SIFEL, 2022]) and the implementation details are described in [Koudelka et al., 2011].

8.4 Time integration of hypoplastic model

The time integration of (8.3) in hypoplastic models was investigated in papers [Tamagnini et al., 2000], [Tamagnini et al., 2013] which contain performance evaluation of different integration methods including simple forward Euler method, Crank-Nicolson scheme and Runge-Kutta-Fehlberg (RKF) methods with substepping. Authors concluded that the performance of the low-order methods was generally unsatisfactory and suggested the usage of the RKF method of suitable order with substepping. In [Janda, 2013], Runge-Kutta-Fehlberg methods with substepping of various orders was used for integration of models for clays and their performance was evaluated. There is also detailed comparison of time integration methods in [Ding et al., 2015] including implicit backward Euler and Crank-Nicholson scheme and explicit high order RKF Dormand-Prince scheme with substepping which was performed with the hypoplastic model proposed in [Gudehus, 1996] and [Bauer, 1996]. Authors of [Ding et al., 2015] observed generally unsatisfactory performance of low-order methods again both in terms of accuracy and efficiency and the source of difficulties was identified in barotropy factor, f_d, which causes high nonlinearity of the model. Implicit integration methods were considered in [Heeres and de Borst, 2000] but generally, there are difficulties with the expression of residual Jacobian for the Newton-Raphson iterative method.

Recall that in hypoplastic model described in Section 8.1, Equation (8.14), (8.16) and (8.27) define the total stress rate $\dot{\boldsymbol{\sigma}}$. Additionally, the hypoplastic model involves state variables given in the vector $\varkappa$ that can be also formulated in the rate form and thus the generalized stress rate, $\boldsymbol{\tau}^{\mathrm{T}} = (\boldsymbol{\sigma}, \varkappa)$, can be defined by

$$\dot{\boldsymbol{\tau}} = \boldsymbol{M}\dot{\boldsymbol{\epsilon}} = \boldsymbol{\Psi}\left(\boldsymbol{\tau}(t), \Delta\boldsymbol{\epsilon}(t)\right), \tag{8.70}$$

where $\boldsymbol{M}$ represents the generalized stiffness matrix, $\boldsymbol{\epsilon}$ is the generalized strain vector $\boldsymbol{\epsilon}^{\mathrm{T}} = (\boldsymbol{\varepsilon}, s)$ and $\boldsymbol{\Psi}$ represents the model response function on the given input of strain increment, $\Delta\boldsymbol{\epsilon}$, of the actual time step and attained stress level, $\boldsymbol{\tau}$.

With respect to the experiences and conclusions in papers [Tamagnini et al., 2000], [Janda, 2013] and [Ding et al., 2015], the explicit integration RKF algorithm with substepping was selected and implemented in open source code SIFEL ([Koudelka et al., 2011], [SIFEL, 2022]). Equation (8.70) represents initial value problem given by the set of ordinary differential equations. These equations can be written in generic substep k of RKF at time interval $[t_n; t_{n+1}]$ formally as follows

$$\boldsymbol{\tau}_{k+1} \;=\; \boldsymbol{\tau}_k + \Delta t_k \sum_{i=1}^{s} b_i\, \boldsymbol{k}_i\left(\boldsymbol{\tau}_k, \Delta\boldsymbol{\epsilon}(t_{n+1}), \Delta t_k\right), \tag{8.71}$$

where subscript n stands for the index of particular load steps, b_i is the RKF weight coefficient and $\Delta t_k \in (0; 1]$ is the dimensionless step length defined as follows

$$\Delta t_k = \frac{t_{k+1} - t_k}{t_{n+1} - t_n}. \tag{8.72}$$

Additionally, $\boldsymbol{k}_i\left(\boldsymbol{\tau}_k, \Delta\boldsymbol{\epsilon}(t_{n+1}), \Delta t_k\right)$ represents the function $\boldsymbol{\Psi}$ evaluated for the given strain increment of the actual time step $\Delta\boldsymbol{\epsilon}(t_{n+1}) = \boldsymbol{\epsilon}(t_{n+1}) - \boldsymbol{\epsilon}(t_n)$ and attained stress

0	0	0	$\cdots$	0	0
$\tilde{c}_2$	$\tilde{a}_{2,1}$	0	$\cdots$	0	0
$\vdots$	$\vdots$	$\ddots$	$\ddots$	$\vdots$	$\vdots$
$\tilde{c}_{s-1}$	$\tilde{a}_{s-1,1}$	$\tilde{a}_{s-1,2}$	$\cdots$	0	0
$\tilde{c}_s$	$\tilde{a}_{s,1}$	$\tilde{a}_{s,2}$	$\cdots$	$\tilde{a}_{s,s-1}$	0
	$\tilde{b}_1$	$\tilde{b}_2$	$\cdots$	$\tilde{b}_{s-1}$	$\tilde{b}_s$
	$\bar{b}_1$	$\bar{b}_2$	$\cdots$	$\bar{b}_{s-1}$	$\bar{b}_s$

Table 8.1: General form of Butcher table for Runge-Kutta-Fehlberg method with substepping.

levels at the prescribed points of time interval.

$$\boldsymbol{k}_i\left(\boldsymbol{\tau}_k, \Delta\boldsymbol{\epsilon}(t_{n+1}), \Delta t_k\right) = \boldsymbol{\Psi}\left(\boldsymbol{\tau}_k + \Delta t_k \sum_{j=1}^{i-1} \tilde{a}_{i,j}\boldsymbol{k}_j, \Delta\boldsymbol{\epsilon}(t_{n+1})\right). \tag{8.73}$$

The RKF matrix coefficients $\tilde{a}_{i,j}$ and weight coefficients b_i are selected so that the method provides the numerical approximation of the solution of order r and $r+1$. These coefficient may be summarized in the form of the Butcher table [Butcher, 2016] whose generalized example is given in Tab.(8.1). Generally, the Butcher table also contains nodal coefficients $\tilde{c}_i$ which would be used in the case that the function $\boldsymbol{\Psi}$ would depend on time directly.

In the Runge-Kutta-Fehlberg method, the step length Δt_k is constructed according to the difference between solutions to two embedded Runge-Kutta algorithms

$$\bar{\boldsymbol{\tau}}_{k+1} = \boldsymbol{\tau}_k + \Delta t_k \sum_{i=1}^{s} \bar{b}_i\, \boldsymbol{k}_i\left(\boldsymbol{\tau}_k, \Delta\boldsymbol{\epsilon}(t_{n+1}), \Delta t_k\right), \tag{8.74}$$

$$\tilde{\boldsymbol{\tau}}_{k+1} = \boldsymbol{\tau}_k + \Delta t_k \sum_{i=1}^{s} \tilde{b}_i\, \boldsymbol{k}_i\left(\boldsymbol{\tau}_k, \Delta\boldsymbol{\epsilon}(t_{n+1}), \Delta t_k\right), \tag{8.75}$$

where weight coefficents $\bar{b}_i$ and $\tilde{b}_i$ provide order r and $r+1$ of the solution accuracy, respectively. Generally, the substep $k+1$ is accepted if the relative error measure R_{k+1} of two solutions $\bar{\boldsymbol{\tau}}_{k+1}$ and $\tilde{\boldsymbol{\tau}}_{k+1}$ is less than the given tolerance ϑ. In this case, the selective error measures ${}^{\sigma}R$ and ${}^{\varkappa}R$ have to be introduced because of different magnitude of particular quantities in the vector $\boldsymbol{\tau}$

$${}^{\sigma}R_{k+1} = \frac{\|\tilde{\boldsymbol{\sigma}}_{k+1} - \bar{\boldsymbol{\sigma}}_{k+1}\|}{\|\tilde{\boldsymbol{\sigma}}_{k+1}\|}, \tag{8.76}$$

$${}^{\varkappa}R_{k+1} = \frac{\|\tilde{\boldsymbol{\varkappa}}_{k+1} - \bar{\boldsymbol{\varkappa}}_{k+1}\|}{\|\tilde{\boldsymbol{\varkappa}}_{k+1}\|}, \tag{8.77}$$

$$R_{k+1} = \sqrt{{}^{\sigma}R_{k+1}^2 + {}^{\varkappa}R_{k+1}^2} \leq \vartheta. \tag{8.78}$$

It should be noted that error measure ${}^{v}R_{k+1}$ may be calculated as a norm from the complete vector $\boldsymbol{\varkappa}$ because all variables are dimensionless, they have the same order with

0			
1/2	1/2		
1	-1	2	
	1/6	2/3	1/6
	0	1	0

0				
1/2	1/2			
3/4	0	3/4		
	2/9	1/3	4/9	
	7/24	1/4	1/3	1/8

Table 8.2: Butcher table for RKF-32 (left) and RKF-32bs Bogacki-Shampine (right).

0					
1/4	1/4				
4/9	4/81	32/81			
6/7	57/98	-432/343	1,053/686		
1	1/6	0	27/52	49/156	
	43/288	0	243/416	343/1,872	1/12
	1/6	0	27/52	49/156	0

Table 8.3: Butcher table for RKF-43.

values ranging in [0;1] or [0;2] for reasonable model input. If working ranges of the state variables varied more, it would be possible to use another approach described in [Fellin et al., 2009]. In practice, dominant influence on the total error had ${}^{\sigma}R_{k+1}$ and error of degree of saturation component of vector $\varkappa$.

If the value of a state variable evolves to zero, the absolute error measure should be used in order to avoid numerical difficulties. In this case, the absolute error measure was used in the case that denominator in (8.77) was less than prescribed treshold value (typically 10^{-4}). The more sophisticated approach with continuous transition between relative and absolute error measure can be found in [Fellin et al., 2009].

The optimum length of the next substep can be estimated by the following term

$$\Delta t_{k+1} = \min\left\{0.9\left(\frac{\vartheta}{R_{k+1}}\right)^{r+1}, 1-\Delta t_k\right\}. \tag{8.79}$$

Several time integration RKF schemes were implemented for the time integration of equation (8.70). Their description in the form of Butcher's tables is given in Tabs.(8.2), (8.3) and (8.4). It should be noted that the algorithm RKF-32bs is the Bogacki-Shampine coefficient pairs proposed in [Bogacki and Shampine, 1989] and the advantage of the method is that it provides the better estimate of error with the minimum cost. In the second order formula, the vector $\boldsymbol{k}_4$ is needed which is evaluated at $\Psi(\tilde{\boldsymbol{\tau}}_k)$ and thus it can be used as $\boldsymbol{k}_1$ in the next step if the tolerance condition is met (First Same As Last concept - FSAL).

8.4.1 Performance of integration schemes on benchmarks

The implemented hypoplastic model was tested on simple benchmarks with axisymmetric specimen 1x1 m subjected to various loading paths:

0						
1/4	1/4					
3/8	3/32	9/32				
12/13	1,932/2,197	-7,200/2,197	7,296/2,197			
1	439/216	-8	3,680/513	-845/4,104		
1/2	-8/27	2	-3,544/2,565	1,859/4,104	-11/40	
	16/135	0	6,656/12,825	28,561/56,430	-9/50	2/55
	25/216	0	1,408/2,565	2,197/4,101	-1/5	0

Table 8.4: Butcher table for RKF-54.

1. Triaxial drained test with constant confining pressure and axial load generated by gradually increasing vertical prescribed displacement ranging from zero at pseudo-time 0.0 to 0.02 m at pseudo-time 2,000, initial stress -200 kPa, constant suction -1.9 MPa.

2. Triaxial drained test with constant volume increment generated by vertical prescribed displacement ranging from zero at pseudo-time 0.0 to 0.2 m at pseudo-time 20,000 and lateral prescribed displacements ranging from zero at pseudo-time 0.0 to 0.1 m at pseudo-time 20,000, initial stress -200 kPa, constant suction -1.9 MPa.

3. Triaxial test with constant pressure and variable suction growing from -1.9 MPa at pseudo-time 0 to -5.0 MPa at pseudo-time 3,000 and then the suction is decreased to -1.0 MPa at pseudo-time 7,100.

All benchmarks were tested with the set of model parameters given in Table 8.5 and initial values of state varibales given by e=0.6 and a_{sc}=0.5. Resulting diagrams from the first

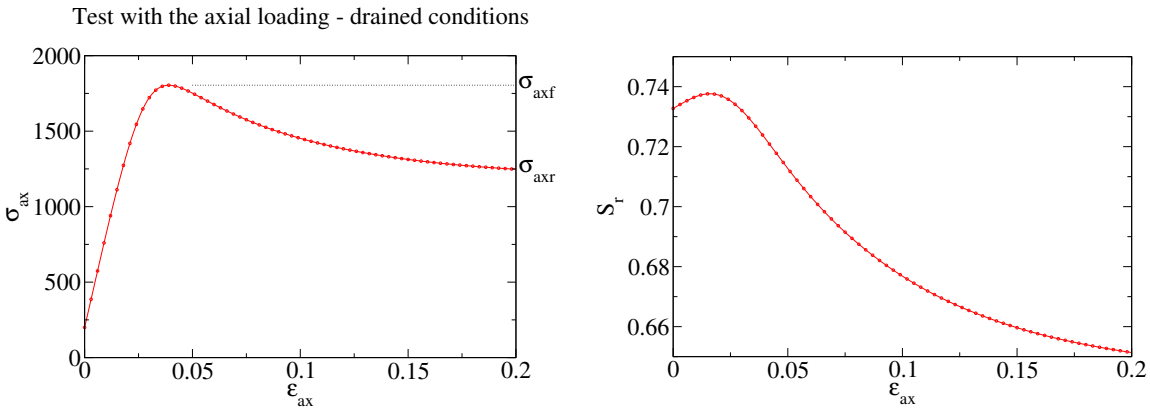
φ_c	N	λ^*	κ^*	n_s	l_s	e_r^m
27°	1.05	0.08	0.008	0.025	0.0	0.38

s_r	r	m	κ_m	s_{e0}	e_0^M	a_e
-2,400 kPa	0.4	2.0	0.04	-200.0 kPa	0.18	0.25

Table 8.5: Set of model parameters used in benchmark examples

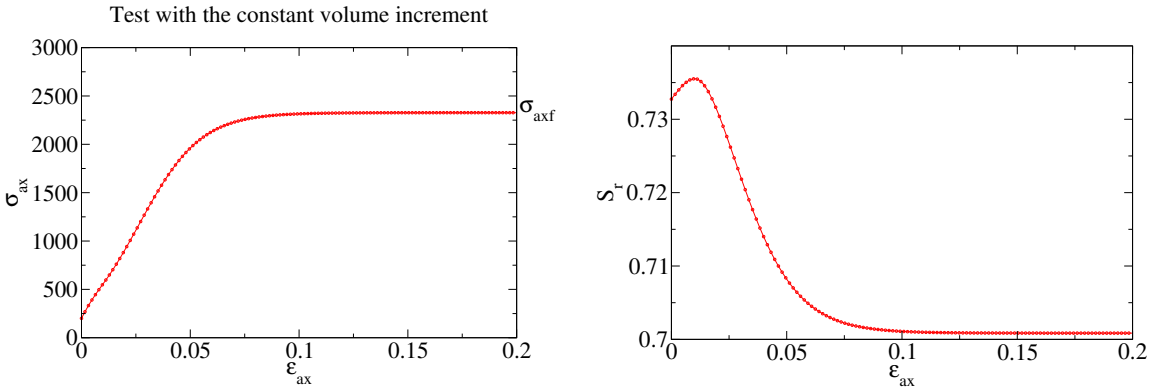
benchmark can be seen in Figure (8.2) and from the second benchmark in Figure (8.3), where evolution of the axial stresses (a) or the degree of saturation (b) versus evolution of the axial strains can be observed. For the third benchmark, only evolution of degree of saturation is depicted in Figure (8.4) because the stress value is kept constant during whole benchmark example.

In benchmark 1, all mentioned integration schemes were used with two pseudo-time step lengths 20 and 100 and tolerance, ϑ, for RKF integration ranging from 10^{-7} to 10^{-3}. Results were compared with the reference solution obtained from RKF54 scheme with pseudo-time step length 1. In this benchmark example, three error indicators were used for the comparison of performance of particular integration schemes:



(a) (b)

Figure 8.2: Benchmark 1 - (a) axial stress, σ_{ax}, vs. axial strain, ε_{ax}, (b) evolution of degree of saturation, S_r, according to axial strain, ε_{ax}.



(a) (b)

Figure 8.3: Benchmark 2 - (a) axial stress, σ_{ax}, vs. axial strain, ε_{ax}, (b) evolution of degree of saturation, S_r, according to axial strain, ε_{ax}.

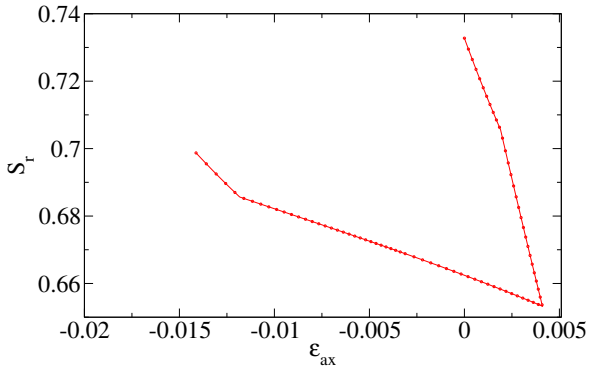


Figure 8.4: Benchmark 3 - evolution of degree of saturation, S_r, according to axial strain, ε_{ax}.

- ξ_f relative error of peak axial stress value, σ_{axf}, in Figure (8.2a),

$$\xi_f = \frac{\sigma_{axf}^{\vartheta} - \sigma_{axf}^{ref}}{\sigma_{axf}^{ref}} \tag{8.80}$$

- ξ_r relative error of residual stress value, σ_{axr}, in Figure (8.2a),

$$\xi_r = \frac{\sigma_{axr}^{\vartheta} - \sigma_{axr}^{ref}}{\sigma_{axr}^{ref}} \tag{8.81}$$

- ξ_W relative error of stress path integral, W, of the diagram Figure (8.2a),

$$\xi_W = \frac{\int \sigma_{ax}^{\vartheta} \dot{\varepsilon}_{ax}^{\vartheta} \mathrm{d}t - \int \sigma_{ax}^{ref} \dot{\varepsilon}_{ax}^{ref} \mathrm{d}t}{\int \sigma_{ax}^{ref} \dot{\varepsilon}_{ax}^{ref} \mathrm{d}t}. \tag{8.82}$$

In the error indicator definitions, superscripts $^{\vartheta}$ and ^{ref} denote solution obtained from RKF with tolerance ϑ and reference solution, respectively.

The total number of model evaluation n_{ev} was monitored together with elapsed computational time of whole problem solution t_c. The results are summarized in Tab. 8.6 and 8.7.

In order to test the integration scheme, a specimen with highly nonlinear behaviour was used in benchmark 1 (see Figure 8.2a) and therefore relatively large number of substeps was necessary in all of them (see n_{ev} values). Tests revealed that all integration schemes attained similar orders of error and their magnitudes were sufficient for engineering problems in all cases. It can be observed that the higher order methods do not attain significantly better results than the low order ones and more important point is the selection of the pseudo-time step length which influences the selected error measures more significantly. From the viewpoint of computational time, the most reasonable choice represented RKF-32bs scheme where the lowest numbers of model evaluation were attained as well as the total computational times.

In benchmark 2, all mentioned integration schemes were used with two pseudo-time step lengths 100 and 1,000 and tolerances for RKF integration ranging from 10^{-7} to 10^{-3}. Results were compared with the reference solution obtained by step length 1 and RKF-54 scheme. In this benchmark, only error indicators ξ_f and ξ_W were used for the performance comparison which is summarized in Tab. 8.8 and 8.9.

In benchmark 2, the integration schemes were tested on nonlinear behaviour of the specimen where all DOFs were controlled directly which led to relatively low numbers of substeps. Similarly to example 1, all integration schemes attained errors that were acceptable for engineering problems. For large pseudo-time step length, the high order methods led to higher errors for most values of tolerances than the low ones while in the case of low pseudo-time step length, the error orders were comparable for all schemes. From the viewpoint of computational time, the most reasonable choice is represented by RKF-32bs scheme, where the lowest numbers of model evaluation were attained in most cases as well as the total computational times.

All mentioned integration schemes were tested in benchmark 3 with two pseudo-time step lengths 10 and 100 and tolerances of RKF integration ranging from 10^{-7} to 10^{-3}. In

ϑ	Error measure type	RKF-32	RKF-32bs	RKF-43	RKF-54
10^{-7}	ξ_f	$1.0780{\cdot}10^{-2}$	$1.0780{\cdot}10^{-2}$	$1.0780{\cdot}10^{-2}$	$1.0872{\cdot}10^{-2}$
	ξ_r	$1.2754{\cdot}10^{-3}$	$1.2758{\cdot}10^{-3}$	$1.2750{\cdot}10^{-3}$	$1.2756{\cdot}10^{-3}$
	ξ_W	$4.0021{\cdot}10^{-3}$	$4.0020{\cdot}10^{-3}$	$4.0020{\cdot}10^{-3}$	$3.9972{\cdot}10^{-3}$
	n_{ev}	9,672,960	5,062,512	6,638,370	18,692,256
	t_c [s]	121.44	65.35	72.66	234.77
10^{-6}	ξ_f	$1.0780{\cdot}10^{-2}$	$1.0782{\cdot}10^{-2}$	$1.0780{\cdot}10^{-2}$	$1.1057{\cdot}10^{-2}$
	ξ_r	$1.2759{\cdot}10^{-3}$	$1.2769{\cdot}10^{-3}$	$1.2748{\cdot}10^{-3}$	$1.2700{\cdot}10^{-3}$
	ξ_W	$4.0020{\cdot}10^{-3}$	$4.0016{\cdot}10^{-3}$	$4.0019{\cdot}10^{-3}$	$3.9882{\cdot}10^{-3}$
	n_{ev}	4,265,280	2,314,400	3,225,920	6,617,088
	t_c [s]	56.54	34.05	35.69	83.29
10^{-5}	ξ_f	$1.0781{\cdot}10^{-2}$	$1.0801{\cdot}10^{-2}$	$1.0777{\cdot}10^{-2}$	$1.1568{\cdot}10^{-2}$
	ξ_r	$1.3351{\cdot}10^{-3}$	$2.5810{\cdot}10^{-3}$	$1.3310{\cdot}10^{-3}$	$3.1754{\cdot}10^{-3}$
	ξ_W	$2.1396{\cdot}10^{-3}$	$3.6643{\cdot}10^{-3}$	$2.1398{\cdot}10^{-3}$	$3.8722{\cdot}10^{-3}$
	n_{ev}	2,092,704	1,340,224	1,738,400	2,532,192
	t_c [s]	25.09	16.38	18.31	30.90
10^{-4}	ξ_f	$1.0805{\cdot}10^{-2}$	$1.1064{\cdot}10^{-2}$	$1.0692{\cdot}10^{-2}$	$1.2399{\cdot}10^{-2}$
	ξ_r	$4.5567{\cdot}10^{-3}$	$1.9902{\cdot}10^{-3}$	$1.3333{\cdot}10^{-3}$	$1.4202{\cdot}10^{-3}$
	ξ_W	$4.0003{\cdot}10^{-3}$	$2.2976{\cdot}10^{-3}$	$4.0070{\cdot}10^{-3}$	$2.0429{\cdot}10^{-3}$
	n_{ev}	1,024,128	829,136	814,800	841,728
	t_c [s]	13.39	9.32	11.80	10.90
10^{-3}	ξ_f	$1.0905{\cdot}10^{-2}$	$1.1826{\cdot}10^{-2}$	$1.0522{\cdot}10^{-2}$	$1.3743{\cdot}10^{-2}$
	ξ_r	$1.5118{\cdot}10^{-3}$	$2.2620{\cdot}10^{-3}$	$6.1085{\cdot}10^{-4}$	$4.1522{\cdot}10^{-3}$
	ξ_W	$3.9983{\cdot}10^{-3}$	$3.9095{\cdot}10^{-3}$	$4.0415{\cdot}10^{-3}$	$2.4129{\cdot}10^{-3}$
	n_{ev}	475,488	298,384	486,000	578,880
	t_c [s]	6.87	4.77	6.74	7.37

Table 8.6: Performance comparison of RKF schemes for benchmark 1 and step length 100.

ϑ	Error measure type	RKF-32	RKF-32bs	RKF-43	RKF-54
10^{-7}	ξ_f	$3.0361{\cdot}10^{-3}$	$3.0360{\cdot}10^{-3}$	$3.0354{\cdot}10^{-3}$	$3.1202{\cdot}10^{-3}$
	ξ_r	$4.4944{\cdot}10^{-4}$	$5.8079{\cdot}10^{-4}$	$5.7955{\cdot}10^{-4}$	$5.9941{\cdot}10^{-4}$
	ξ_W	$2.5149{\cdot}10^{-4}$	$2.9683{\cdot}10^{-4}$	$2.9688{\cdot}10^{-4}$	$2.9487{\cdot}10^{-4}$
	n_{ev}	7,843,824	3,722,352	4,297,920	13,085,400
	t_c [s]	114.31	50.48	52.96	169.84
10^{-6}	ξ_f	$3.0347{\cdot}10^{-3}$	$3.0375{\cdot}10^{-3}$	$3.0306{\cdot}10^{-3}$	$3.2732{\cdot}10^{-3}$
	ξ_r	$5.4006{\cdot}10^{-4}$	$5.8280{\cdot}10^{-4}$	$5.7929{\cdot}10^{-4}$	$6.2511{\cdot}10^{-4}$
	ξ_W	$2.9676{\cdot}10^{-4}$	$2.9677{\cdot}10^{-4}$	$2.9685{\cdot}10^{-4}$	$2.9143{\cdot}10^{-4}$
	n_{ev}	2,586,960	1,495,376	1,705,520	4,509,408
	t_c [s]	34.13	19.65	22.85	63.33
10^{-5}	ξ_f	$3.0248{\cdot}10^{-3}$	$2.2873{\cdot}10^{-3}$	$1.7388{\cdot}10^{-3}$	$3.1159{\cdot}10^{-3}$
	ξ_r	$5.8107{\cdot}10^{-4}$	$3.5279{\cdot}10^{-4}$	$6.3580{\cdot}10^{-5}$	$6.2763{\cdot}10^{-4}$
	ξ_W	$2.9691{\cdot}10^{-4}$	$1.7255{\cdot}10^{-4}$	$8.9412{\cdot}10^{-5}$	$2.8055{\cdot}10^{-4}$
	n_{ev}	1,316,976	970,896	107,390,000	1,261,536
	t_c [s]	17.79	15.99	2106.05	20.2
10^{-4}	ξ_f	$2.2874{\cdot}10^{-3}$	$3.1482{\cdot}10^{-3}$	$2.0848{\cdot}10^{-3}$	$3.1482{\cdot}10^{-3}$
	ξ_r	$3.5025{\cdot}10^{-4}$	$6.8376{\cdot}10^{-4}$	$4.2193{\cdot}10^{-4}$	$6.8376{\cdot}10^{-4}$
	ξ_W	$1.7286{\cdot}10^{-4}$	$2.8738{\cdot}10^{-4}$	$1.1459{\cdot}10^{-4}$	$2.8738{\cdot}10^{-4}$
	n_{ev}	843,168	378,800	465,479,760	448,896
	t_c [s]	14.27	6.97	9979.09	7.68
10^{-3}	ξ_f	$3.2799{\cdot}10^{-3}$	$3.2714{\cdot}10^{-3}$	$3.1474{\cdot}10^{-3}$	$3.2714{\cdot}10^{-3}$
	ξ_r	$6.9036{\cdot}10^{-4}$	$6.8332{\cdot}10^{-4}$	$5.8743{\cdot}10^{-4}$	$6.8332{\cdot}10^{-4}$
	ξ_W	$2.8732{\cdot}10^{-4}$	$2.8930{\cdot}10^{-4}$	$2.9549{\cdot}10^{-4}$	$2.8930{\cdot}10^{-4}$
	n_{ev}	321,024	298,816	373,680	448,896
	t_c [s]	6.44	5.92	6.85	7.86

Table 8.7: Performance comparison of RKF schemes for benchmark 1 and step length 20.

ϑ	Error measure type	RKF-32	RKF-32bs	RKF-43	RKF-54
10^{-7}	ξ_f	$6.1594{\cdot}10^{-6}$	$6.0511{\cdot}10^{-6}$	$6.1509{\cdot}10^{-6}$	$9.3072{\cdot}10^{-5}$
	ξ_W	$6.3023{\cdot}10^{-5}$	$1.4820{\cdot}10^{-5}$	$1.4799{\cdot}10^{-5}$	$4.4048{\cdot}10^{-4}$
	n_{ev}	34,896	18,208	17,760	119,862
	t_c [s]	0.52	0.29	0.27	1.76
10^{-6}	ξ_f	$6.0729{\cdot}10^{-6}$	$5.6872{\cdot}10^{-6}$	$6.1620{\cdot}10^{-6}$	$1.0468{\cdot}10^{-4}$
	ξ_W	$1.4786{\cdot}10^{-5}$	$1.4666{\cdot}10^{-5}$	$1.4812{\cdot}10^{-5}$	$4.1186{\cdot}10^{-4}$
	n_{ev}	15,648	8,368	8,400	41,376
	t_c [s]	0.26	0.15	0.20	0.61
10^{-5}	ξ_f	$5.6926{\cdot}10^{-6}$	$4.6421{\cdot}10^{-6}$	$6.1697{\cdot}10^{-6}$	$1.3709{\cdot}10^{-4}$
	ξ_W	$1.4454{\cdot}10^{-5}$	$1.3311{\cdot}10^{-5}$	$1.5625{\cdot}10^{-5}$	$3.3540{\cdot}10^{-4}$
	n_{ev}	7,536	4,528	5,280	14,496
	t_c [s]	0.14	0.09	0.10	0.24
10^{-4}	ξ_f	$6.3561{\cdot}10^{-6}$	$1.2246{\cdot}10^{-5}$	$6.1603{\cdot}10^{-6}$	$1.9919{\cdot}10^{-4}$
	ξ_W	$1.4210{\cdot}10^{-5}$	$7.2751{\cdot}10^{-6}$	$1.2791{\cdot}10^{-5}$	$1.9088{\cdot}10^{-4}$
	n_{ev}	3,840	2,656	3,440	6,144
	t_c [s]	0.09	0.06	0.08	0.12
10^{-3}	ξ_f	$1.2796{\cdot}10^{-6}$	$3.5550{\cdot}10^{-4}$	$5.9340{\cdot}10^{-6}$	$2.4109{\cdot}10^{-4}$
	ξ_W	$7.9662{\cdot}10^{-6}$	$6.7320{\cdot}10^{-5}$	$1.5685{\cdot}10^{-5}$	$1.3272{\cdot}10^{-4}$
	n_{ev}	2,160	1,792	2,240	3,360
	t_c [s]	0.06	0.06	0.06	0.08

Table 8.8: Performance comparison of RKF schemes for benchmark 2 and step length 1,000.

ϑ	Error measure type	RKF-32	RKF-32bs	RKF-43	RKF-54
10^{-7}	ξ_f	$9.0659{\cdot}10^{-6}$	$9.0820{\cdot}10^{-6}$	$9.0811{\cdot}10^{-6}$	$1.1202{\cdot}10^{-5}$
	ξ_W	$1.1800{\cdot}10^{-5}$	$1.1800{\cdot}10^{-5}$	$1.1812{\cdot}10^{-5}$	$9.8569{\cdot}10^{-6}$
	n_{ev}	67,200	39,040	45,600	242,088
	t_c [s]	1.26	0.82	0.90	3.78
10^{-6}	ξ_f	$9.0819{\cdot}10^{-6}$	$9.0816{\cdot}10^{-6}$	$9.0810{\cdot}10^{-6}$	$1.5101{\cdot}10^{-5}$
	ξ_W	$1.1805{\cdot}10^{-5}$	$1.1839{\cdot}10^{-5}$	$1.1767{\cdot}10^{-5}$	$6.5142{\cdot}10^{-6}$
	n_{ev}	34,896	22,048	29,520	86,778
	t_c [s]	0.76	0.55	0.66	1.63
10^{-5}	ξ_f	$9.0792{\cdot}10^{-6}$	$9.0442{\cdot}10^{-6}$	$9.0812{\cdot}10^{-6}$	$9.0804{\cdot}10^{-6}$
	ξ_W	$1.1889{\cdot}10^{-5}$	$1.1900{\cdot}10^{-5}$	$1.2077{\cdot}10^{-5}$	$1.1951{\cdot}10^{-5}$
	n_{ev}	19,152	14,128	19,680	19,104
	t_c [s]	0.52	0.44	0.53	0.51
10^{-4}	ξ_f	$9.0596{\cdot}10^{-6}$	$9.0044{\cdot}10^{-6}$	$9.0811{\cdot}10^{-6}$	$9.0804{\cdot}10^{-6}$
	ξ_W	$1.2016{\cdot}10^{-5}$	$1.1306{\cdot}10^{-5}$	$1.1748{\cdot}10^{-5}$	$1.1951{\cdot}10^{-5}$
	n_{ev}	12,528	12,736	15,920	19,104
	t_c [s]	0.45	0.41	0.48	0.49
10^{-3}	ξ_f	$9.0078{\cdot}10^{-6}$	$9.0044{\cdot}10^{-6}$	$9.0811{\cdot}10^{-6}$	$9.0804{\cdot}10^{-6}$
	ξ_W	$1.1678{\cdot}10^{-5}$	$1.1306{\cdot}10^{-5}$	$1.1748{\cdot}10^{-5}$	$1.1951{\cdot}10^{-5}$
	n_{ev}	9,552	12,736	15,920	19,104
	t_c [s]	0.41	0.44	0.45	0.49

Table 8.9: Performance comparison of RKF schemes for benchmark 2 and step length 100.

this benchmark example, another two error indicators were estimated for the comparison of performance of particular integration schemes with respect to different test conditions (constant stresses and suction variation):

- ξ_{E_i} relative error of stress path integral of the given tolerance and reference solutions

$$\xi_{E_i} = \frac{\int \boldsymbol{\sigma}^{\vartheta} : \dot{\boldsymbol{\varepsilon}}^{\vartheta} \, \mathrm{d}t - \int \boldsymbol{\sigma}^{ref} : \dot{\boldsymbol{\varepsilon}}^{ref} \, \mathrm{d}t}{\int \boldsymbol{\sigma}^{ref} : \dot{\boldsymbol{\varepsilon}}^{ref} \, \mathrm{d}t},$$

- ξ_{Sr} relative error of degree of saturation for the given tolerance and reference solutions

$$\xi_{S_r} = \frac{\int \dot{S_r}^{\vartheta} \, \mathrm{d}t - \int \dot{S_r}^{ref} \, \mathrm{d}t}{\int \dot{S_r}^{ref} \, \mathrm{d}t}.$$

Performed analyses revealed numerical problems connected with the definition of hydraulic part of the hypoplastic model. Therefore the results were compared with the reference solution obtained by step length 0.1 and integration with the help of simple forward Euler scheme.

The source of difficulties in the hydraulic part of the model is the definition of state variable a_{sc} in the form of piecewise linear function. The state variable controls the evolution of S_r^M which is the main source of the stress rates in the hypoplastic part of the model in this benchmark and it also causes the nonlinear behaviour. The rates of state variables $\dot{\varkappa}$ connected with the hydraulic part of the model are therefore given as increments, $\Delta\varkappa$, for the given time increment Δt.

In the RKF method, the function $\boldsymbol{\Psi}$ should be differentiable, and special treatment must be given if the condition is not met. For example, the quality of the solution for a conventional elastoplastic model by RKF depends on the accurate detection of the point of yielding from which starts the evolution of plastic strains given by (8.2) (see [Sloan, 1987], [Büttner and Simeon, 2002], [Hiley and Rouainia, 2008]).

In the case of the hydraulic part of the model, two possible solutions to numerical problems were proposed. In the first investigations, a smooth approximation of piecewise linear function were considered. Tests of approximation described by (8.36) for artificial ramp function showed that error of the approximation can be simply controlled by the parameter c_α, for the higher value of parameter, the better approximation in vicinity of singular points of original function can be attained. Problems raised with the allowed range of argument values in exponential function. For the `double` precision type (64 bit), the maximum value of `exp` function argument is about 710 in the C++ standard library.

The problem can be overcome partially in the linear part if the normalized suction values would be used ($s_r = s/s_e$) but the maximum value of suction cannot be determined generally and thus the problem is not fixed for constant values $a_{sc} = 1$ and higher values of suction. Unfortunately, the model response is very sensitive to the a_{sc} value and differences in magnitude 10^{-4} still lead to a significant error of S_r^M. The problem with `exp` argument range can be resolved partially by adoption of `long double` precision type whose working range is extended to 80 bit on many compilers and platforms but unfortunately, the most used Windows compiler MS Visual C++ does not support extended precision for `long double` and therefore this code modification is not fully portable.

The extended data type precision can be combined with the constant limit values of a_{sc} if $a_{sc} \notin [10^{-7}; 0.9999999]$ which provides reasonable values of time derivatives with respect to RKF.

Another solution to difficulties with the hydraulic part of the model represents selective integration scheme for a_{sc} state variable in the vicinity of its limit values. The point is that the RKF method is substituted by simple forward Euler scheme just in the vicinity of singular points and if the constant value of a_{sc} is attained, the computation proceeds with the original RKF scheme. Decision which integration scheme should be applied can be made according to the condition

$$\bar{a}_{sc} = \tau_{k,ia} + k_{1,ia}\,\Delta t_k \tag{8.83}$$

$$\begin{aligned}
&\textbf{if}(1 - \bar{a}_{sc} < 10^{-7} \wedge 1 - \tau_{k,ia} > 10^{-7}) \vee \\
&\quad (\bar{a}_{sc} < 10^{-7} \wedge \tau_{k,ia} > 10^{-7}) \\
&\qquad \text{apply forward Euler scheme} \\
&\textbf{else} \\
&\qquad \text{apply original RKF scheme}
\end{aligned} \tag{8.84}$$

where $\tau_{k,ia}$ and $k_{1,ia}$ are components of vectors defined in (8.71) and (8.73), index k denotes generic substep of RKF used and index ia is the index of the vector component which corresponds to the a_{sc} state variable.

The performance comparison is summarized in Tab. 8.10 and 8.11. For low pseudo-time step length, the high order methods RKF-54 led to high error for degree of saturation in the case of more strict tolerances than the low ones. In the case of larger pseudo-time step length, the errors of degree of saturation in RKF32 were high for most of tolerances. The most reasonable choice is represented by RKF-32bs scheme, where the reasonable numbers of model evaluation were attained in most cases as well as the total computational times and the solution errors were acceptable in all cases. It can be concluded that the hydraulic part of the model with hysteretic behaviour caused numerical problems in many cases. Required tolerance and suitable integration scheme should be selected carefully because the more strict tolerance need not to lead to the better results and similarly, the higher order of the integration scheme does not provided better results in many cases.

8.4.2 Performance of integration schemes on the strip footing problem

Integration schemes were also tested on a problem of strip footing settlement. In this case, the 2D block 20x10 m of soil was assumed with flexible strip footing of the width 4 m placed on the surface. Uniform loading, f, was applied on the whole width of the strip and its magnitude was increased gradually from 0.0 to 400 kPa. The suction pressure was kept constant and the Dirichlet boundary conditions were applied on the block bottom and lateral edges. Because of problem symmetry along the vertical axis, only one half of the problem can be solved and thus the bottom was fixed fully while the left and right edges were fixed in horizontal direction only. The problem was discretized by structured

ϑ	Error measure type	RKF-32	RKF-32bs	RKF-43	RKF-54
10^{-7}	ξ_{E_i}	$5.7806{\cdot}10^{-3}$	$5.9820{\cdot}10^{-3}$	$5.9912{\cdot}10^{-3}$	$3.2441{\cdot}10^{-2}$
	ξ_{Sr}	$4.2915{\cdot}10^{-2}$	$5.0229{\cdot}10^{-4}$	$5.0490{\cdot}10^{-4}$	$4.7744{\cdot}10^{-4}$
	n_{ev}	428,418	29,841,584	100,376,560	7,016,160
	t_c [s]	9.75	661.62	2189.50	156.18
10^{-6}	ξ_{E_i}	$5.9610{\cdot}10^{-3}$	$5.8488{\cdot}10^{-3}$	$5.9914{\cdot}10^{-3}$	$1.9919{\cdot}10^{-2}$
	ξ_{Sr}	$2.3876{\cdot}10^{-2}$	$4.6884{\cdot}10^{-4}$	$5.0437{\cdot}10^{-4}$	$5.0297{\cdot}10^{-4}$
	n_{ev}	233,157	3,003,776	10,075,120	1,308,096
	t_c [s]	5.58	66.40	220.72	23.62
10^{-5}	ξ_{E_i}	$5.9963{\cdot}10^{-3}$	$5.4144{\cdot}10^{-3}$	$6.0136{\cdot}10^{-3}$	$3.4408{\cdot}10^{-4}$
	ξ_{Sr}	$8.7505{\cdot}10^{-3}$	$1.7924{\cdot}10^{-4}$	$5.0365{\cdot}10^{-4}$	$4.9881{\cdot}10^{-4}$
	n_{ev}	70,176	319,072	103,360	442,560
	t_c [s]	1.78	7.27	23.63	9.29
10^{-4}	ξ_{E_i}	$5.6551{\cdot}10^{-3}$	$1.4363{\cdot}10^{-2}$	$5.9507{\cdot}10^{-3}$	$5.3134{\cdot}10^{-3}$
	ξ_{Sr}	$4.3929{\cdot}10^{-3}$	$2.9762{\cdot}10^{-4}$	$4.3212{\cdot}10^{-4}$	$4.2241{\cdot}10^{-4}$
	n_{ev}	41,712	65,184	138,080	93,216
	t_c [s]	1.15	1.72	3.29	2.31
10^{-3}	ξ_{E_i}	$7.2703{\cdot}10^{-3}$	$2.1362{\cdot}10^{-2}$	$7.0266{\cdot}10^{-3}$	$6.4422{\cdot}10^{-3}$
	ξ_{Sr}	$9.7935{\cdot}10^{-4}$	$3.7122{\cdot}10^{-3}$	$3.1784{\cdot}10^{-4}$	$3.7816{\cdot}10^{-4}$
	n_{ev}	32,352	38,624	51,760	51,264
	t_c [s]	0.95	1.09	1.39	1.36

Table 8.10: Performance comparison of RKF schemes for benchmark 3 and step length 100.

ϑ	Error measure type	RKF-32	RKF-32bs	RKF-43	RKF-54
10^{-7}	ξ_{E_i}	$5.9483{\cdot}10^{-3}$	$5.9413{\cdot}10^{-3}$	$5.9520{\cdot}10^{-3}$	$2.0944{\cdot}10^{-2}$
	ξ_{Sr}	$4.8772{\cdot}10^{-4}$	$4.9594{\cdot}10^{-4}$	$4.9712{\cdot}10^{-4}$	$2.9649{\cdot}10^{-2}$
	n_{ev}	301,440	24,157,264	81,114,080	12,944,160
	t_c [s]	8.87	513.27	1701.82	276.37
10^{-6}	ξ_{E_i}	$5.9976{\cdot}10^{-3}$	$5.9720{\cdot}10^{-3}$	$5.9524{\cdot}10^{-3}$	$8.3811{\cdot}10^{-3}$
	ξ_{Sr}	$4.9794{\cdot}10^{-4}$	$4.8491{\cdot}10^{-4}$	$4.9667{\cdot}10^{-4}$	$1.1223{\cdot}10^{-2}$
	n_{ev}	215,664	2,515,392	8,266,880	889,248
	t_c [s]	6.97	55.85	176.75	21.34
10^{-5}	ξ_{E_i}	$5.9608{\cdot}10^{-3}$	$5.8803{\cdot}10^{-3}$	$5.9560{\cdot}10^{-3}$	$6.1040{\cdot}10^{-3}$
	ξ_{Sr}	$4.9758{\cdot}10^{-4}$	$3.7839{\cdot}10^{-4}$	$4.9084{\cdot}10^{-4}$	$1.5820{\cdot}10^{-4}$
	n_{ev}	373,968	332,064	969,520	418,272
	t_c [s]	11.52	9.47	22.89	11.40
10^{-4}	ξ_{E_i}	$6.0627{\cdot}10^{-3}$	$7.7612{\cdot}10^{-3}$	$4.6247{\cdot}10^{-3}$	$5.4960{\cdot}10^{-3}$
	ξ_{Sr}	$4.9394{\cdot}10^{-4}$	$4.6384{\cdot}10^{-6}$	$3.3411{\cdot}10^{-4}$	$2.6177{\cdot}10^{-4}$
	n_{ev}	193,680	259,664	321,520	377,088
	t_c [s]	6.54	7.94	9.25	10.66
10^{-3}	ξ_{E_i}	$5.5285{\cdot}10^{-3}$	$5.9941{\cdot}10^{-3}$	$4.0815{\cdot}10^{-3}$	$6.0154{\cdot}10^{-3}$
	ξ_{Sr}	$4.5472{\cdot}10^{-4}$	$4.9827{\cdot}10^{-4}$	$3.0691{\cdot}10^{-4}$	$4.9691{\cdot}10^{-4}$
	n_{ev}	187,872	249,536	311,600	374,112
	t_c [s]	6.44	7.71	9.04	10.37

Table 8.11: Performance comparison of RKF schemes for benchmark 3 and step length 10.

FE mesh with linear quadrilateral plane-strain elements, where the mesh density was decreased according to the increasing distance from the footing in three steps. The mesh and loading of the strip are depicted in Figure 8.5.

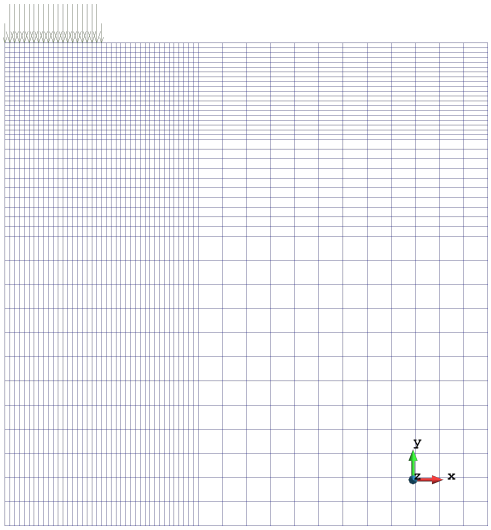
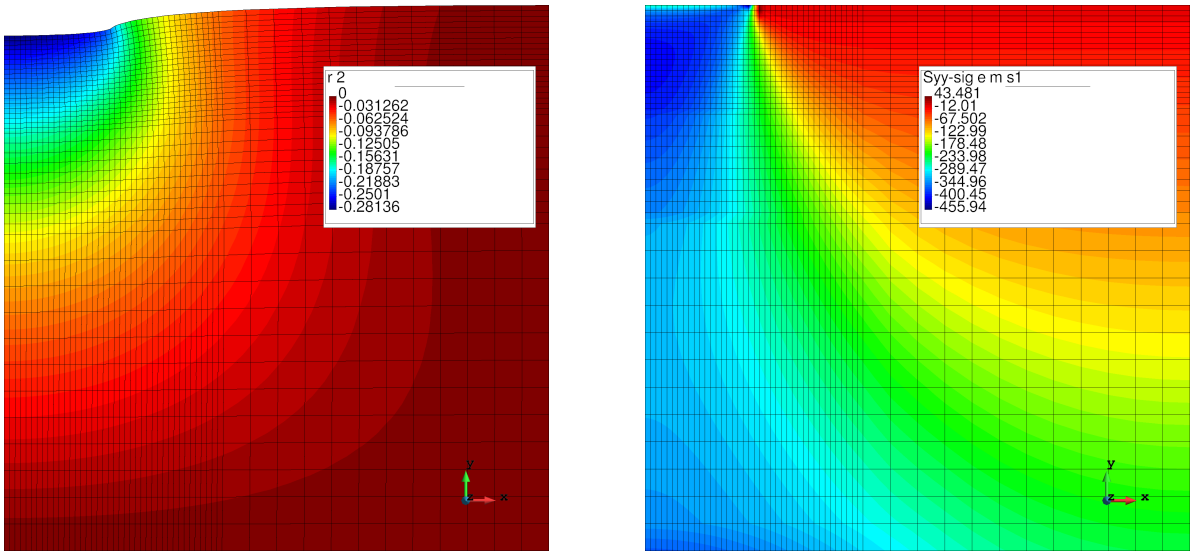


Figure 8.5: FE mesh and loading of footing strip problem.

There were 2,279 nodes and 2,184 elements in the FE mesh which results in 4,368 number of degrees of freedom. Resulting distribution of vertical displacement is depicted in Figure 8.6a while the vertical stress components, σ_y, is being captured in Figure 8.6b, respectively. Figure 8.7 represents diagram of evolution of vertical nodal displacement at



(a) (b)

Figure 8.6: Distribution of the vertical displacement, w, on the deformed mesh (a) and, σ_y, component (b).

the middle of the footing strip.

Settlement problem was tested with all implemented integration schemes and tolerances for RKF integration ranging from 10^{-7} to 10^{-3}. The pseudo-time step length was

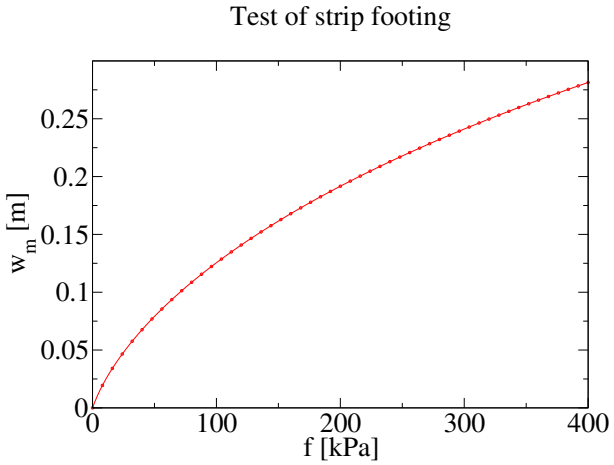


Figure 8.7: Diagram of settlement in dependence on applied loading pressure.

controlled automatically in the Newton-Raphson iteration solver. Results were compared with the reference solution obtained by RKF-54 scheme for $\vartheta = 10^{-7}$ and the error indicator ξ_w was established by the difference between settlement curves as follows

$$\xi_w = \sqrt{\sum_{i=1}^{tns} \frac{(w_i^{\vartheta} - w_i^{ref})^2}{(w_i^{ref})^2}}, \tag{8.85}$$

where subscript i denotes the loading step and tns denotes the total number of the loading steps. The performance comparison is summarized in Tab. 8.12.

In the footing problem test, all integration schemes provide good results whose errors were acceptable in common engineering practice. Differences in the computational times were not so high because most of the nonlinear behaviour of the material was exhibited in the vicinity of the footing load area while the stress state changes in the remaining part of domain were moderate and therefore minimum number of substeps was required in RKF method used. The comparison of attained errors implies that the both low order schemes RKF-32 and RKF-32bs provide good results with reasonable numbers of model evaluation and computational times.

ϑ	Error measure type	RKF-32	RKF-32bs	RKF-43	RKF-54
10^{-7}	ξ_w	$3.1161{\cdot}10^{-4}$	$1.1464{\cdot}10^{-4}$	$1.1510{\cdot}10^{-4}$	0.0
	n_{ev}	232,923	308,760	385,350	465,510
	t_c [s]	894	888	940	1063
10^{-6}	ξ_w	$2.3368{\cdot}10^{-4}$	$1.1230{\cdot}10^{-4}$	$1.1486{\cdot}10^{-4}$	$4.5370{\cdot}10^{-5}$
	n_{ev}	231,756	307,737	384,435	461,418
	t_c [s]	753	829	971	1085
10^{-5}	ξ_w	$1.5320{\cdot}10^{-4}$	$1.0164{\cdot}10^{-4}$	$1.1370{\cdot}10^{-4}$	$1.4320{\cdot}10^{-5}$
	n_{ev}	231,531	307,182	383,855	461,874
	t_c [s]	1024	1177	1289	1481
10^{-4}	ξ_w	$4.5732{\cdot}10^{-4}$	$6.2021{\cdot}10^{-5}$	$1.0920{\cdot}10^{-4}$	$1.1393{\cdot}10^{-4}$
	n_{ev}	231,639	307,029	383,765	459,564
	t_c [s]	989	1152	1300	1492
10^{-3}	ξ_w	$2.3570{\cdot}10^{-4}$	$4.4557{\cdot}10^{-4}$	$6.9615{\cdot}10^{-4}$	$4.9381{\cdot}10^{-4}$
	n_{ev}	230,268	305,898	819,140	393,186
	t_c [s]	972	1094	1721	1411

Table 8.12: Performance comparison of RKF schemes for strip footing problem.

8.5 Simulation of excavation contruction phases

The problem of construction phases is often solved in the geotechnical field. It may take the form of building an embankment or, conversely, trench excavation. In the case of an embankment or dam, the simulation of the gradual construction process is similar to the one of the bridge building, and the algorithm in Table 7.1 for the determination of the initial displacements can be used.

The different case represents the excavation problem where the groups of elements are removed to simulate excavation. The initial displacements need not be determined, but the problem of forces due to removed elements arises on the interface between the removed part and the current one. These forces originate from the internal forces at interface nodes, which were in equilibrium before element removal. After the element block removal, there are missing contributions due to removed elements if the internal force vector is calculated with the original displacement vector, see Figure 8.8.

Let the group of elements Ω_r be removed at time $t + \Delta t$. The vector of forces due to element removal can be simply defined as

$$\boldsymbol{f}_r = \boldsymbol{f}_{ir}\left(\boldsymbol{d}(t)\right) - \boldsymbol{f}_{er}(t), \tag{8.86}$$

where $\boldsymbol{f}_{er}(t)$ is the vector of prescribed forces on the domain Ω_r assembled in the time step before element removal and internal force vector, $\boldsymbol{f}_{ir}$, is given on the removed domain, Ω_r, as

$$\boldsymbol{f}_{ir} = \int_{\Omega_r} \boldsymbol{B}^{\mathrm{T}} \boldsymbol{\sigma}\left(\boldsymbol{d}(t)\right) \mathrm{dV}. \tag{8.87}$$

The force vector, $\boldsymbol{f}_r$, may have significant magnitude, and it can cause the failure of the iteration process in the global Newton-Raphson procedure. This problem is more pronounced in the highly nonlinear behaviour of the given material, such as soil. Usually, tension states are not allowed in the soil models, and thus the model requires slow evolution of load increments in order to avoid the tensional stress states.

In a typical Newton-Raphson load step, the magnitude of the prescribed load is dependent on the actual time, and if the iteration process fails, the time increment is reduced, which leads to a decrease in load magnitude. However, this strategy cannot be used in the case of element removal because the element life function, $l_e(t)$, has a stepwise character, which results in the sudden application of the force $\boldsymbol{f}_r$ that any time step reduction cannot mitigate.

Theoretically, the magnitude of the forces due to element removal can be controlled by the size of the removed domain, Ω_r, but it is impractical and may not help in certain cases. In the common cases, the vector $\boldsymbol{f}_r$ is considered to be time-dependent virtually, and it is added to the right-hand side with the decreasing magnitude in the specific period (time steps) after the element removal. The magnitude ranges from the initial value of 1 to the zero value, and its value is decreased in the given range of time steps. Hence, the residual force vector in the global Newton-Raphson iteration contains contributions due to $\boldsymbol{f}_r$ and if the iteration fails, the time increment is reduced, which results in a decrease of the $\boldsymbol{f}_r$ magnitude.

A modified approach was implemented in SIFEL where an independent new Newton-Raphson procedure is spawned in the time step of element removal. At the new procedure, the constant right-hand side vector $\boldsymbol{f}_c$ represents the load vector assembled at the time

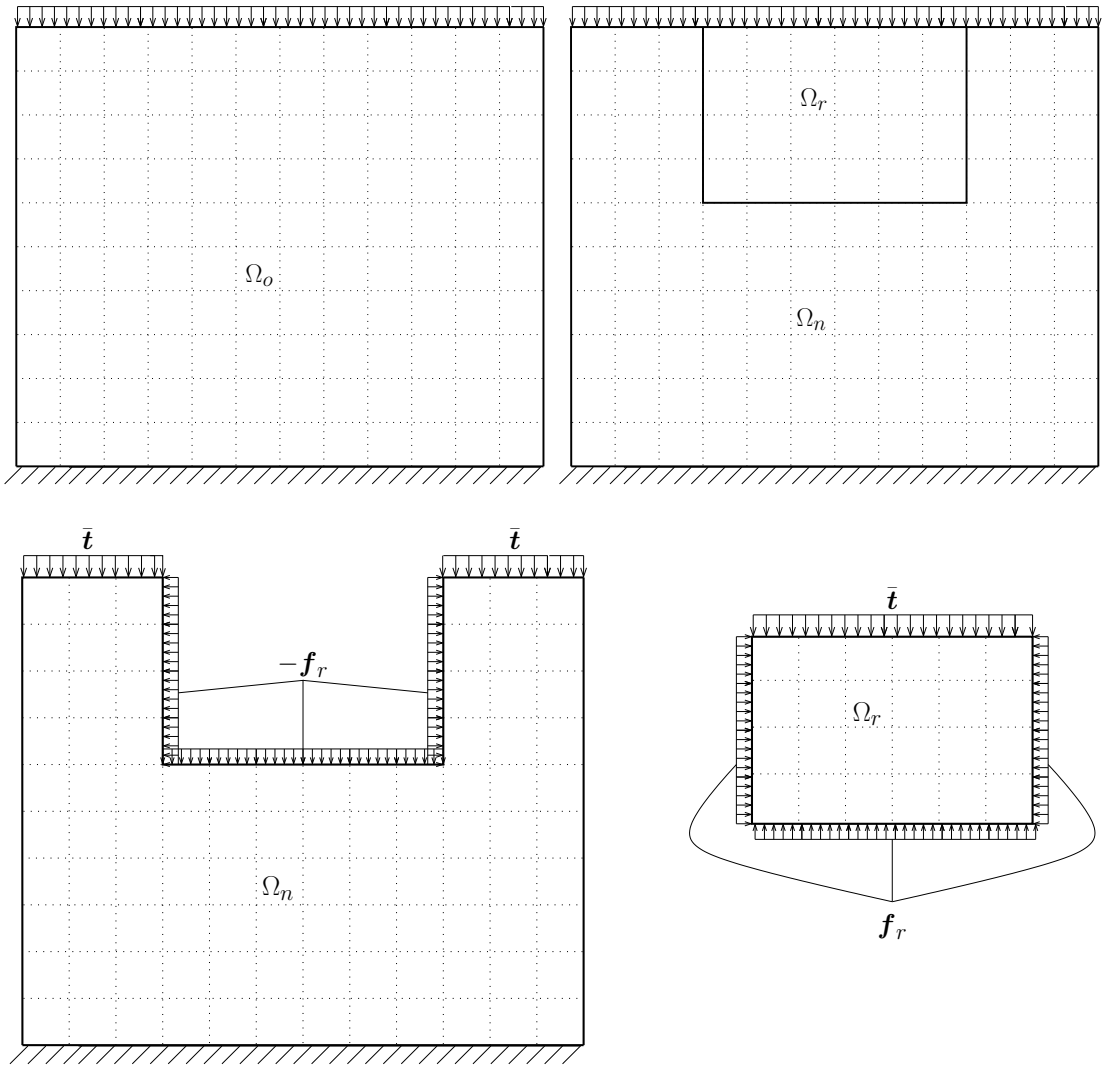


Figure 8.8: Original domain, Ω_o at time t (top left), new domain, Ω_n, and domain removed, Ω_r, at time $t + \Delta t$ (top right) and split domains with forces due to domain removed $\boldsymbol{f}_r$ (bottom).

t and the proportional part of the right-hand side, $\boldsymbol{f}_{pr} = -\boldsymbol{f}_r$. The whole algorithm is summarized in Table 8.13. In the Newton-Raphson procedure, the load coefficient, λ_N, is gradually increased from the zero value until the value one is attained, at which the process finishes. The size of λ_N increments and the solution error tolerance are the required inputs of the procedure together with vectors $\boldsymbol{f}_c$ and $\boldsymbol{f}_{pr}$. At the end of the procedure, the displacement vector, $\boldsymbol{d}$, contains contribution due to element removal, $\Delta\boldsymbol{d}$, and computation proceeds in the main time-stepping loop.

This approach allows for better control of the iteration of the load vector due to element removal, which can be set independently of the global Newton-Raphson procedure. Moreover, if the iteration fails, the user can be clearly informed that the failure was caused by element removal and not by some other load.

1	Identify set of elements on new domain configuration, Ω_n $\Omega_n = \{e_i \in \Omega \mid l_i(t + \Delta t) > 0\}$.
2	Identify set of removed elements, Ω_r $\Omega_r = \{e_i \in \Omega \mid l_i(t + \Delta t) = 0 \ \wedge \ l_i(t) > 0\}$.
3	Generate new code number for nodes on the domain Ω_n, all vectors of the global problem $\boldsymbol{f}_r, \quad \boldsymbol{f}_{ir}, \quad \boldsymbol{f}_{er}, \quad \boldsymbol{f}_c, \quad \boldsymbol{f}_{pr} \in \mathbb{R}^{nd}$, where nd is the maximum DOF number on the domain Ω_n.
4	Assemble the load vector, $\boldsymbol{f}_l$, on domain Ω_n at time t, load vector, $\boldsymbol{f}_{er}$, on domain Ω_r and vector of internal forces due to removed elements, $\boldsymbol{f}_{ir}$, on domain Ω_r.
5	Assemble the vector due to removed elements, $\boldsymbol{f}_r$, vector of constant load, $\boldsymbol{f}_c = \boldsymbol{f}_l - \boldsymbol{f}_r$, and proportional load vector, $\boldsymbol{f}_{pr} = \boldsymbol{f}_r$.
6	Solve the problem $\boldsymbol{K}\Delta\boldsymbol{d} = \boldsymbol{f}_{ext} - \boldsymbol{f}_{int} = \boldsymbol{f}_c + \lambda_N \boldsymbol{f}_{pr} - \boldsymbol{f}_{int}$ Newton-Raphson method on the domain Ω_n for $\lambda_N \in [0; 1]$.
7	Add displacement increment $\Delta\boldsymbol{d}$ to the attained displacement vector $\boldsymbol{d}$.

Table 8.13: Algorithm of handling with the vector of forces due to element removal.

8.6 Excavtion problem with hypoplastic model for unsaturated expansive soils

The algorithm proposed in Section 8.5 was tested on the excavation problem where the soil behaviour was described with the implemented hypoplastic model. There is a block of soil with dimensions 75 × 40 m where a trench is excavated with a side slope 10:17 and 10 m depth. The pit is created in 10 phases, where each stage consists of removing the soil layer with a thick 1 m. Nodes at the block bottom were fixed completely, while the nodes on the left and right sides were supported in the horizontal direction only. The FE mesh of the problem is depicted in Figure 8.9, where different colours distinguish particular soil layers for removal. A dead weight load was assumed in the problem by the value $\gamma_s = 20 \quad \mathrm{kNm}^{-3}$. The list of used material parameters is given in Table 8.14. The fully saturated state was considered in the problem, i.e. $s = 0$. After each stage,

φ_c	N	λ^*	κ^*	n_s	l_s	e_r^m
25°	1.56	0.08	0.01	0.0	0.0	0.0

s_r	r	m	κ_m	s_{e0}	e_0^M	a_e
-140 kPa	0.4	10.0	0.0	-200.0 kPa	0.5	0.5

Table 8.14: Set of model parameters used in the excavation problem.

the force vector due to removed elements was decreased gradually according to algorithm in Table 8.13. The increment of the load coefficient λ_N was considered to be 0.1 and

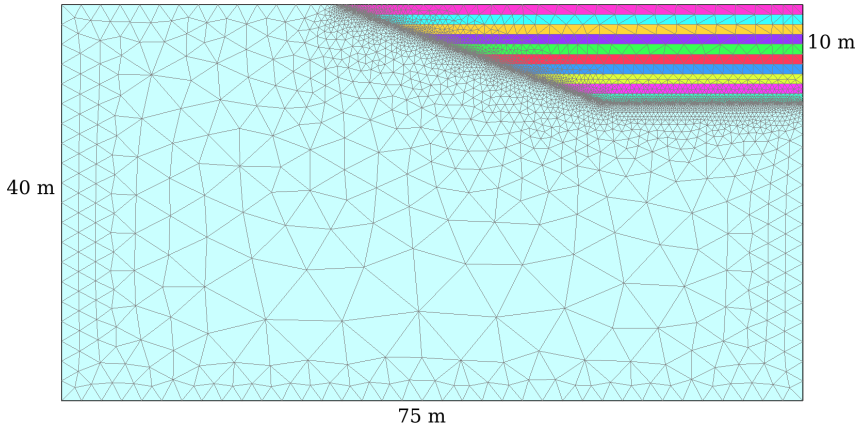
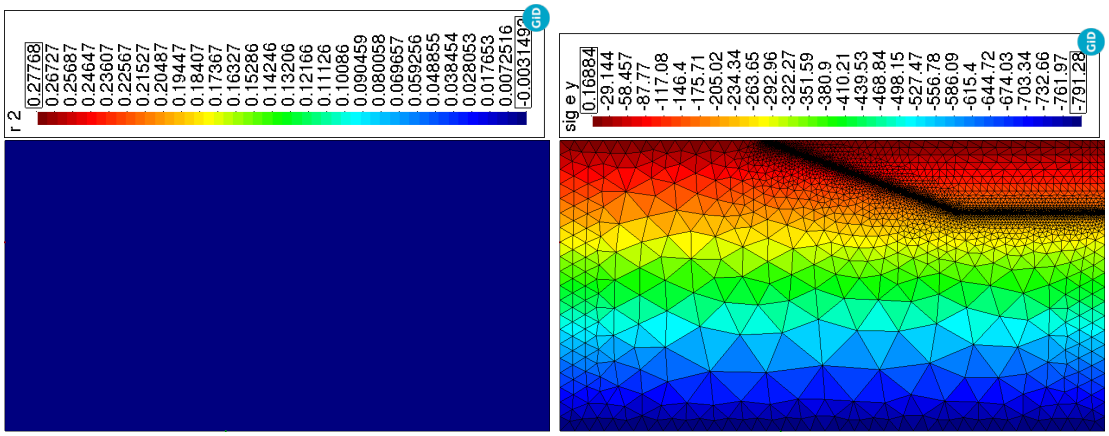


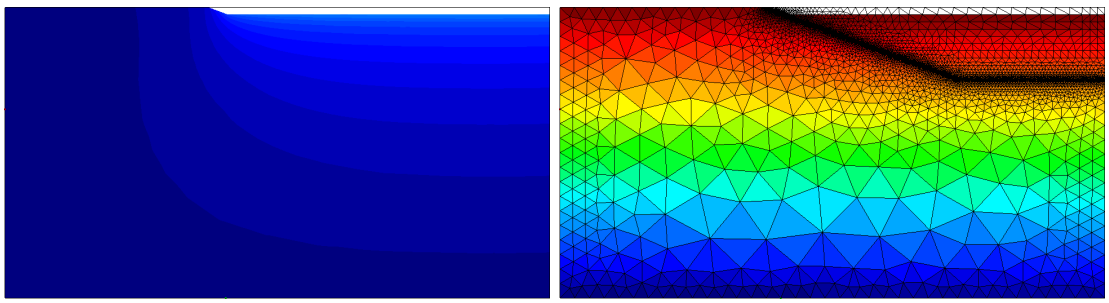
Figure 8.9: FE mesh of the excavation problem at the initial stage.

the solution error tolerance was set to 0.001. Note that the high nonlinearity of the hypoplastic model caused that the problem cannot be solved without proper treatment of the forces due to element removal.

Results from the analysis are captured in Figures 8.10–8.13.

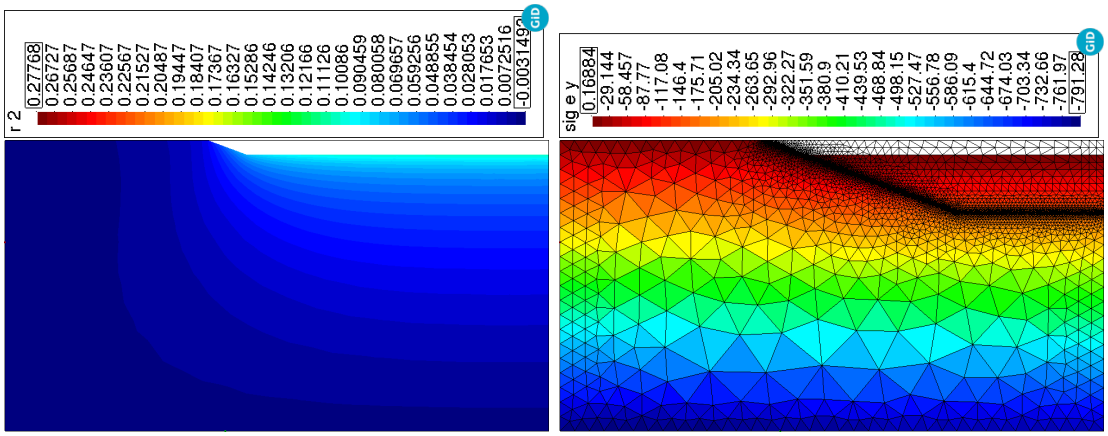


(a) Initial stage.

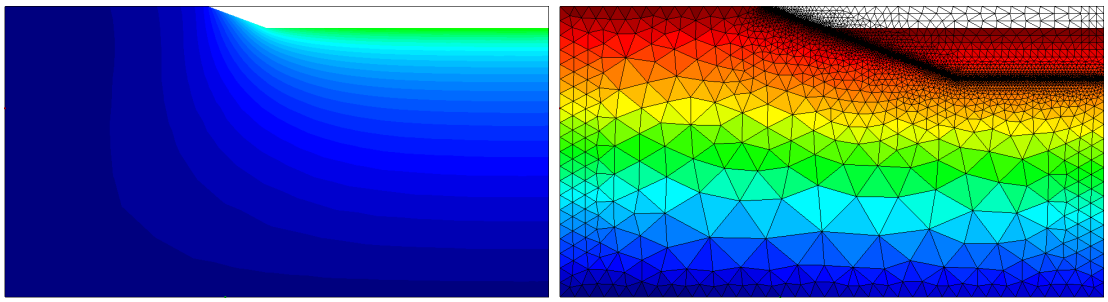


(b) Removal of soil layer 1.

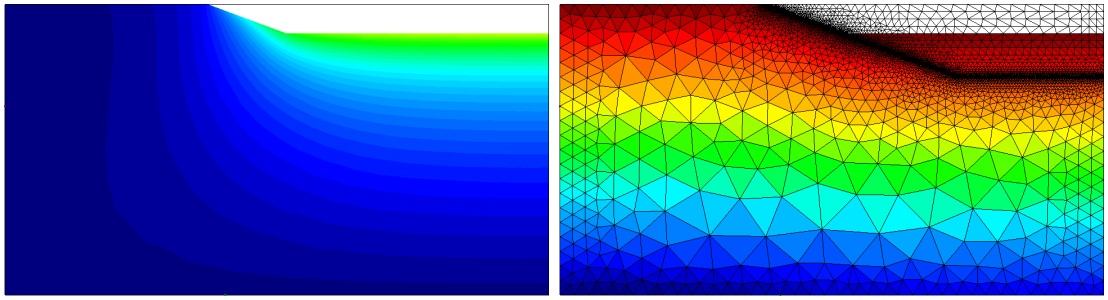
Figure 8.10: Evolution of the vertical displacement component [m] (left) and vertical stress component, σ_y, [kPa] (right) at initial stage and after removal of layer 1.



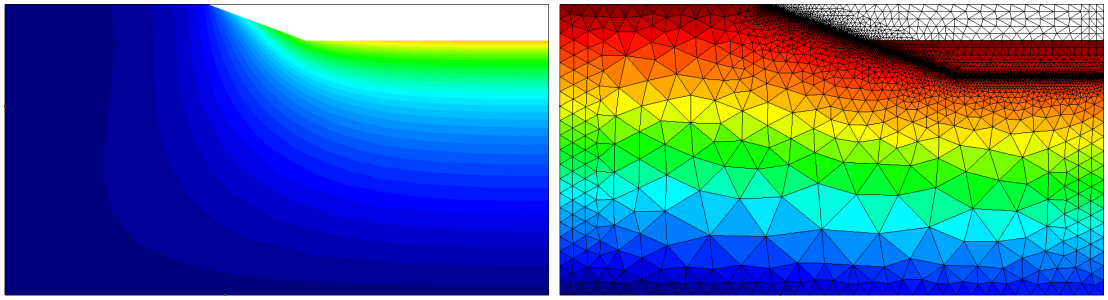
(a) Stage after the removal of soil layer 2.



(b) Stage after the removal of soil layer 3.

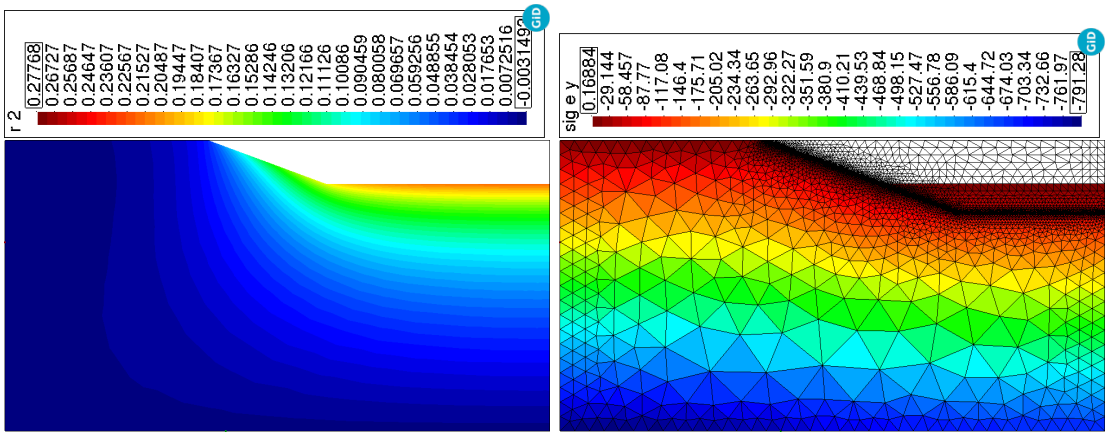


(c) Stage after the removal of soil layer 4.

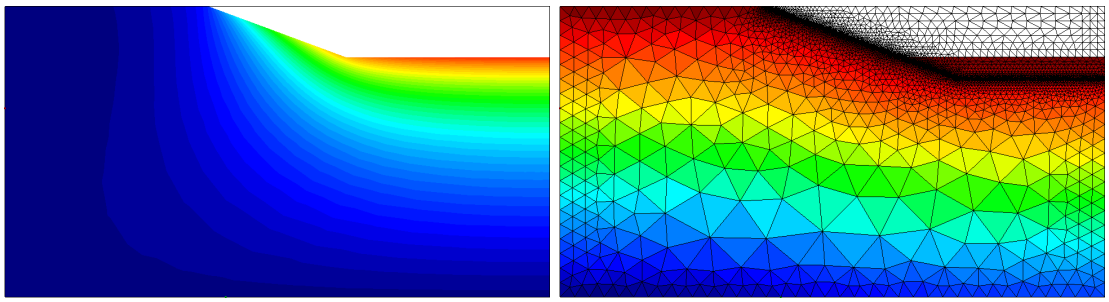


(d) Removal of soil layer 5 - vertical displacement [m] (left) and stress σ_y [kPa] (right).

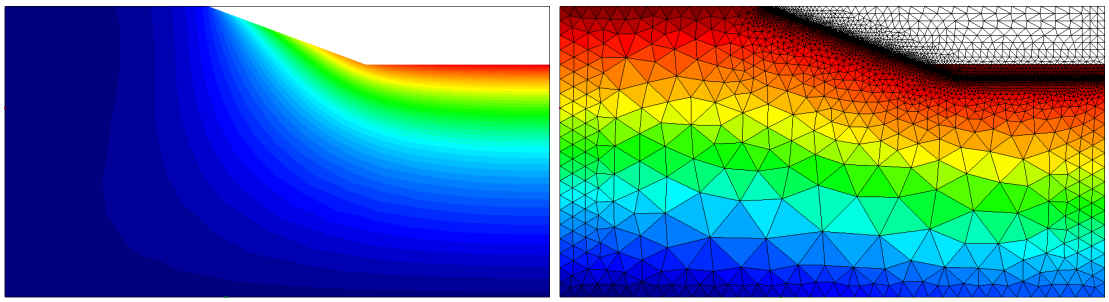
Figure 8.11: Evolution of vertical displacement component [m] (left) and vertical stress component, σ_y, [kPa] (right) at removal stages 2–5.



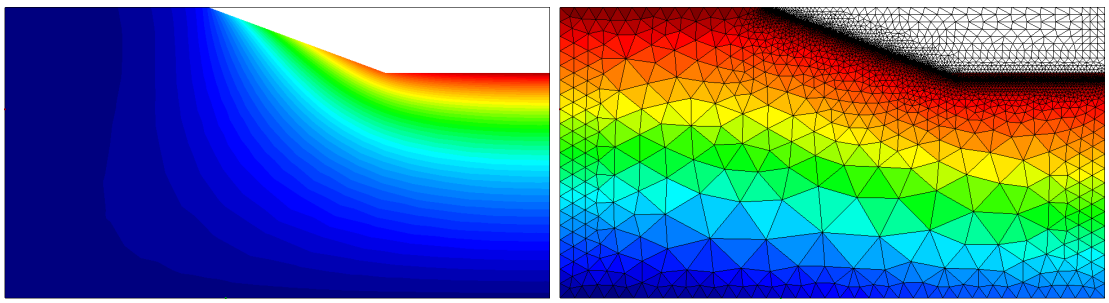
(a) Stage after the removal of soil layer 6.



(b) Stage after the removal of soil layer 7.

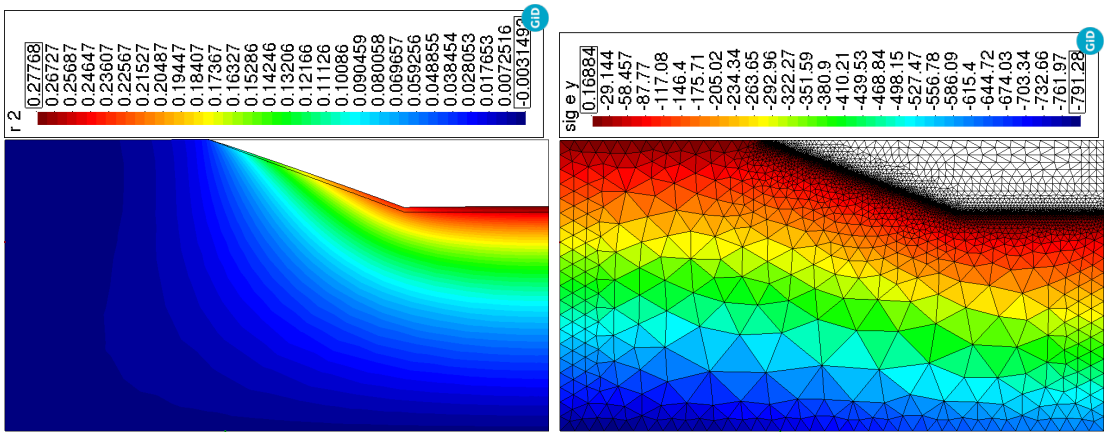


(c) Stage after the removal of soil layer 8.



(d) Removal of soil layer 9 - vertical displacement [m] (left) and stress σ_y [kPa] (right).

Figure 8.12: Evolution of vertical displacement component [m] (left) and vertical stress component, σ_y, [kPa] (right) at removal stages 6–9.



(a) Removal of soil layer 10 - vertical displacement [m] (left) and stress σ_y [kPa] (right).

Figure 8.13: Evolution of vertical displacement (left) and vertical stress component, σ_y, (right) at final stages after the removal of soil layer 10.

The vertical displacement evolution shows gradual elevation of the pit bottom with a maximum value of 277.6 mm. Figure 8.13 captures the final stage, where the vertical displacement field is depicted on the deformed shape of the domain together with the outline of the original mesh for a better comparison.

8.7 Conclusion

Hypoplastic model for unsaturated expansive soils was implemented in open source FE code SIFEL. The model was defined in the rate form and therefore integration in time had to be provided. Several RKF schemes were implemented and their performance was compared in the time integration of the model. All implemented schemes provided good results form the viewpoint of the attained error in tests with constant suction level and thus the number of model evaluation or elapsed computational time should be decisive for the scheme selection. In this case, the RKF-32bs or RKF-32 schemes performed better than the high order schemes.

Test with variable suction path revealed numerical difficulties connected with the definition of hydraulic part of the model which is very sensitive to the a_{sc} state variable. Because of piecewise linear nature of the a_{sc} variable which is not smooth function, the RKF methods failed to integrate the model response. Two approaches were proposed for the improvement of the model integration, the first one consist in approximation of the a_{sc} definition with logarithmic-exponential function and the second approach introduces selective integration of a_{sc} state variable with simple forward Euler scheme. The approximation suffers by the limitation of computer arithmetic of the standard `double` type, because of too high value of exponential function argument and therefore selective integration was used in the tests with variable suction path. Performed test showed that high order method RKF-54 did not provide reliable results of stresses/strains for large pseudo-time step length and low values of tolerance ϑ and similarly RKF32 method did not provide good results of the degree of saturation for large pseudo-time steps and low

values of tolerance ϑ. For shorter pseudo-time steps and low tolerance values, problems occurred with the degree of saturation values in high order RKF-54 scheme.

It can be concluded that integration scheme based on RKF-32bs or RKF-32 are suitable for the considered hydro-mechanical hypoplastic model, where RKF-32bs scheme proposed by Bogacki-Shampine performed better than standard RKF-32 in most of cases and can be considered as a good alternative integration scheme for the proposed hydro-mechanical model.

Simulation of removing structural parts in the FE computation requires special treatment in the case of nonlinear material models. The forces due to element removal may cause computation failure because they may have a significant magnitude, and they can lead to convergence problems. The algorithm of gradually diminishing these forces was successfully tested on the excavation problem with a highly nonlinear material model. The problem could not be solved reasonably without that algorithm, and the simulation proved the algorithm's correct performance.

Chapter 9

Conclusions

Experiences with developing the large engineering open source software SIFEL resulted in the code based on C++ language. However, concepts of OOP were used cautiously concerning the clarity of the code for new users and portability to the various platforms and compilers. The actual implementation is certainly one of many possibilities, but it is robust, intelligible and extensible. The code was proven on many real-world engineering problems, and authors believe that it will be able to cover more complicated problems in the future and remain exploitable for starting students or co-workers. There were also showed some alternative approaches that exploit new features of C++ established at the C++11 standard. They were designed to preserve the high performance of the procedural style design while improving the source code's maintenance level close to the OOP style with polymorphism and inheritance.

Similar views were expressed in the interview with Alexander Stepanov [Stepanov and Russo, 2008], the author of STL, whose quotation follows:
"I find OOP methodologically wrong. It starts with classes. It is as if mathematicians would start with axioms. You do not start with axioms - you start with proofs. Only when you have found a bunch of related proofs, can you come up with axioms. You end with axioms. The same thing is true in programming: you have to start with interesting algorithms. Only when you understand them well, can you come up with an interface that will let them work."

Since its beginning in 2001, the software has been successfully used many times for solving real-world engineering problems. Among the most demanding problems belong simulation of the gradual construction processes, where the structure exhibits nonlinear behaviour. It requires a special treatment and algorithm to properly determine the initial displacements, strains and stresses.

The problem of the casting procedure of a thick concrete foundation slab under groundwater level represented the first attempt to handle gradual construction phases in the simulation. The objective of the analysis was to determine crack propagation in the period of the first 30 days from the beginning of casting. The slab behaviour was considered a complex coupled thermo-hydro-mechanical problem, where several phenomena were considered, such as hydration heat generation, climatic conditions, shrinkage, drying, creep and damage. The concept of element life functions was established in the code, which provided some performance benefits. Because the crack pattern and stress distribution were the point of interest, determining initial displacements was not needed in this case.

Creep analysis of the bridge constructed with the help of the cantilever method was another case, where the proper simulation of the gradual construction played a significant role. In this case, the code was extended by automatically detecting newly added parts and determining their initial displacement. The real construction procedure contained elevation of the bridge deck level either by the change of pre-tensioning or tilting of the form traveller. Therefore the code was extended by a special procedure for the tilting of the newly added. The implemented procedures were tested on the bridge in Mělník, where the parameters of the creep model (B3) were determined by the optimization techniques. Because of the lack of complete data on construction phases, the model parameters were determined to fit the deflection increments after the final stage of the bridge construction.

The last code extension dealing with the construction phases was implemented in connection with a highly nonlinear hypoplastic model for expansive clays. The model was formulated in the rate form, and thus the integration with the help of the Runge-Kutta-Fehlberg's methods had to be proposed. Tests revealed convergence problems due to the non-smooth formulation of the hydro-mechanical coupling. The retention curve was therefore smoothed, which improved the model integration performance. Alternatively, the selective integration procedure could be involved in the vicinity of the problematic points of the hydraulic part of the model. Concerning the area of the model application, it was necessary to tackle the problem of the gradual removal of structural parts, which can be observed in trench excavation problems. Removal of structural parts results in the sudden evolution of forces due to removed elements that may cause convergence problems, especially for highly nonlinear material models. Therefore the code was extended by the procedure, where the vector of forces due to removed elements is gradually applied within the scope of the individual Newton-Raphson procedure. The functionality of the implemented solution was successfully tested on the example of trench excavation.

It can be concluded that the implemented code extensions allow for advanced simulation of construction phases that can be faced in engineering practice. Most of them were tested successfully on complex real-world problems hardly solvable in commercial software, but the availability of input data for the proper problem setting is still challenging.

Appendix A

Source codes for benchmarks

A.1 General classes for vector/matrix algebra

```
1   class ivector{
2    public :
3     int n, capacity;
4     bool dealloc;
5     int *a;
6     ivector() : n{0}, a{nullptr}, dealloc{false}, capacity{0} {};
7     ivector(int in, int *ptr) : n{in}, a{ptr},
8       capacity{in}, dealloc{false}  {
9         memset(a, 0, sizeof(*a)*n);
10    };
11    ivector(int in) : n{in}, dealloc{true}, capacity{in}  {
12      a = new int[capacity];
13      memset(a, 0, sizeof(*a)*capacity);
14    }
15    ~ivector()  {if (dealloc)  delete [] a; };
16    int &operator ()(int i) const {return a[i];};
17    int &operator ()(int i) {return a[i];};
18    void realloc(int in) {
19      int newcap = in;
20      if (newcap > capacity){
21        capacity = newcap;
22        if (dealloc)  delete [] a;
23        n = capacity;  a = new int[capacity];  dealloc = true;
24        memset(a, 0, sizeof(*a)*capacity);
25      }
26      else {n = in;  memset(a, 0, sizeof(*a)*newcap); }
27    };
28    void realloc(int in, int *mem){
29      if (dealloc)  delete [] a;
30      n = in;  capacity = n;      a = mem;  dealloc = false;
31      memset(a, 0, sizeof(*a)*n);
32    };
33  };
```

Table A.1: Class for vector with integer components.

```
1   class matrix{
2    public :
3     int m, n, capacity;
4     bool dealloc;
5     double *a;
6     matrix() : m{0},n{0},capacity{0},dealloc{false},a{nullptr} {};
7     matrix(int im, int in, double *ptr) :
8       m{im}, n{in}, capacity{im*in}, dealloc{false}, a{ptr}
9   {  memset(a, 0, sizeof(*a)*m*n); };
10    matrix(int im, int in) :
11      m{im}, n{in}, dealloc{true}, capacity{im*in}  {
12          a = new double[capacity];
13          memset(a, 0, sizeof(*a)*capacity); };
14    ~matrix()  {if (dealloc)  delete [] a; };
15    double &operator ()(int i, int j) const {return a[i*n+j];};
16    double &operator ()(int i, int j) {return a[i*n+j];};
17    void realloc(int im, int in){
18      const int newcap = im*in;
19      if (newcap > capacity){
20        capacity = newcap;
21        if (dealloc)   delete [] a;
22        a = new double[capacity];  dealloc = true;
23        memset(a, 0, sizeof(*a)*capacity);
24        m = im;   n = in;
25      }
26      else {m = im;  n = in;  memset(a, 0, sizeof(*a)*newcap);}
27    };
28    void realloc(int im, int in, double *mem){
29      if (dealloc)  delete [] a;
30      m = im;   n = in;  capacity = m*n;  dealloc = false;
31      a = mem;  memset(a, 0, sizeof(*a)*capacity);
32    };
33    void zero(){memset(a, 0, sizeof(*a)*m*n);};
34    void addm(matrix &b){
35      assert((m == b.m) && (n == b.n));
36      const int ncomp = m*n;
37      for(int i=0; i<ncomp; i++)  a[i] += b.a[i];
38    }
39    void scale(double c){
40      const int ncomp = m*n;
41      for (int i=0; i<ncomp; i++) a[i] *= c; };
42    void localize(const matrix &lm, const ivector &dofnum){
43      int idof, jdof;
44        assert((lm.m == dofnum.n) && (lm.n == dofnum.n));
45      int n = lm.m;
46      for(int i=0; i<n; i++){
47        idof = dofnum(i);
48        if (idof > 0){
49          for(int j=0; j<n; j++){
50            jdof = dofnum(j);
51            if (jdof > 0) this->operator()(idof-1, jdof-1)+=lm(i,j);
52          } } } } };
```

Table A.2: Class for matrix with the real components.

```
1   class vector{
2    public :
3     int n, capacity;
4     bool dealloc;
5     double *a;
6     vector() : n{0}, capacity{0}, dealloc{false}, a{nullptr} {};
7     vector(int in, double *ptr) :
8       n{in}, capacity{in}, dealloc{false}, a{ptr} {
9             memset(a, 0, sizeof(*a)*n);
10    };
11    vector(int in) : n{in}, capacity{in}, dealloc{true}{
12      a = new double[capacity];
13      memset(a, 0, sizeof(*a)*capacity);
14    };
15    ~vector()  {if (dealloc)  delete [] a; };
16    double &operator ()(int i) {return a[i];};
17    double &operator ()(int i) const {return a[i];};
18    void realloc(int in){
19          int newcap = in;
20      if (newcap > capacity){
21        capacity = newcap;
22        if (dealloc)      delete [] a;
23        n = capacity;  a = new double[capacity];  dealloc = true;
24        memset(a, 0, sizeof(*a)*capacity);
25          }
26      else{
27        n = in;
28        memset(a, 0, sizeof(*a)*newcap);
29      }
30    };
31    void realloc(int in, double *mem){
32      if (dealloc)  delete [] a;
33      n = in;   capacity = n;  a = mem;  dealloc = false;
34      memset(a, 0, sizeof(*a)*n);
35    }
36    void scale(double c){
37      for (int i=0; i<n; i++)  a[i] *= c;
38    };
39    void zero(){memset(a, 0, sizeof(*a)*n);};
40    void addv(vector &v){
41      for (int i=0; i<n; i++)  a[i] += v(i);
42    };
43    void fill(double c){
44      for(int i=0; i<n; i++)  a[i] = c;
45    };
46    void localize(const vector &lv, const ivector &dofnum){
47      assert(lv.n == dofnum.n);
48      int idof, n = lv.n;
49      for(int i=0; i<n; i++)
50        {  idof = dofnum(i);  if (idof > 0)  a[idof-1] += lv(i);  }
51    };
52  };
```

Table A.3: Class for vector with the real components.

A.2 Procedures for the assembling of element stiffness matrix and internal force vector

```
1   void stiff_matrix(matrix &sm, int eid,
2                     const vector &x, const vector &y){
3     matrix d(ncomp, ncomp);
4     matrix b(ncomp,ndofe), aux(ndofe, ncomp), lsm(ndofe, ndofe);
5     double jac;
6     // natural coordinates of integration points
7     double xi[nip]  = {-0.57735, -0.57735,  0.57735, 0.57735};
8     double eta[nip] = {-0.57735,  0.57735, -0.57735, 0.57735};
9     // ip weights
10    double w[nip] = {1.0, 1.0, 1.0, 1.0};
11    //  stiffness matrix of material mm->matstiff(d, eid, mt);
12    double e = 2.0e9, nu=0.25;
13    double c = e/(1.0+nu)/(1.0-2.0*nu);
14    // thickness mc->give_area(eid, a);
15    double thick = 0.1;
16    for(int i = 0; i<nip; i++){ // loop over integration points
17      jac = geom_matrix(b, x, y, xi[i], eta[i]);
18      d(0,0) = c*(1.0-nu); d(0,1) = c*nu;       d(0,2) = 0.0;
19      d(1,0) = c*nu;       d(1,1) = c*(1.0-nu); d(1,2) = 0.0;
20      d(2,0) = 0.0;        d(2,1) = 0.0;  d(2,2) = e/2.0/(1.0+nu);
21      // sm =  B^T. D. B
22      mtxm(b, d, aux);
23      mxm(aux, b, lsm);
24      lsm.scale(jac*thick*w[i]); // ip weight
25      sm.addm(lsm);
26    }
27  };
```

Table A.4: Procedure for the assembling of the stiffness matrix for the linear quadrilateral element.

```
1   void int_forces(vector &ifor, int eid,
2                   const vector &x, const vector &y){
3     vector sig(ncomp), eps(ncomp), contr(ndofe);
4     matrix d(ncomp, ncomp), b(ncomp,ndofe);
5     double jac;
6     // natural coordinates of integration points
7     double xi[nip]  = {-0.57735, -0.57735,  0.57735, 0.57735};
8     double eta[nip] = {-0.57735,  0.57735, -0.57735, 0.57735};
9     // ip wieghts
10    double w[nip] = {1.0, 1.0, 1.0, 1.0};
11    //  stiffness matrix of material mm->matstiff(d, eid, mt);
12    double e = 2.0e9, nu=0.25;
13    double c = e/(1.0+nu)/(1.0-2.0*nu);
14    // thickness mc->give_area(eid, a);
15    double thick = 0.1;
16    for(int i = 0; i<ncomp; i++)   // simulation of D.eps
17      mxv(d, eps, sig);
18    for(int i = 0; i<nip; i++){ // loop over integration points
19      mtxv(b, sig, contr);     // B^T sig
20      ifor.scale(jac*thick*w[i]);        // ip weight
21      ifor.addv(contr);
22    }
23  };
```

Table A.5: Procedure for the assembling of the stiffness matrix for the linear quadrilateral element.

A.3 Procedural approach

```
1   const int ne = 50000;
2   Element *elems = new Element[ne];
3   elemtype *readarray = new elemtype[ne];
4   srand(1);
5   vector sky(51*1000*1000);
6   for(int i=0; i<ne; i++){
7     elems[i].et = elemtype(rand()%3+1);
8     int ndofe = get_ndofe(elems[i].et);
9     for (int j = 0, k=ndofe/2; j<ndofe/2; j++, k++){
10      elems[i].cn[j] = i*1000+j;
11      elems[i].cn[k] = i*1000+1000+j;
12    }
13  }
14  matrix sm;
15  vector x(4), y(4);
16  x(0) = 0.0;  y(0) = 0.0;  x(1) = 1.0;  y(1) = 0.0;
17  x(2) = 1.0;  y(2) = 1.0;  x(3) = 0.0;  y(3) = 1.0;
18  for (int j=0; j<100; j++){
19    for (int i=0; i<ne; i++){
20      int ndofe = get_ndofe(elems[i].et);
21      sm.realloc(ndofe, ndofe);
22      stiff_matrix(elems[i].et, sm, i, x, y);
23      ivector edof;
24      edof.realloc(ndofe, elems[i].cn);
25      for (int k=0; k<ndofe; k++){
26        vector tmp;
27        tmp.realloc(ndofe, &sm(k,0));
28        sky.localize(tmp, edof);
29      }
30    }
31    vector ifor;
32    for (int i=0; i<ne; i++){
33      int ndofe = get_ndofe(elems[i].et);
34      ifor.realloc(ndofe);
35      int_forces(elems[i].et, ifor, i, x, y);
36      ivector edof;
37      edof.realloc(ndofe, elems[i].cn);
38      sky.localize(ifor, edof);
39    }
40  }
```

Table A.6: Procedure `main` of the procedural approach.

A.4 OOP inheritance approach

```
 1   const int ne = 50000;
 2   Element **elems = new Element*[ne];
 3   elemtype *readarray = new elemtype[ne];
 4   srand(1);
 5   vector sky(51*1000*1000);
 6   for(int i=0; i<ne; i++){
 7     readarray[i] = elemtype(rand()%3+1);
 8   }
 9   for (int i=0; i<ne; i++){
10     if (readarray[i] == linbar)   elems[i] = new LinBarElem;
11     if (readarray[i] == lintr)    elems[i] = new LinTrElem;
12     if (readarray[i] == linquad)  elems[i] = new LinQuadElem;
13     int ndofe = elems[i]->get_ndofe();
14     for (int j = 0, k=ndofe/2; j<ndofe/2; j++, k++){
15       elems[i]->cn[j] = i*1000+j;
16       elems[i]->cn[k] = i*1000+1000+j;
17     }
18   }
19   matrix sm;
20   vector x(4), y(4), ifor;
21   x(0) = 0.0;  y(0) = 0.0;  x(1) = 1.0;  y(1) = 0.0;
22   x(2) = 1.0;  y(2) = 1.0;  x(3) = 0.0;  y(3) = 1.0;
23   for (int j=0; j<100; j++){
24     for (int i=0; i<ne; i++){
25       int ndofe = elems[i]->get_ndofe();
26       sm.realloc(ndofe, ndofe);
27       elems[i]->stiff_matrix(sm, i, x, y);
28       ivector edof;
29       edof.realloc(ndofe, elems[i]->cn);
30       for (int k=0; k<ndofe; k++){
31         vector tmp;
32         tmp.realloc(ndofe, &sm(k,0));
33         sky.localize(tmp, edof);
34       }
35     }
36     for (int i=0; i<ne; i++){
37       int ndofe = elems[i]->get_ndofe();
38        ifor.realloc(ndofe);
39       elems[i]->int_forces(ifor, i, x, y);
40       ivector edof;
41       edof.realloc(ndofe, elems[i]->cn);
42        sky.localize(ifor, edof);
43     }
44   }
```

Table A.7: Procedure `main` of the OOP inheritance approach.

A.5 Interface approach - direct

```
1   const int ne = 50000;
2   Element *elems = new Element[ne];
3   elemtype *readarray = new elemtype[ne];
4   srand(1);
5   int i, j;
6   vector sky(51*1000*1000);
7   for(i=0; i<ne; i++)     readarray[i] = elemtype(rand()%3+1);
8   for (i=0; i<ne; i++){
9     std::size_t j = std::find(eldefifcarray.begin(),
10                              eldefifcarray.end(),
11                              readarray[i]) - eldefifcarray.begin();
12    assert(j<nelemt);
13    elems[i].init(readarray[i], &eldefifcarray[j]);
14    int ndofe = elems[i].defifc->get_ndofe();
15    for (int j = 0, k=ndofe/2; j<ndofe/2; j++, k++){
16      elems[i].cn[j] = i*1000+j;
17      elems[i].cn[k] = i*1000+1000+j;
18    }
19  }
20  matrix sm;
21  vector x(4), y(4), ifor;
22  x(0) = 0.0;  y(0) = 0.0;  x(1) = 1.0;  y(1) = 0.0;
23  x(2) = 1.0;  y(2) = 1.0;  x(3) = 0.0;  y(3) = 1.0;
24  for (int j=0; j<100; j++){
25    for (int i=0; i<ne; i++){
26      int ndofe = elems[i].defifc->get_ndofe();
27      sm.realloc(ndofe, ndofe);
28      elems[i].defifc->stiff_matrix(sm, i, x, y);
29      ivector edof;
30      edof.realloc(ndofe, elems[i].cn);
31      for (int k=0; k<ndofe; k++){
32        vector tmp;
33        tmp.realloc(ndofe, &sm(k,0));
34        sky.localize(tmp, edof);
35      }
36    }
37    for (int i=0; i<ne; i++){
38      int ndofe = elems[i].defifc->get_ndofe();
39      ifor.realloc(ndofe);
40      elems[i].defifc->int_forces(ifor, i, x, y);
41      ivector edof;
42      edof.realloc(ndofe, elems[i].cn);
43      sky.localize(ifor, edof);
44    }
45  }
```

Table A.8: Procedure `main` of the interface approach – direct.

A.6 Interface approach - `std::function`

```
1   const int ne = 50000;
2   Element *elems = new Element[ne];
3   elemtype *readarray = new elemtype[ne];
4   srand(1);
5   vector sky(51*1000*1000);
6   for(int i=0; i<ne; i++){
7     readarray[i] = elemtype(rand()%3+1);
8     if (readarray[i] == linbar)
9       std::get<LinBarElem>(elemobjs).ndofe = 4;
10  }
11  for(int i=0; i<ne; i++){
12    std::size_t j = std::find(eldefifcarray.begin(),
13                              eldefifcarray.end(),
14                              readarray[i]) - eldefifcarray.begin();
15    assert(j<nelemt);
16    elems[i].init(readarray[i], &eldefifcarray[j]);
17    int ndofe = elems[i].defifc->get_ndofe();
18    for (int j = 0, k=ndofe/2; j<ndofe/2; j++){
19      elems[i].cn[j] = i*1000+j;
20      elems[i].cn[k] = i*1000+1000+j;
21    }
22  }
23  matrix sm;
24  vector x(4), y(4), ifor;
25  x(0) = 0.0;  y(0) = 0.0;  x(1) = 1.0;  y(1) = 0.0;
26  x(2) = 1.0;  y(2) = 1.0;  x(3) = 0.0;  y(3) = 1.0;
27  for (int j=0; j<100; j++){
28    for (int i=0; i<ne; i++){
29      int ndofe = elems[i].defifc->get_ndofe();
30      sm.realloc(ndofe, ndofe);
31      elems[i].defifc->stiff_matrix(sm, i, x, y);
32      ivector edof;
33      edof.realloc(ndofe, elems[i].cn);
34      for (int k=0; k<ndofe; k++){
35        vector tmp;
36        tmp.realloc(ndofe, &sm(k,0));
37        sky.localize(tmp, edof);
38      }
39    }
40    for (int i=0; i<ne; i++){
41      int ndofe = elems[i].defifc->get_ndofe();
42      ifor.realloc(ndofe);
43      elems[i].defifc->int_forces(ifor, i, x, y);
44      ivector edof;
45      edof.realloc(ndofe, elems[i].cn);
46      sky.localize(ifor, edof);
47    }
48  }
```

Table A.9: Procedure `main` of the interface approach – `std::function`.

Appendix B

Definition of macrostructure effective stresses for the hypoplastic model

This chapter summarises the definition of the selected parts of the hypoplastic model connected with the macrostructure effective stress. A full description of the model can be found in Mašín [2013] and Mašín [2017].

In the following text, the second-order tensors or matrices are denoted by bold italic font (e.g. $\boldsymbol{\sigma}$, $\boldsymbol{N}$) while the fourth-order tensors are written in capital calligraphic bold font (e.g. $\boldsymbol{\mathcal{M}}$, $\boldsymbol{\mathcal{I}}$). There are also used symbols "$\cdot$" and ":" between tensors of various orders for the single and double contraction, respectively, while the symbol "$\otimes$" indicates a dyadic product of two tensors. Additionally, $\boldsymbol{I}$ is the second-order identity tensor, $\|\boldsymbol{\sigma}\|$ denotes the Euclidean norm of tensor and symbol tr is defined as tensor trace $\mathrm{tr}(\boldsymbol{\sigma}) = \sigma_{ii}$. The symbol $\det(\boldsymbol{\sigma})$ represents the determinant of the second-order tensor, and the dot $\dot{()}$ denotes the time derivative.

Recall that the macrostructure effective stress rate is governed by the hypoplastic model given according to Equation (8.27) as

$$\dot{\boldsymbol{\sigma}}^M = f_s \left(\boldsymbol{\mathcal{L}} : \dot{\boldsymbol{\varepsilon}}^M + f_d \boldsymbol{N} \|\dot{\boldsymbol{\varepsilon}}^M\| \right) + f_u \boldsymbol{H}. \tag{B.1}$$

The symbol $\boldsymbol{\mathcal{L}}$ is the hypoelastic fourth-order tensor defined by

$$\boldsymbol{\mathcal{L}} = 3 \left(c_1 \boldsymbol{\mathcal{I}} + c_2 \, a^2 \, \hat{\boldsymbol{\sigma}}^M \otimes \hat{\boldsymbol{\sigma}}^M \right). \tag{B.2}$$

In Equation (B.1), $\hat{\boldsymbol{\sigma}}^M$ represents dimensionless stress tensor given by $\hat{\boldsymbol{\sigma}}^M = \dfrac{\boldsymbol{\sigma}^M}{\mathrm{tr}(\boldsymbol{\sigma}^M)}$, c_1, c_2 and a are scalar factors defined as:

$$c_1 = \frac{2\left(3 + a^2 - 2^\alpha \, a \sqrt{3}\right)}{9\, r}, \tag{B.3}$$

$$c_2 = 1 + (1 - c_1)\frac{3}{a^2}, \tag{B.4}$$

$$a = \frac{\sqrt{3}(3 - \sin\phi_c)}{2\sqrt{2}\sin\phi_c}, \tag{B.5}$$

where r is a material parameter and α is the function of material parameters ϕ_c, λ^* and κ^*

$$\alpha = \frac{1}{\ln 2} \ln \left[\frac{\lambda^* - \kappa^*}{\lambda^* + \kappa^*} \left(\frac{3 + a^2}{a\sqrt{3}} \right) \right] \tag{B.6}$$

The second-order tensor $\boldsymbol{N}$ can be defined according to failure condition in the form

$$\boldsymbol{N} = \boldsymbol{\mathcal{L}}\left(Y\frac{\hat{\boldsymbol{m}}}{\|\hat{\boldsymbol{m}}\|}\right), \tag{B.7}$$

where Y represents the failure criterion in the from

$$Y = \left(\frac{\sqrt{3}\,a}{3+a^2} - 1\right)\frac{(I_1 I_2 + 9I_3)\left(1-\sin^2\phi_c\right)}{8I_3\,\sin^2\phi_c} + \frac{\sqrt{3}\,a}{3+a^2}, \tag{B.8}$$

where I_1, I_2 and I_3 stands for the first, second and third stress invariants defined as follows

$$I_1 = \mathrm{tr}\left(\boldsymbol{\sigma}^M\right), \quad I_2 = \frac{1}{2}\left(\boldsymbol{\sigma}^M : \boldsymbol{\sigma}^M - \mathrm{tr}\left(\boldsymbol{\sigma}^M\right)\right), \quad I_3 = \det\left(\boldsymbol{\sigma}^M\right). \tag{B.9}$$

Direction of 'hypoplastic flow' is given by the second order tensor $\hat{\boldsymbol{m}}$

$$\hat{\boldsymbol{m}} = -\frac{a}{F}\left[\hat{\boldsymbol{\sigma}}^M + \mathrm{dev}\left(\hat{\boldsymbol{\sigma}}^M\right) - \frac{\hat{\boldsymbol{\sigma}}^M}{3}\left(\frac{6\,\hat{\boldsymbol{\sigma}}^M : \hat{\boldsymbol{\sigma}}^M - 1}{(F/a)^2 + \hat{\boldsymbol{\sigma}}^M : \hat{\boldsymbol{\sigma}}^M}\right)\right], \tag{B.10}$$

where $\mathrm{dev}\left(\hat{\boldsymbol{\sigma}}^{\mathrm{M}}\right) = \hat{\boldsymbol{\sigma}}^{\mathrm{M}} - \boldsymbol{I}\mathrm{tr}\left(\hat{\boldsymbol{\sigma}}^{\mathrm{M}}\right)/3$ and factor F defined by

$$F = \sqrt{\frac{1}{8}\tan^2\psi_h + \frac{2-\tan^2\psi_h}{2+\sqrt{2}\tan\psi_h\cos 3\theta} - \frac{1}{2\sqrt{2}}\tan\psi_h}, \tag{B.11}$$

where

$$\tan\psi_h = \sqrt{3}\|\mathrm{dev}(\hat{\boldsymbol{\sigma}}^M)\|, \quad \cos 3\theta = -\sqrt{6}\frac{\mathrm{tr}\left(\hat{\boldsymbol{\sigma}}^M\cdot\hat{\boldsymbol{\sigma}}^M\cdot\hat{\boldsymbol{\sigma}}^M\right)}{\left[\mathrm{dev}(\hat{\boldsymbol{\sigma}}^M) : \mathrm{dev}\left(\hat{\boldsymbol{\sigma}}^M\right)\right]^{3/2}}. \tag{B.12}$$

The barotropy factor f_s introduces the pressure dependency of the model response according to the mean stress level attained

$$f_s = \frac{3\,p^M}{\lambda^*(s)}\left(3 + a^2 - 2^\alpha\, a\sqrt{3}\right)^{-1}, \tag{B.13}$$

while the pyknotropy factor f_d is rather connected with specific volume influence

$$f_d = \left(\frac{2p^M}{p_e}\right)^\alpha, \quad p_e = p_r\,\exp\left[\frac{N(s) - \ln(1+e)}{\lambda^*(s)}\right], \tag{B.14}$$

where p^M is the mean stress at macro level and p_r is the reference pressure.

The model adopts a similar concept of the normal compression line similar to the one in Cam-Clay model where the compression line is being defined with the influence of suction pressure s as

$$\ln(1+e) = N(s) - \lambda^*(s)\ln\left(\frac{p^M}{p_r}\right), \tag{B.15}$$

with the slope of normal consolidation line λ^* and position of the normal consolidation line $N(s)$ defined by

$$N(s) = N + n_s \left\langle \ln \frac{s}{s_e} \right\rangle, \tag{B.16}$$

$$\lambda^*(s) = \lambda^* + l_s \left\langle \ln \frac{s}{s_e} \right\rangle, \tag{B.17}$$

$$s_e = s(S_r^M)^{(1/\gamma)}, \tag{B.18}$$

where N, n_s and l_s are model parameters.

The second order tensor $\boldsymbol{H}$ and factor f_u introduce the wetting-induced collapse of the clay and they are given by terms

$$\boldsymbol{H} = -c_i \frac{\hat{\boldsymbol{\sigma}}^m}{s\,\lambda^*(s)} \left(n_s - l_s \ln \frac{p_e}{p_r} \right) \langle -\dot{s} \rangle \quad \text{for } s > s_{exp} \text{ and } S_r < 1, \tag{B.19}$$

$$\boldsymbol{H} = \boldsymbol{0} \quad \text{otherwise}, \tag{B.20}$$

$$f_u = \left(\frac{f_d}{f_d^{SBS}} \right)^{m/\alpha}, \tag{B.21}$$

where s_{exp} is the suction at air expulsion value, f_d^{SBS} is the value of pyknotropy factor for stress states at the state boundary surface, m is a model parameter and factor c_i is defined as follows

$$c_i = \frac{3 + a^2 - f_d\, a\sqrt{3}}{3 + a^2 - f_d^{SBS} a\sqrt{3}}. \tag{B.22}$$

The pyknotropy factor f_d is defined by term

$$f_d^{SBS} = \| f_s \boldsymbol{\mathcal{A}} : \boldsymbol{N}^{-1} \|, \tag{B.23}$$

where the fourth-order tensor $\boldsymbol{\mathcal{A}}$ is given by

$$\boldsymbol{\mathcal{A}} = f_s \boldsymbol{\mathcal{L}} + \frac{1}{\lambda^*(s)} \boldsymbol{\sigma}^M \otimes \boldsymbol{1}. \tag{B.24}$$

On the microstructure level, the reversible behaviour linear in $\ln p^m$ vs. $\ln(1 + e^m)$ plot is adopted and the stresses are defined as follows

$$\dot{\boldsymbol{\sigma}}^m = \boldsymbol{I} \frac{p^m}{\kappa_m} \dot{\varepsilon}_V^m, \tag{B.25}$$

where p^m denotes the mean stress at micro level and κ_m is the model parameter. There is an explicit formulation of void ratio on the micro structural level given by term

$$e^m = \exp \left[\kappa_m \ln \frac{s_r}{p^m} + \ln(1 + e_r^m) \right] - 1, \tag{B.26}$$

where e_r^m and s_r are the material parameters representing an arbitrary reference value of void ration at micro level for the reference value of suction. The coupling factor f_m is defined as follows

$$f_m = 1 - (r_e)^m, \tag{B.27}$$

where r_e is the relative void ratio

$$r_e = \frac{e - e_d}{e_i - e_d}, \tag{B.28}$$

with

$$e_i = \exp\left[N(s) - \lambda^*(s) \ln\, p^M\right] - 1, e_d = e_m. \tag{B.29}$$

The summary of the model parameters follows:

φ_c**:** Critical state friction angle.

N**:** Position of the isotropic normal compression line.

λ^***:** Slope of normal compression lines of a saturated soil in the $\ln(p^M/p_r)$ vs $\ln(1+e)$ plane.

κ^***:** Slope of the macrostructural isotropic unloading line.

n_s**:** Parameter controlling the position of isotropic normal compression line with respect to suction.

l_s**:** Parameter controlling the slope of isotropic compression line with respect to suction.

e_r^m**:** Reference value of the microstructural void ratio for the reference suction s_r.

s_r**:** Reference value of suction

r**:** Parameter controlling stiffness in shear.

m**:** Controls the dependency of wetting-induced collapse on the overconsolidation ratio and the macropore occlusion by microporosity on relative void ratio.

κ_m**:** Specifies the dependency of microstructural swelling/shrinkage on the microstructural effective stress.

s_{e0}**:** The air entry value of suction for the (arbitrary) reference macrostructural void ratio e_0^M.

e_0^M**:** The reference macrostructural void ratio.

a_e**:** The ratio between the air expulsion and entry values of suction.

In addition, it is necessary to specify the initial values of the following state variables:

e**:** global void ratio.

a_{sc}**:** Specifies whether the current state belongs to the main drying branch of macrostructural WRC, main wetting branch or specify the position along the macrostructural scanning curve.

Bibliography

Alexandrescu, A. (2000). Generic: Change the Way You Write Exception-Safe Code — Forever.

Alonso, E. E., Romero, E., and Hoffman, C. (2011). Hydromechanical behaviour of compacted granular expansive mixtures: experimental and constitutive study. *Géotechnique*, 61(4):329–344.

Bathe, K. J. (1996). *Finite Element Procedures*. Prentice Hall, New Jersey.

Bauer, E. (1996). Calibration of a Comprehensive Hypoplastic Model for Granular Materials. *Soils and Foundations*, 36(1):13–26.

Bažant, Z. P. (1988). *Mathematical Modeling of Creep and Shrinkage of Concrete*. John Wiley&Sons, Chichester-Singapore.

Bažant, Z. P. and Xi, Y. (1995). Continuous retardation spectrum for solidification theory of concrete creep. *Journal of Engineering Mechanics*, 121(2):281–288.

Baweja, S. and Bažant, Z. P. (1995). Creep and shrinkage prediction model for analysis and design of concrete structures – model B3. *Materials and Structures*, 28(6):357–365. RILEM, Paris.

Bazant, Z. and Chern, J.-C. (1985). Concrete creep at variable humidity: constitutive law and mechanism. *Materials and Structures*, 18:1–20.

Bazant, Z., Cusatis, G., and Cedolin, L. (2004). Temperature Effect on Concrete Creep Modeled by Microprestress-Solidification Theory. *Journal of Engineering Mechanics-asce - J ENG MECH-ASCE*, 130:691–699.

Bazant, Z. and Jirásek, M. (2018). *Creep and Hygrothermal Effects in Concrete Structures*. Springer Dordrecht.

Bishop, A. W. (1959). The principle of effective stress. *Teknisk Ukeblad*, 106(39):859–863.

Bittnar, Z. and Šejnoha, J. (1996). *Numerical Methods in Structural Mechanics*. USA. ASCE Press, New York.

Bogacki, P. and Shampine, L. (1989). A 3(2) pair of Runge–Kutta formulas. *Applied Mathematics Letters*, 2(4):321–325.

Borja, R. I. (2004). Cam-Clay plasticity. Part V: A mathematical framework for three-phase deformation and strain localization analyses of partially saturated porous media. *Computer Methods in Applied Mechanics and Engineering*, 193(48):5301–5338. Advances in Computational Plasticity.

Butcher, J. C. (2016). *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, Ltd.

Büttner, J. and Simeon, B. (2002). Runge–Kutta methods in elastoplasticity. *Applied Numerical Mathematics*, 41(4):443–458.

CEB (2008). *CEB-FIP Model Code 1990*. Thomas Telford Ltd.

Chen, A. and Chen, W. (1975). Constitutive Relations for Concrete. *Journal of Engineering Mechechnics - ASCE*, 101(EM4):465–481.

Clang (2012). LLVM Coding Standards.

Crisfield, M. A. (1991). *Non-linear Finite Element Analysis of Solids and Structures*. UK. John Wiley & Sons Ltd, Chichester.

de Borst, R. (1987). Smeared cracking, plasticity, creep, and thermal loading - unified approach. *Computer Methods in Applied Mechanics and Engineering*, 62(1):89–110.

Ding, Y., Huang, W., Sheng, D., and Sloan, S. (2015). Numerical study on finite element implementation of hypoplastic models. *Computers and Geotechnics*, 68:78–90.

Fellin, W., Mittendorfer, M., and Ostermann, A. (2009). Adaptive integration of constitutive rate equations. *Computers and Geotechnics*, 36(5):698–708.

Fiedler, J. and Koudelka, T. (2011). SIFEL tools home page.

Gallipoli, D., Wheeler, S. J., and Karstunen, M. (2003). Modelling the variation of degree of saturation in a deformable unsaturated soil. *Géotechnique*, 53(1):105–112.

Gens, A., Valleján, A. B., Sánchez, M., Imbert, C., Villar, M., and van Geet, M. (2011). Hydromechanical behaviour of a heterogeneous compacted soil: experimental observations and modelling. *Géotechnique*, 61(5):367–386.

Google (2014). Google C++ Style Guide.

Grassl, P., Xenos, D., Nyström, U., Rempling, R., and Gylltoft, K. (2013). CDPM2: A damage-plasticity approach to modelling the failure of concrete. *International Journal of Solids and Structures*, 50(24):3805–3816.

Grunewald, J. (2000). *DELPHIN 4.1–Documentation, theoretical fundamentals*. TU Dresden.

Gudehus, G. (1996). A Comprehensive Constitutive Equation for Granular Materials. *Soils and Foundations*, 36(1):1–12.

Heeres, O. M. and de Borst, R. (2000). *Implicit integration of hypoplastic models*, chapter Numerical applications, pages 457–470. Springer Berlin Heidelberg, Berlin, Heidelberg.

Herle, I. and Kolymbas, D. (2004). Hypoplasticity for soils with low friction angles. *Computers and Geotechnics*, 31(5):365–373.

Hiley, R. A. and Rouainia, M. (2008). Explicit Runge–Kutta methods for the integration of rate-type constitutive equations. *Computational Mechanics*, 42(1):53–56.

Hughes, T. J. R. (1987). *The Finite Element Method. Linear Static and Dynamic Finite Element Analysis*. Prentice-Hall, inc. Englewood Cliffs, New Jersey, USA.

Janda, T. (2013). Hypoplastic Model for Clays: Issues Specific to the Finite Element Implementation. In Topping, B. and Iványi, P., editors, *Proceedings of the Fourteenth International Conference on Civil, Structural and Environmental Engineering Computing*, page Paper 193.

Jimenez-Fernandez, V. M., Jimenez-Fernandez, M., Vazquez-Leal, H., Muñoz-Aguirre, E., Cerecedo-Nuñez, H. H., Filobello-Niño, U. A., and Castro-Gonzalez, F. J. (2016). Transforming the canonical piecewise-linear model into a smooth-piecewise representation. *SpringerPlus*, 5(1):1612.

Jirásek, M. and Bazant, Z. P. (2002). *Inelastic Analysis of Structures*. John Wiley&Sons, Ltd, Chichester, United Kingdom.

Kolymbas, D. (1991). Computer-aided design of constitutive laws. *International Journal for Numerical and Analytical Methods in Geomechanics*, 15(8):593–604.

Koudelka, P. and Koudelka, T. (2005). Risk Assessment of a Heterogeneous Stratified Rock Cliff under an Elevated Road. In Logar, J., Gaberc, A., and Majes, B., editors, *Proceedings of XIIIth DEC on Geotechnical Engineering, Ljubljana, Slovenian Geotechnical Society*, pages 747–752.

Koudelka, P. and Koudelka, T. (2007). Advanced numerical models - Influence of partial material factors. In Kanda, J., Takada, T., and Furuta, H., editors, *Proceedings of 10th IC on Applications of Statistics and Probability in Civil Engineering - Tokyo*, volume Vol. 2, pages 597–598. Taylor & Francis Group.

Koudelka, T., Krejčí, T., and Kruis, J. (2011). *Modeling of Building constructions in SIFEL Environment*. CTU Reports, Prague, Czech Republic.

Koudelka, T., Krejčí, T., and Šejnoha, J. (2007). Studie termo-hygro-mechanického chování tlusté základové desky. *BETON-technologie, konstrukce, sanace*, 7(5):54–58.

Koudelka, T., Krejčí, T., and Šejnoha, J. (2009). Analysis of a Nuclear Power Plant Containment. In Topping, B. H. V., Costa Neves, L. F., and Barros, R. C., editors, *Proceedings of the Twelfth International Conference on Civil, Structural and Environmental Engineering Computing*, page paper 132, Stirlingshire, United Kingdom. Civil-Comp Press.

Koudelka, T., Kruis, J., Lepš, M., and Nodek, J. (2014). Fitting Model Parameters of Prestressed Concrete Bridge: Computational Aspects. In Fuis, V., editor, *Engineering Mechanics 2014*, pages 316–319. Brno University of Technology, Brno University of Technology.

Krejčí, T., Koudelka, T., and Brouček, M. (2014). Numerical modelling of consolidation processes under the water level elevation changes. *Advances in Engineering Software*, 72(Supplement C):166–178.

Krejčí, T., Koudelka, T., and Šejnoha, J. (2007). Modelling of Sequential Casting Procedure of Foundation Slabs. In Topping, B. H. V., editor, *Proceedings of the Eleventh International Conference on Civil, Structural and Environmental Engineering Computing*, page paper 115. Civil-Comp Press, Kippen, Stirling, Scotland, United Kingdom.

Krejčí, T., Koudelka, T., Šejnoha, J., and Kuklík, P. (2006). Numerical Analysis of Foundation Slabs. In Topping, B. H. V., Montero, G., and Montenegro, R., editors, *Proceedings of the Eighth International Conference on Computational Structures Technology*, page paper 63. Civil-Comp Press, Kippen, Stirling, Scotland, United Kingdom.

Krejčí, T., Koudelka, T., Šejnoha, J., and Zeman, J. (2009). Computer Simulation of Concrete Structures subject to Cyclic Temperature Loading. In Topping, B. H. V., Costa Neves, L. F., and Barros, R. C., editors, *Proceedings of the Twelfth International Conference on Civil, Structural and Environmental Engineering Computing*, page paper 131, Stirlingshire, United Kingdom. Civil-Comp Press.

Krejčí, T., Nový, T., Sehnoutek, L., and Šejnoha, J. (2001). *Structure - subsoil interaction in view of transport processes in porous media.* CTU Reports, Prague, Czech Republic.

Kruis, J., Koudelka, T., Bittnar, Z., and Petkovski, M. (2005). Hygro-Thermo-Mechanical Analysis of a Nuclear Power Plant Prestressed Concrete Reactor Vessel. In Topping, B. H. V., editor, *Proceedings of the Tenth International Conference on Civil, Structural and Environmental Engineering Computing*, page paper 176. Civil-Comp Press, Kippen, Stirlingshire, Scotland, United Kingdom.

Künzel, H. M. and Kiessl, K. (1996). Calculation of heat and moisture transfer in exposed building components. *International Journal of Heat and Mass Transfer*, 40(1):159–167.

Lemaitre, J. and Chaboche, J. L. (1994). *Mechanics of solid materials.* Cambridge University Press, Cambridge, United Kingdom.

Lewis, R. W. and Schrefler, B. A. (1998). *The finite element method in static and dynamic deformation and consolidation of porous media.* John Wiley & Sons, Chichester-Toronto.

Maděra, J. and Černý, R. (2005). TRANSMAT - A computer simultaion tool for modeling coupled heat and moisture transport in building materials. In *Proceedings of Workshop 2005*, pages 470–471. Faculty of Civil Engineering, Czech Technical University.

Mašín, D. (2012). Hypoplastic Cam-clay model. *Géotechnique*, 62(6):549–553.

Mašín, D. (2013). Double structure hydromechanical coupling formalism and a model for unsaturated expansive clays. *Engineering Geology*, 165(Supplement C):73–88.

Mašín, D. (2017). Coupled Thermohydromechanical Double-Structure Model for Expansive Soils. *Journal of Engineering Mechanics*, 143(9):04017067.

Mazars, J. and Pijaudier-Cabot, G. (1989). Continuum Damage Theory - Application to Concrete. *Journal of Engineering Mechanics*, 115(2):345–365.

Meyers, S. (2005). *Effective C++ (3rd Edition)*. Addison-Wesley Professional.

Miao, L., Houston, S. L., Cui, Y., and Yuan, J. (2007). Relationship between soil structure and mechanical behavior for an expansive unsaturated clay. *Canadian Geotechnical Journal*, 44(2):126–137.

Mozilla (2014). Mozilla.org C++ Portability Guide.

Myšáková, E. and Lepš, M. (2012). Method for constrained designs of experiments in two dimensions. In Náprstek, J. and Fischer, C., editors, *Engineering Mechanics 2012*, pages 901–914. Czech Academy of Science, Institute of Theoretical and Applied Mechanics.

Neto, E. A. S., Perić, D., and Owen, D. R. J. (2008). *Computational Methods for Plasticity*. John Wiley & Sons, Ltd.

Niemunis, A. (2002). *Extended hypoplastic models for soils.* PhD thesis, Ruhr-University, Bochum.

Ottosen, N. S. and Ristinmaa, M. (2005). *The Mechanics of Constitutive Modeling*. Elsevier Science.

Papa, E. and Taliercio, A. (1996). Anisotropic Damage Model for the Multiaxial Static and Fatigue Behaviour of Plain Concrete. *Engineering Fracture Mechanics*, 55(2):163–179.

Pietruszczak, S. and Mróz, Z. (1981). Finite element analysis of deformation of strain-softening materials. *International Journal for Numerical Methods in Engineering*, 17(3):327–334.

Romero, E., Della Vecchia, G., and Jomi, C. (2011). An insight into the water retention properties of compacted clayey soils. *Géotechnique*, 61(4):313–328.

Roscoe, K. H. and Burland, J. B. (1968). On the Generalized Stress-Strain Behaviour of Wet Clay. *Engineering Plasticity*, 1:553–609.

Sánchez, M., Gens, A., and Do Nascimento Guimarães, L. (2005). A double structure generalised plasticity model for expansive materials. *International Journal for Numericaland Analytical Methods in Geomechanics*, 29:751–787.

SIFEL (2022). SIFEL home page.

Simo, J. and Hughes, T. (2000). *Computational Inelasticity*. Interdisciplinary Applied Mathematics. Springer New York.

Skrzypek, J. and Ganczarski, A. (1999). *Modeling of Material Damage and Failure of Structures*. Springer-Verlag Berlin Heidelberg, Germany.

Sloan, S. W. (1987). Substepping schemes for the numerical integration of elastoplastic stress–strain relations. *International Journal for Numerical Methods in Engineering*, 24(5):893–911.

Stepanov, A. and Russo, G. L. (2008). STLport: An Interview with A. Stepanov.

Svoboda, L. (2012). MIDAS home page.

Tamagnini, C., Salciarini, D., and Ragni, R. (2013). Implementation of a 6-dof hypoplastic macroelement in a finite element code. In *International Conference on Computational Geomechanics - Comgeo III, Cracovia, Poland*, page 08.

Tamagnini, C., Viggiani, G., Chambon, R., and Desrues, J. (2000). Evaluation of different strategies for the integration of hypoplastic constitutive equations: Application to the CLoE model. *Mechanics of Cohesive-frictional Materials*, 5(4):263–289.

Tschoegl, N. W. (1971). A general method for determination of approximations to the spectral distributions from the transient response functions. *Rheologica Acta*, 10:595–600.

Tschoegl, N. W. (1989). *The phenomenological theory of linear viscoelastic behavior*. Germany. Springer-Verlag, Berlin.

Villar, M. V. and Lloret, A. (2007). Dismantling of the first section of the FEBEX in situ test: THM laboratory tests on the bentonite blocks retrieved. *Physics and Chemistry of the Earth, Parts A/B/C*, 32(8):716–729. Clay in natural and engineered barriers for radioactive waste confinement - Part 2.

Vráblík, L., Štroner, M., and Urban, R. (2008). Zaměření tvaru nosné konstrukce mostu přes Labe v Mělníku. *Beton - křižovatka požadavků*, 8(4):84–87.

Widder, D. (1946). *The Laplace transform*. Princeton University Press.

Zienkiewicz, O. C. and Taylor, R. L. (2000). *The Finite Element Method*, volume Volume 1: The Basis. Butterworth-Heinemann.