ČERVENKA CONSULTING

Červenka Consulting s.r.o.
Na Hřebenkách 55
150 00 Prague
Czech Republic
Phone: +420 220 610 018
E-mail: cervenka@cervenka.cz
Web: https://www.cervenka.cz

# ATENA Module for New Multi-spiral Reinforced Concrete and Steel Column-Beam Connections

*Project Result TM04000013-V1*

## Written by

Jan Červenka

Michal Ženíšek

Tomáš Altman

Version 3.0

**Prague, March 2025**

*Acknowledgements:*

*The software was developed with partial support of* ***TAČR DELTA 2 Programme***

T A

Č R

*("DELTA 2 Funding programme for applied research, experimental development and innovation", project TM04000013 "Virtual modelling of green concrete - structures with novel multi-spiral reinforcement and steel members")*

# Contents


Executive Summary ..... 4
1 Introduction ..... 5
2 User's Manual ..... 6
3 Analysis of Functional Requirements ..... 12
3.1 User Interface Requirements ..... 12
3.2 Material Model Functionalities ..... 12
3.3 Analysis and Simulation ..... 13
3.4 Output, Visualization, and Reporting ..... 13
3.5 Validation and Traceability ..... 13
3.6 Extensibility and Interoperability ..... 13
4 Technical Documentation and Theoretical Manual ..... 14
4.1 Nonlinear Structural Analysis ..... 14
4.2 Constitutive Models ..... 16
4.2.1 Constitutive model for concrete ..... 16
4.3 Safety formats for nonlinear analysis of reinforced concrete ..... 26
5 Programming Manual ..... 30
5.1 Setting Up the Environment ..... 30
5.2 XML part ..... 30
5.3 Python part ..... 31
5.4 Run Time Analysis, Output and Post-Processing ..... 42
5.5 Advanced Usage (Optional) ..... 42
6 Examples and Validation ..... 43
6.1 MRCS Joint Experiments ..... 43
6.1.1 Static Simulation – Push-over Analysis ..... 47
6.1.2 Cycling Simulation ..... 48
6.2 Cycling Tests of Multi-Story Structure ..... 51
6.3 Shear Failure in Pre-stressed Girder ..... 55
6.4 Punching Shear fib WG 2.4.1 Validation ..... 56
6.5 Summary of all Recent ATENA Validation Benchmarks ..... 58
6.6 Parametric Generation of New-MRCS Joints ..... 60
7 Conclusion ..... 62
References ..... 63

# Executive Summary

This document provides both **user guidance and technical documentation** for the new **ATENA software module** developed specifically for modeling and simulating the **new-MRCS structural system**, which integrates **Multi-spiral Reinforced Concrete (MRC) columns** with **Steel (S) beams**.

The first section introduces the **user interface**, focusing on the definition of new-MRCS structural elements. It includes step-by-step instructions for selecting and defining appropriate **material models** for both concrete and reinforcement, as well as for applying loads and configuring a suitable **loading history**.

The following sections present the **technical documentation**, beginning with a summary of the **functional requirements** as originally specified at the start of the project. It details the **numerical methods**, modeling strategies, and **implementation techniques**, including the data structures used in the software's development. The **programming manual** explains the core algorithms and their integration into the ATENA framework.

The final section showcases **application examples** of the new module, particularly focusing on **new-MRCS beam-column joints**, and presents the **validation results** based on comparison with experimental data.

# 1 Introduction

The **CeSTaR-3 bilateral research project** contributes significantly to the strategic objectives outlined for the 2020–2030 horizon by promoting innovation in structural engineering. A central theme of the project is the application of advanced construction technologies that enhance sustainability without compromising structural efficiency, safety, or economic viability.

One of the key innovations explored in this project is the development of the **new-MRCS structural system**, which combines **Multi-spiral Reinforced Concrete (MRC) columns** with **Steel (S) beams**. This system offers a novel alternative to conventional Reinforced Concrete (RC) or Steel (S) structural systems typically used in buildings. Designed with a focus on carbon footprint reduction, the new-MRCS system serves as a sustainable evolution of the traditional RCS (Reinforced Concrete Column and Steel Beam) system.

The **new-MRCS system** integrates several environmentally conscious and structurally effective design strategies:

1. **Prefabricated High-Strength Green Concrete Components**

   The system enables the use of prefabricated structural units, which can be assembled on-site. Prefabrication not only enhances construction efficiency and schedule flexibility but also lowers carbon emissions—by an estimated 10% per cubic meter of precast concrete. Moreover, careful sequencing of construction tasks allows the use of **90-day strength concrete** instead of conventional **28-day strength concrete**. This longer curing period supports the use of **Supplementary Cementitious Materials (SCMs)** in place of Ordinary Portland Cement (OPC), significantly reducing $CO_2$ emissions. High-strength materials also allow smaller structural cross-sections, resulting in savings in both materials and transport-related emissions.

2. **Multi-Spiral Transverse Reinforcement**

   The concrete columns in the new-MRCS system utilize **multi-spiral reinforcement layouts** instead of conventional rectilinear ties. This approach not only reduces steel consumption but also enhances seismic performance, as demonstrated in prior research. Furthermore, multi-spiral reinforcement is well-suited to prefabrication and can be manufactured automatically in precast facilities, further streamlining the production process and enabling wider use of SCM-based concrete.

3. **Use of SCMs in Place of Portland Cement**

   Replacing OPC with SCMs is a proven method to decrease the carbon intensity of concrete. In the Czech Republic, large-scale use of fly ash in dam construction (e.g., the Orlík Dam built between 1958–1960) demonstrated the long-term viability of such mixes. In-situ testing after 55 years revealed a substantial increase in concrete strength, confirming durability and performance over time—even when lower-quality fly ash was used. Modern SCMs offer higher quality and are readily available as industrial by-products, making them an attractive and sustainable option for concrete production.

This document contains the program documentation of ATENA software module specifically developed to facilitate the modelling and simulation of the New-MRCS connections taking into account the new methods and models developed within the CeSTaR-3 project.

# 2 User's Manual

This section describes the new user interface and dialogs of the new ATENA module for the New-MRCS structural elements. It describes the installation of the new plugin and explains the main user's steps in its application to New-MRCS joints.

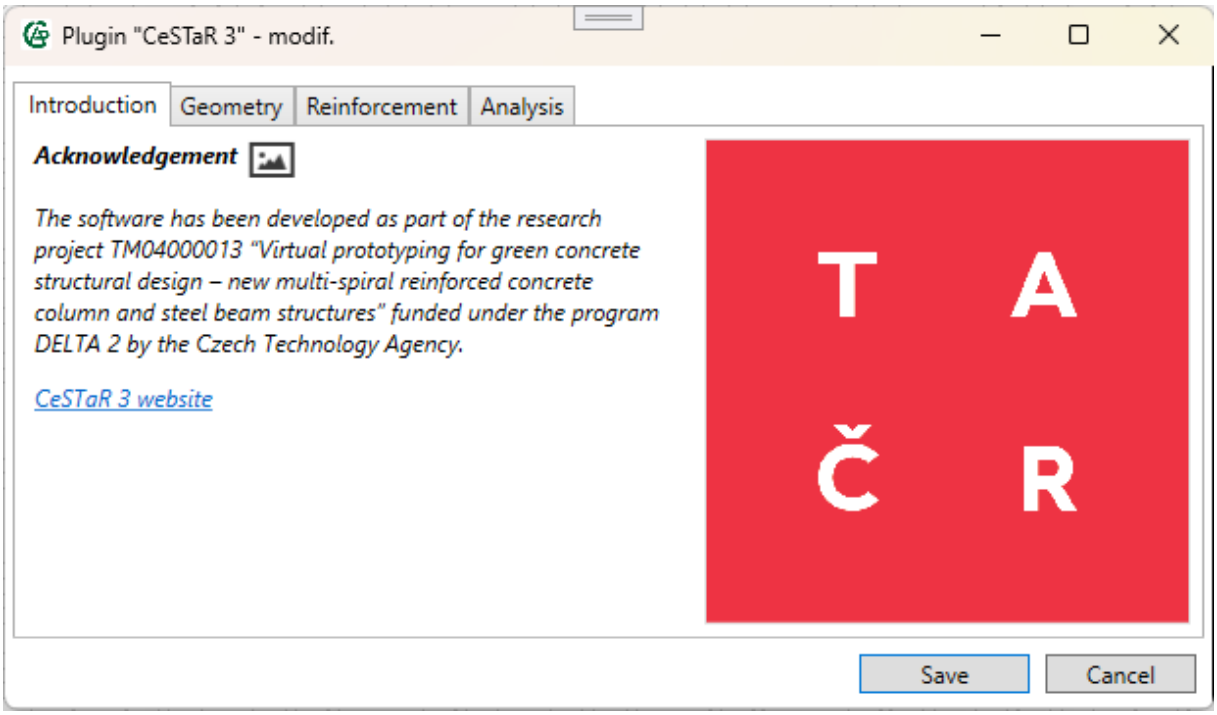


Fig. 1: Plugin - Acknowledgement

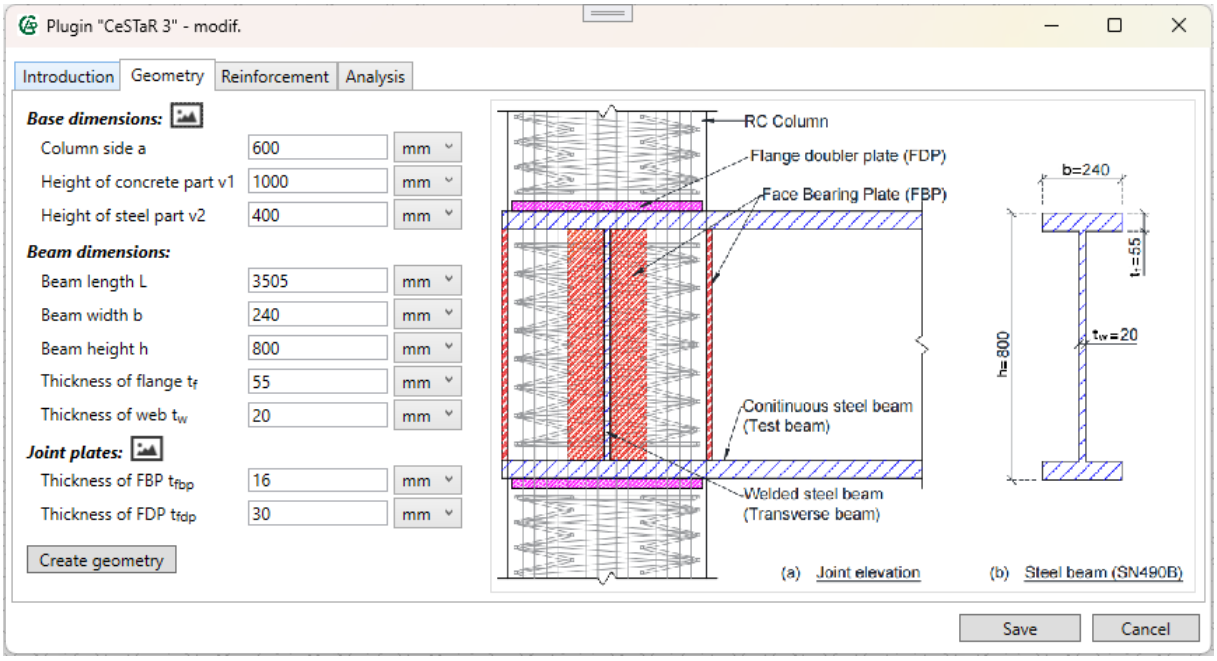


Fig. 2: Plugin - Geometry tab

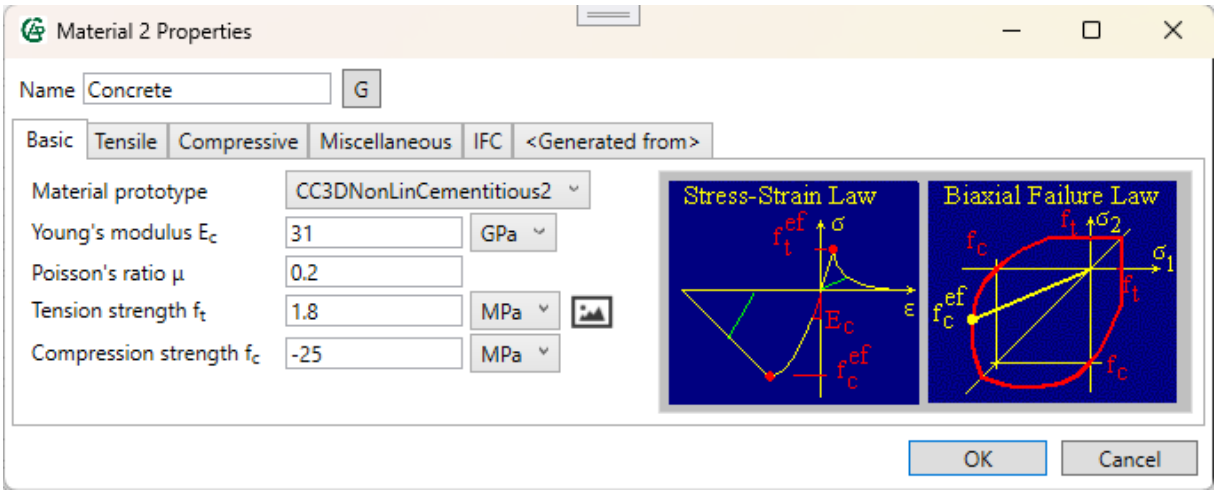


Fig. 3: Concrete material dialog

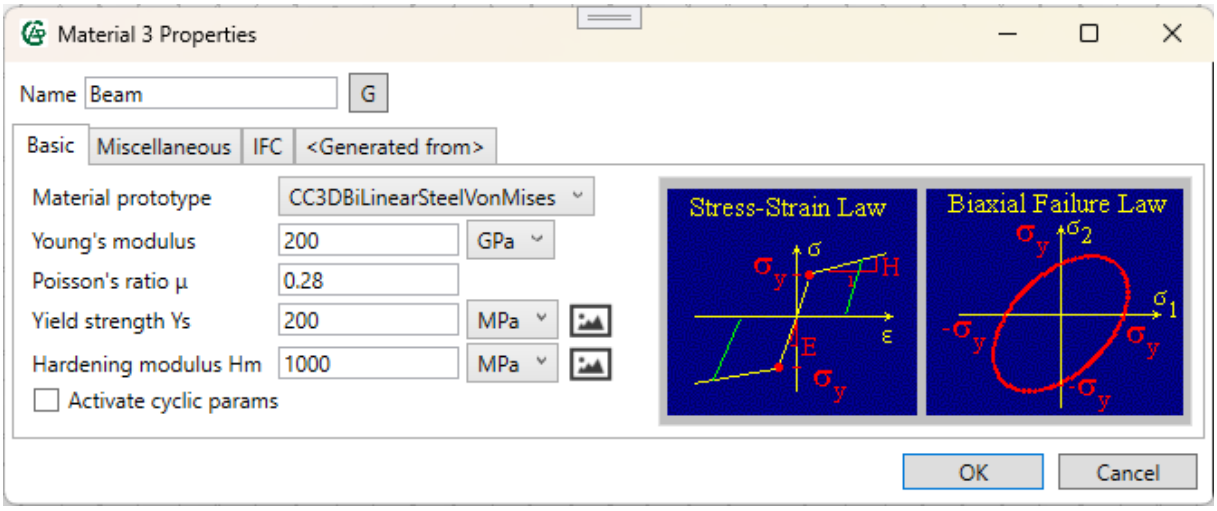


Fig. 4: Steel material dialog

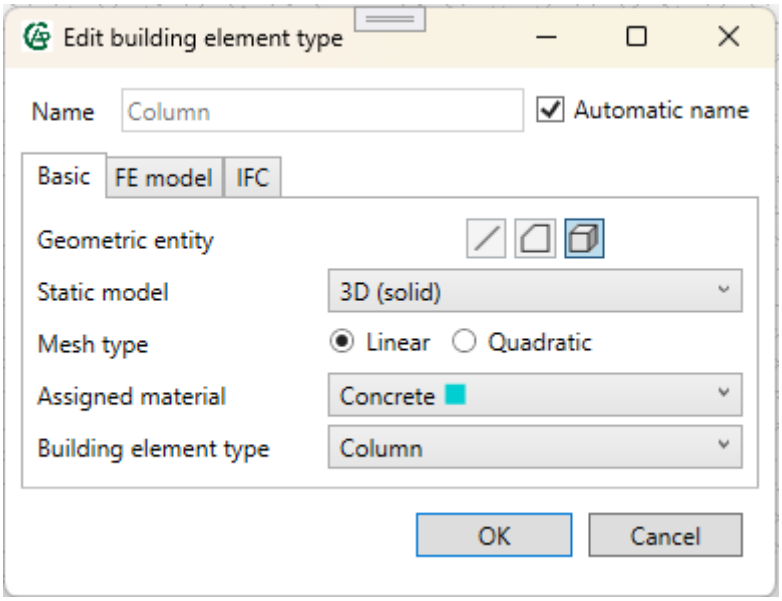


Fig. 5: Building element dialog

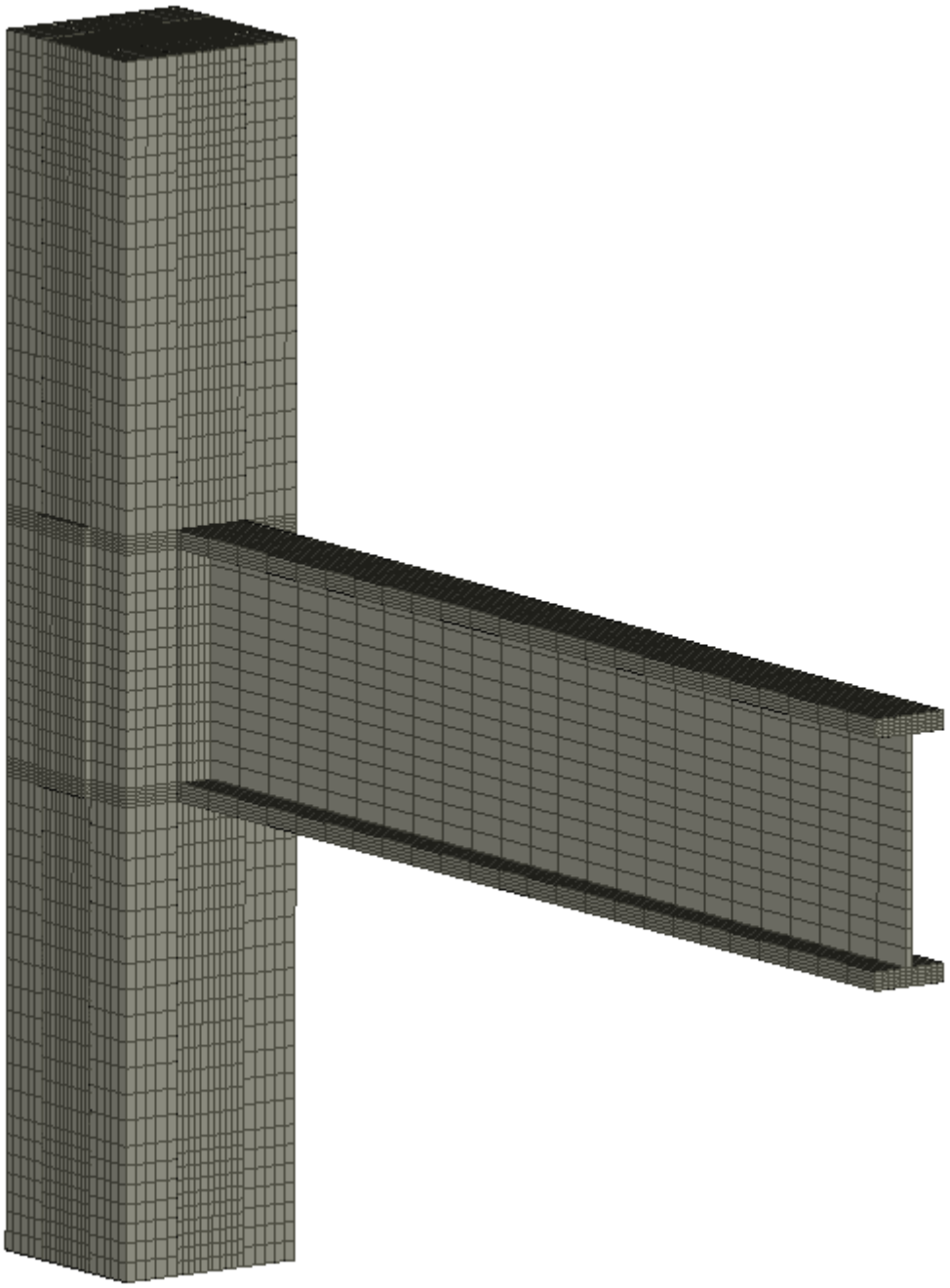

Fig. 6: FE mesh

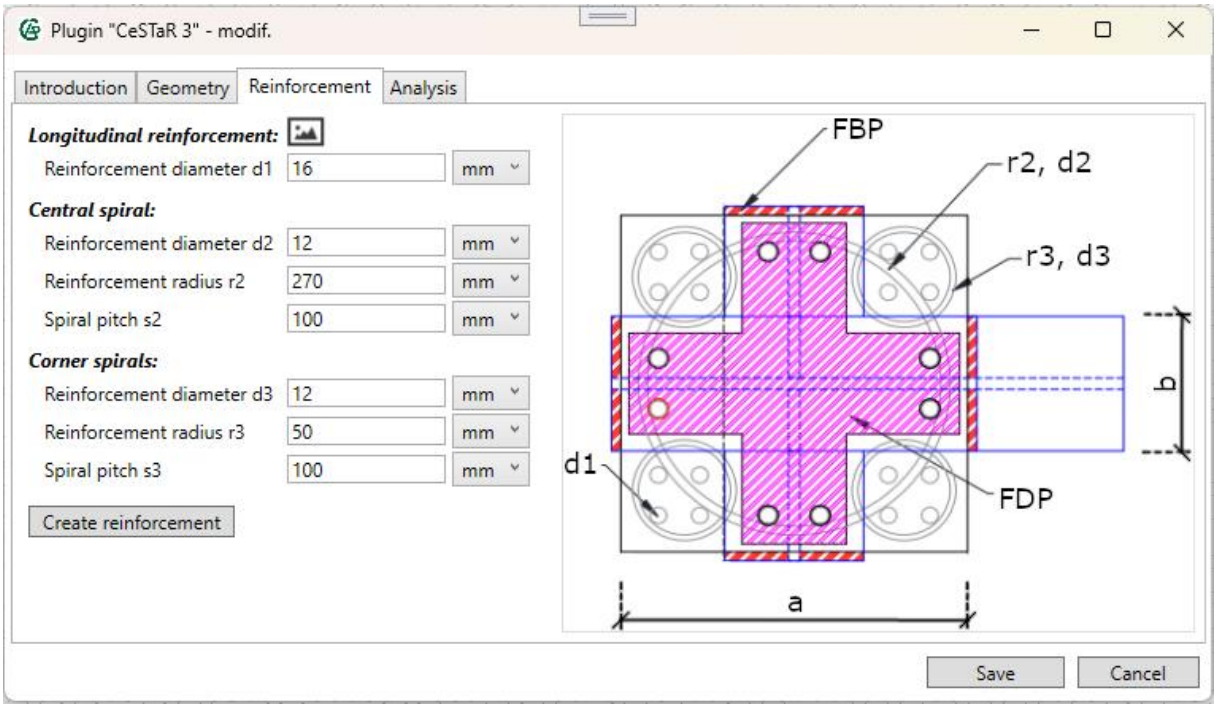


Fig. 7: Plugin - Reinforcement tab

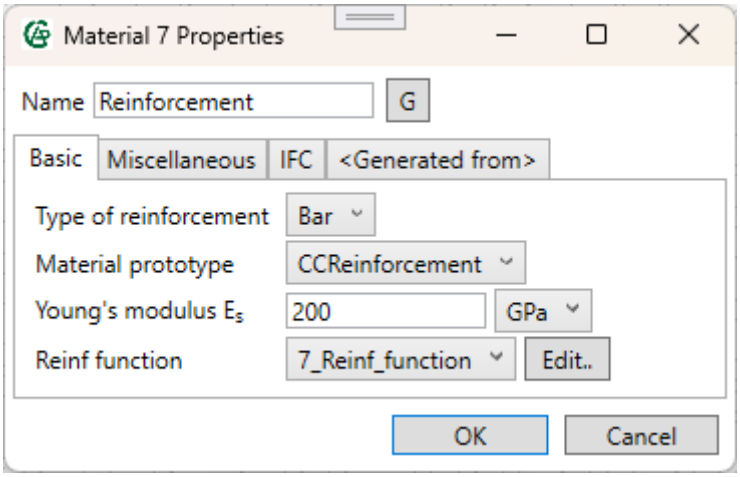


Fig. 8: Reinforcement material dialog

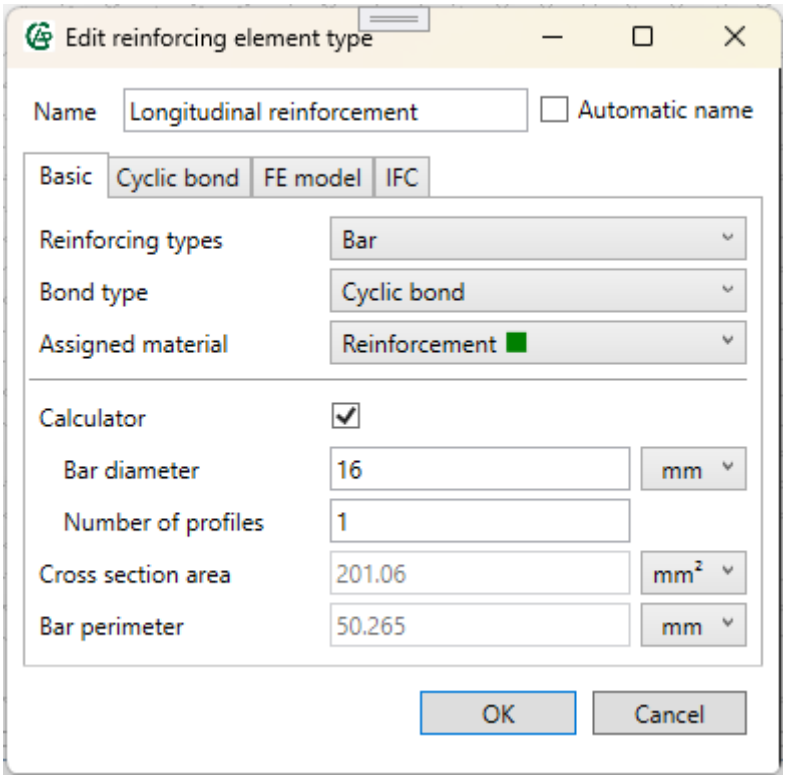


Fig. 9: Reinforcement element dialog – Basic tab

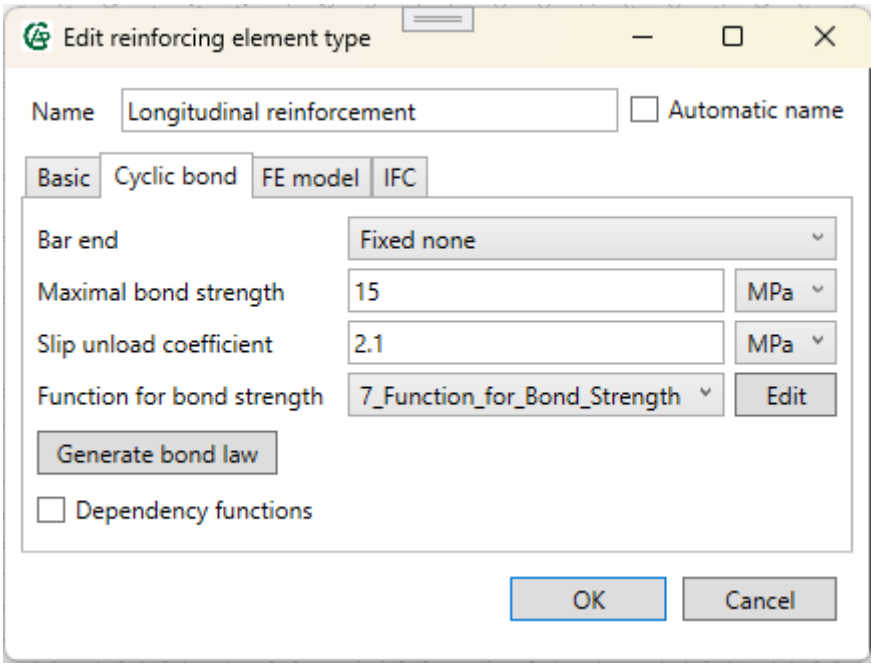


Fig. 10: Reinforcement element dialog – Cyclic bond

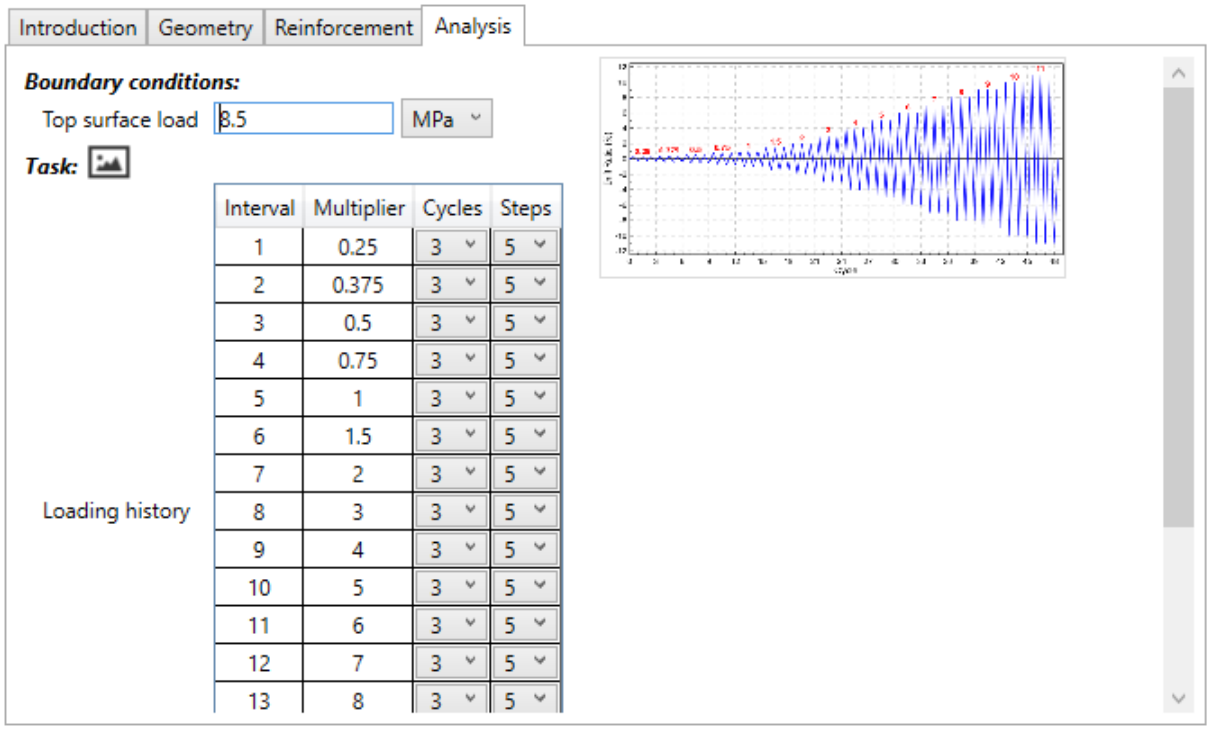


Fig. 11: Plugin - Analysis tab

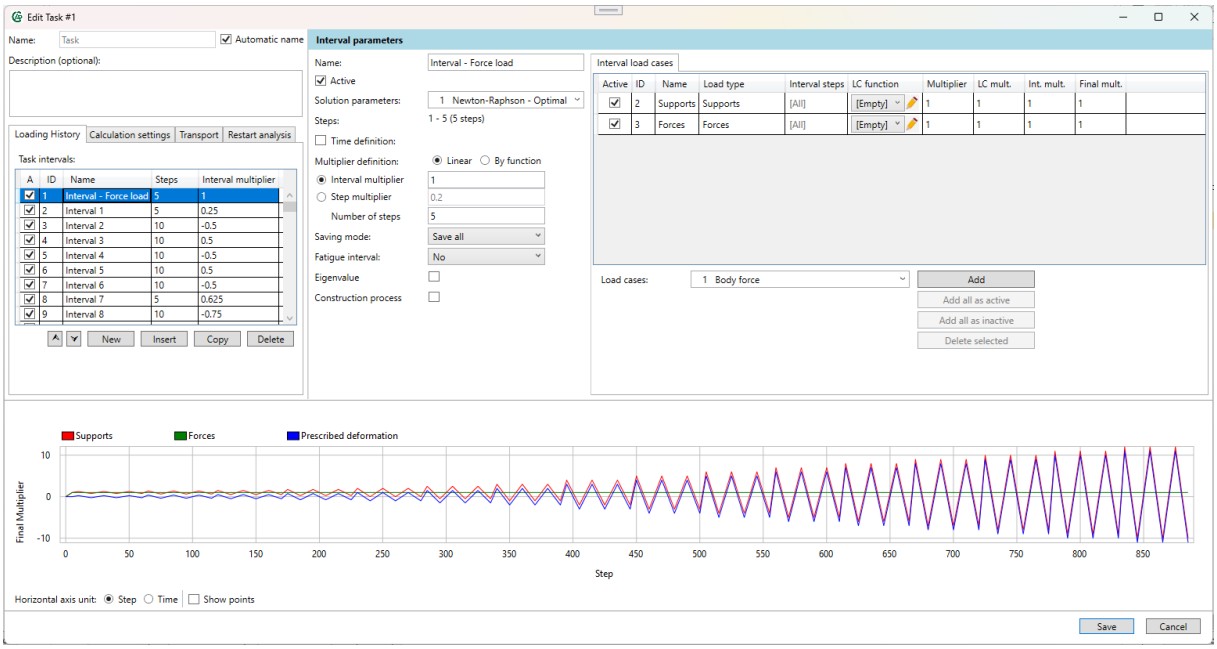


Fig. 12: Task dialog

# 3 Analysis of Functional Requirements

This chapter outlines the functional requirements for the newly developed module in **ATENA software**, aimed at modeling and simulating the **New-MRCS structural system**—a novel composite system integrating **Multi-spiral Reinforced Concrete (MRC) columns** and **Steel (S) beams**. The module supports advanced material modeling, prefabrication-oriented detailing, and sustainable design practices through seamless integration of experimental research and numerical analysis.

---

## 3.1 User Interface Requirements

**Structural Element Definition**

- Support intuitive **graphical and parametric definition** of MRC columns and S-beams.
- Enable **hybrid element assembly** (column-beam nodes, joint regions).
- Provide templates for commonly used **joint geometries** and **element topologies**.

**Material Assignment**

- Allow assignment of **custom or pre-defined concrete models**, including:
    - Green high-strength concrete
    - SCM-enhanced concrete
- Support **advanced steel reinforcement modeling**, including:
    - High-strength spiral reinforcement
    - Ductile S-beams

**Load and Boundary Conditions**

- Enable assignment of:
    - Static and cyclic loads
    - Monotonic and multi-phase load histories
    - Seismic-type displacements and reversals
- Allow definition of multiple **load combinations and loading sequences**

---

## 3.2 Material Model Functionalities

**Concrete Modeling**

- Support **nonlinear constitutive models** suitable for high-strength and green concrete mixes
- Enable creep, shrinkage, and aging effects for long-duration simulations
- Support models allowing **partial replacement of Portland cement with SCMs**

**Reinforcement Modeling**

- Include **multi-spiral reinforcement geometry generator**
- Enable spatial variation of spiral pitch, diameter, and confinement zones
- Simulate bond-slip behavior and transverse confinement effects

## 3.3 Analysis and Simulation

**Solver Integration**

- Full integration with ATENA's **nonlinear static and dynamic solvers**
- Support for **large deformation**, **crack propagation**, and **material degradation**
- Allow **incremental-iterative solution procedures** with adjustable convergence criteria

**Computational Features**

- Adaptive meshing for joint regions and spiral-reinforced zones
- Option to export simulation results for **post-processing and comparison with experimental data**
- Parallel processing support for enhanced performance

## 3.4 Output, Visualization, and Reporting

- Real-time graphical display of:
  - Load-displacement curves
  - Stress/strain contours in concrete and steel
  - Crack patterns and failure mechanisms
- Export options:
  - Raw simulation data
  - Summary reports in PDF and Excel formats
  - Graphical snapshots of model stages and final states

## 3.5 Validation and Traceability

- Built-in support for **benchmark validation examples** using available experimental data
- Enable tracking of:
  - Model versions and parameters
  - Input assumptions and boundary conditions
- Log output for solver steps, convergence, and numerical warnings

## 3.6 Extensibility and Interoperability

- Compatible with previous ATENA projects and file formats
- Support for data import/export from:
  - AutoCAD/DXF
  - Excel (for material definitions and loading protocols)
- API hooks for research-driven customization and scripting (e.g., Python)

# 4 Technical Documentation and Theoretical Manual

This document provides the **Technical and Theory Manual** for the newly developed **ATENA software module** dedicated to the modeling and simulation of the **New-MRCS structural system**, an innovative structural concept integrating **Multi-spiral Reinforced Concrete (MRC) columns** with **Steel (S) beams**. Developed as part of the **CeSTaR-3 bilateral research project**, the module aims to support sustainable design practices and advanced structural performance through the integration of modern construction materials, prefabrication techniques, and state-of-the-art numerical methods. This manual is intended for researchers, structural engineers, and software developers who require a deeper understanding of the theoretical formulations, numerical implementation, and system architecture that underpin the software. It complements the user documentation by providing the scientific background, mathematical models, and programming structure necessary for effective use, extension, and validation of the New-MRCS simulation tools within the ATENA framework.

## 4.1 Nonlinear Structural Analysis

The nonlinear finite element analysis (NLFEA) was first applied for of reinforced concrete structures already in the 70`s by landmark works of Ngo & Scordelis (1967) [46], Rashid (1968) [48] and Červenka V. & Gerstle (1971) [9]. Various material models for concrete and reinforced concrete were developed in 70's, 80's and 90's such as for instance Suidan & Schnobrich (1973) [51], Lin & Scordelis (1975) [42], De Borst (1986) [32], Rots & Blaauwendrad (1989) [49], Pramono & Willam (1989) [47], Etse (1992) [27] or Lee & Fenves (1998) [41]. These models are typically based on the finite element method and a concrete material model is formulated as a constitutive model applied at each integration point for the evaluation of internal forces. It was soon discovered that material models with strain softening, if not formulated properly, exhibit severe mesh dependency (De Borst & Rots 1989) [35], and tend to zero energy dissipation if the element size is reduced (Bažant 1976) [2].

The crack band approach was introduced by Bažant and Oh (1983) [3] to remedy the convergence towards zero energy dissipation. A more rigorous solution of the ill-posed nature of the strain softening problem represent nonlocal or higher-order continuum models: such as non-local damage model by Bažant & Pijaudier-Cabot (1987) [4], gradient plasticity model by de Borst & Muhlhaus (1992) [34] or gradient damage model by de Borst et. al. (1996) [33]. The non-local models introduce additional material parameters related to an internal material length scale. These models are mathematically rigorous, but currently are seldom used in engineering practice or available in commercial finite element codes.

The limitations of the crack band model in practical engineering calculations, namely when it comes to large finite elements or in the presence of reinforcement, were clearly understood already in the work of Bažant and Oh (1983) [3]. These limitations were described and treatment was proposed by Červenka (2018a) [15] for the cases when large element sizes as well as small ones are used in the finite element nonlinear analyses.

Until recently the application of these advanced nonlinear analyses in design practice was difficult since it was not compatible in many cases with the existing design codes and engineering practice, which is still mostly based on linear analysis. The fib Model Code 2010 (MC2010) [30] introduced a comprehensive system for the treatment of safety and model uncertainty for structural assessment and design based on nonlinear analysis. This discrepancy was eliminated by introduction of the global safety formats in the fib Model Code 2010 [30] and in the evolution of the new fib Model Code 2020 and of the Eurocodes. The model uncertainty of the approaches based on NLFEA reflects the knowledge quality of numerical simulations and serves as an important reliability tool.

A design verification based on NLFEA offers an alternative to the standard design method based on elastic analysis. It is typically used for complex structures, where a robust reliability assessment is substantiated and in the cases of accidents and forensic investigations.

The original publications to the finite element method are due to J. Argyris 0 [1], M.J. Turner and R.W.Clough (1956) [54], and O.C. Zienkiewicz (1967) [56]. It became the most significant tool for structural analysis and a basis of computer codes for solutions of engineering problems. A displacement-based version of the finite element method is the most popular version for the engineering applications and is used in this report.

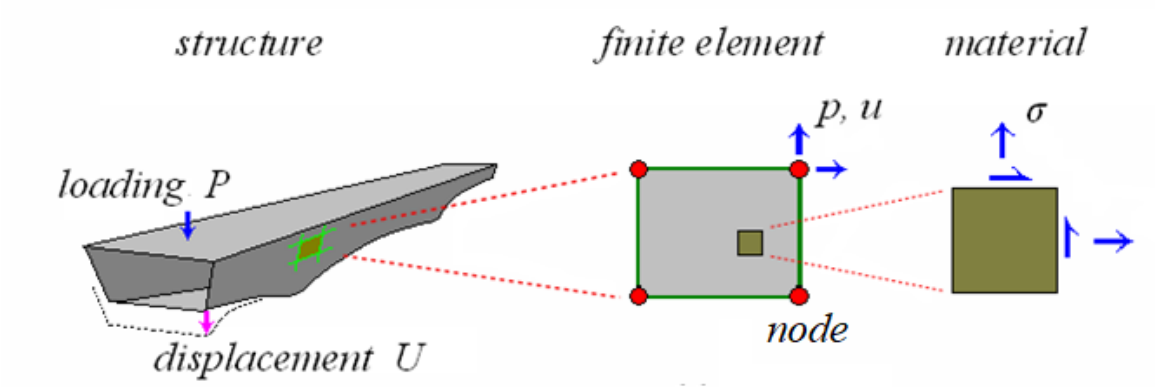


Fig. 1. NLFEA solution scheme.

The exact solution of the continuum problem is approximated by a solution of the set of linear equations for discrete number of nodal displacements. This discretization is illustrated in Fig. 1, where three solution levels are recognized, namely structure (describing the engineering task to be solved), finite element (defining an approximation by the finite element mesh), and material (defining a non-linear behavior). A nonlinear response (due to material behavior or geometry) is solved by an iterative Newton-Raphson method. The solution is illustrated in Fig. 2and is described by the following set of matrix equations:

$$K_{n,i}\,\Delta U_{n,i} = P_n - R_{n,i} \tag{1}$$

$$K_{n,i} = \sum_{j=1}^{m} k_j,\ R_{n,i} = \sum_{j=1}^{m} r_j\ ,\ U_{n+1,i} = U_{n,i} + \Delta U_{n,i} \tag{2}$$

In this: $K_i$ – stiffness matrix of the structure at the load step $n$ and iteration $i$, $\Delta U_i$ – vector of displacement increments, $U_{n,i}$ – vector of total displacements of the structure in step $n$ and iteration $i$, $P_n$ – vector of total nodal forces at the load step $n$, $R_{n,i}$ – vector of resisting nodal forces, $k_j$ – element stiffness matrix, $r_j$ – vector of resisting element nodal forces.

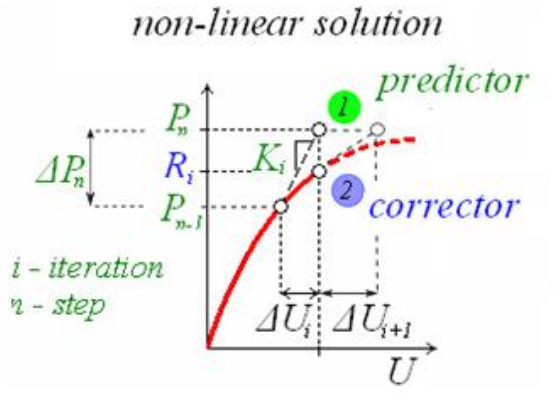


Fig. 2. Non-linear iterative solution.

The number of rows in the matrix Eq. (1) corresponds to the number of nodal displacements in the structure. Eq. (2) represents the assembly of the global stiffness matrix, nodal forces and nodal displacements as a summation from local element entities, where $m$ is the number of finite elements. It should be noted that that the above formulation is schematic and does not indicate the detail structure of the matrices due to the displacement boundary conditions and corresponding reactions.

The non-linear response is solved in increments. In a load step, a displacement increment $\Delta U_{n,i}$ due to the load increment $\Delta P_n$ is estimated by the *predictor*, marked as point 1. It estimates the response based on the element stiffness matrix $k$ , which is defined in the matrix format as follows:

$$\varepsilon = Bu, \;\; \sigma = D\varepsilon, k = \int_v \; B^T D \; B \; dv \tag{3}$$

where $\varepsilon$ – element strains, $\sigma$ – element stresses, $B$ – strain-displacement matrix for given finite element formulation, u – element nodal displacements, $k$ – element stiffness matrix. The Integration is performed over the volume of the finite element. The matrix $B$ can also include high order terms and reflect the geometrical non-linearity. Matrix $D$ is based on the linear elastic theory (with two parameters, elastic modulus and Poison's ratio). It is based on a tangent elastic modulus, but alternative methods are available based on other values. Eq.(3) follows form the theorem of minimum of total potential energy and more detailed derivation can be found, for example, in Zienkiewicz (1967) [56].

Due to the linear approximation the solution obtained by the predictor differs from a nonlinear response. Therefore, a correct resistance is found by the *corrector* as follows:

$$\sigma = F(\sigma, \varepsilon), \; r = \int_v \; B^T \sigma \; dv \tag{4}$$

In this $F(\sigma, \varepsilon)$ – constitutive law, and $r$ – vector of resisting nodal forces consistent with the constitutive law. The constitutive law is a function of stress and strain and typically include additional parameters. In this presentation two laws, based on the theories of plasticity and fracture are described.

In case of a unique solution the difference $P_n - R_{n,i}$ wannish and the iterative solution converges to a nonlinear response. The process can be refined by adopting methods to improve the iteration stability, such as the methods of line-search and arc-length.

## 4.2 Constitutive Models

### 4.2.1 Constitutive model for concrete

Considering a large variety of existing material models or approaches to nonlinear analysis, it is impossible to provide guidelines or recommendations for their general application in engineering practice. It is therefore evident that very detailed guidelines or approaches for the treatment of for instance the modelling uncertainties can be only specific to certain class of constitutive models or even to a particular material model or even only to a single software. A comprehensive summary of this topic is available for instance at Jirásek & Bažant (2001) [40]. The models presented in this paper are based on the research of the authors, which are implemented in the finite element software ATENA, Červenka et al. (2025) [20] . Some of the conclusions are valid only for this software or at most are applicable for the class of models based on smeared crack approach and crack band method. However, the presented safety formats or the general treatment of model uncertainties is applicable for other nonlinear models for concrete structures as well. The material model used in this paper is a fracture-plastic model described in more detail in Červenka at al. (1998) [12] and Červenka & Papanikolaou (2008) [13].

The constitutive model formulation assumes small strains and is based on the strain decomposition into elastic ($\varepsilon_{ij}^{e}$), plastic ($\varepsilon_{ij}^{p}$) and fracture ($\varepsilon_{ij}^{f}$) components. The stress development is described by a rate equation reflecting the progressive damage (concrete cracking) and plastic yielding (concrete crushing):

$$\dot{\sigma}_{ij} = D_{ijkl} \cdot (\dot{\varepsilon}_{kl} - \dot{\varepsilon}^{p}_{kl} - \dot{\varepsilon}^{f}_{kl}) \tag{5}$$

The flow rules govern the evolution of plastic and fracturing strains:

$$\text{Plastic model:} \qquad \dot{\varepsilon}^{p}_{ij} = \dot{\lambda}^{p} \cdot m^{p}_{ij}, m^{p}_{ij} = \frac{\partial g^{p}}{\partial \sigma_{ij}} \tag{6}$$

$$\text{Fracture model:} \qquad \dot{\varepsilon}^{f}_{ij} = \dot{\lambda}^{f} \cdot m^{f}_{ij}, \; m^{f}_{ij} = \frac{\partial g^{f}}{\partial \sigma_{ij}} \tag{7}$$

where $\dot{\lambda}^{p}$ is the plastic multiplier rate and $g^{p}$ is the plastic potential function, $\dot{\lambda}^{f}$ is the inelastic fracturing multiplier and $g^{f}$ is the potential defining the direction of inelastic fracturing strains. The multipliers are evaluated from the consistency conditions.

#### 4.2.1.1 Compressive failure model

Compressive behavior is governed by the plasticity of concrete in multiaxial stress state with nonlinear hardening/softening. In the plasticity model, Eq.(5), the plastic strain rate $\dot{\bar{\varepsilon}}_{p}$ is calculated from the consistency condition. $\lambda_{p}$ is the plastic multiplier, and $\bar{n}$ and $\bar{m}$ are stress derivatives of the plastic and potential surface respectively. The plastic surface $F_{p}$ is defined by the three-parameter criterion, ($f_{c}, f_{t}, e$), according to Menetrey & Willam (1995), Fig. 3.

$$F_{p} = \left[\sqrt{1.5}\,\frac{\rho}{\hat{f}_{c}}\right]^{2} + m\left[\frac{\rho}{\sqrt{6}\hat{f}_{c}}\,r(\theta,e) + \frac{\xi}{\sqrt{3}\hat{f}_{c}}\right] - c = 0 \tag{8}$$

$$m = 3\,\frac{\hat{f}_{c}^{2} - f_{t}^{2}}{\hat{f}_{c} f_{t}}\,\frac{e}{e+1}$$

$$r(\theta,e) = \frac{4(1-e^{2})\cos^{2}\theta + (2e-1)^{2}}{2(1-e^{2})\cos\theta + (2e-1)\left[4(1-e^{2})\cos^{2}\theta + 5e^{2} - 4e\right]^{\frac{1}{2}}}$$

In the above equations, $(\xi, \rho, \theta)$ are the Haigh-Westergaard stress coordinates and $f_{c}$ and $f_{t}$ are the compressive strength and tensile strength, respectively. Parameter $e \in (0.5, 1.0)$ defines the roundness of the Menétrey-Willam failure surface, with a recommended value $e$ = 0.52. Note, that the shape of the failure function reflects the typical behavior of concrete, namely the confinement effect, described by the cone opening with increased hydrostatic pressure, and a quasi-triangular shape on the deviatoric plane. The plastic surface is not constant. Its evolution is governed by the equivalent plastic strain $\varepsilon_{eqp}$:

$$\dot{\varepsilon}_{eqp} = \min\left(\dot{\varepsilon}_{p1}, \dot{\varepsilon}_{p2}, \dot{\varepsilon}_{p3}\right) \tag{9}$$

where $\varepsilon_{pi}$ is the i-th component of the principal plastic strains. The hardening is modeled by adjusting the compressive strength $\hat{f}_{c}$, while the softening is controlled through the parameter $c$:

hardening $\varepsilon_{eqp} \in \langle -\varepsilon_{cp}; 0 \rangle$ $\varepsilon_{eqp} = \langle -\varepsilon_{cp}; 0 \rangle$

$$\hat{f}_c(\varepsilon_{eqp}) = f_{c0} + (f_c - f_{c0}) \sqrt{1 - \left( \frac{\varepsilon_{cp} - \varepsilon_{eqp}}{\varepsilon_{cp}} \right)^2} \tag{10}$$

softening $\varepsilon_{eqp} \in \langle -\infty; -\varepsilon_{cp} \rangle$

$$\begin{aligned} c &= \left( 1 - \frac{w_c}{w_{co}} \right)^2, \quad w_c \in \langle -w_d; 0 \rangle \\ c &= 0, \quad w_c \in (-\infty; w_d) \\ w_c &= (\varepsilon_{eqp} - \varepsilon_{cp}) L_c^{'} \end{aligned} \tag{11}$$

In the above formulas, $\varepsilon_{cp}$ - plastic strain when the compression strength $f_c$ is reached in a uni-axial compression test, $f_{c0}$ - onset of nonlinear behavior in compression, $w_d$ is the critical value of compressive displacement and $w_c$ - displacement of the compressive plastic zone, $L_c^{'}$, - crush band, as will be described in Section 4.2.1.4.

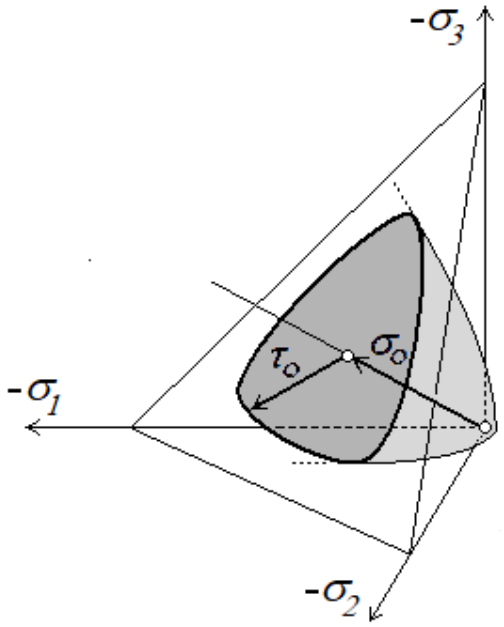


Fig. 3. Concrete failure criterion due to Menetrey&Willam 1995 [44].

The plastic flow in compression is not associated (normal) to the plastic function in Eq.(8), see Fig. 4. It is controlled by the parameter $\beta$, which defines a volumetric change during the crushing process:

$$G_p(\bar{\sigma}) = \beta \frac{1}{\sqrt{3}} I_1 + \sqrt{2J_2} \tag{12}$$

In which $\beta > 0$ - volume expansion, $\beta < 0$ - material compaction, $\beta = 1$ - material volume is preserved, $I_1$ - 1st invariant of stress vector and $J_2$ - 2nd invariant of deviatoric stress vector.

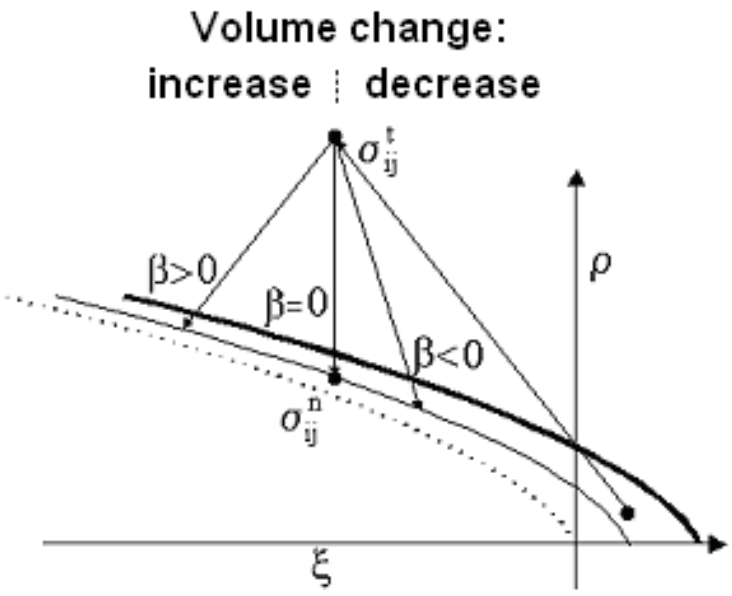


Fig. 4. Non-associated plastic flow in compression.

#### 4.2.1.2 Tensile failure model - smeared cracks

In an uncracked state the concrete is represented by homogenous isotropic body. A crack forms when the tensile strength is reached in principal tension. In the smeared crack concept, a discrete crack is represented by a damage due to a family of parallel cracks distributed in a volume associated with a material point and introduces an orthotropy in the material. The considered stress and strain variables are indicated in Fig. 5. (For clarity a 2D plane stress state is illustrated, but an extension to a crack in 3D is straightforward). Two crack models can be considered:

(a) Fixed crack model. The material axes of orthotropy $m_1$ and $m_2$ are fixed in the subsequent nonlinear analysis, whereas the weak axis $m_1$ is normal to the crack direction and the strong axis $m_2$ is parallel with the crack direction. Under a general loading, the axes of principal stress $\sigma_1$, strain $\varepsilon_1$, and orthotropy $m_1$ do not coincide and a shear stress $\tau^{cr}$ can occur on the crack face. Angle $\alpha$ is a normal to the crack direction and $\phi$ is a deviation of the maximal principal strain form the crack normal.

(b) Rotated crack model. The material axis $m_1$ is coincident with the principal tension $\varepsilon_1$, and with the principal stress $\sigma_1$, and angle $\varphi=0$. Thus, when the principal stress axes rotate, so does the crack direction. No shear stress occurs on the crack face, $\tau^{cr} = 0$.

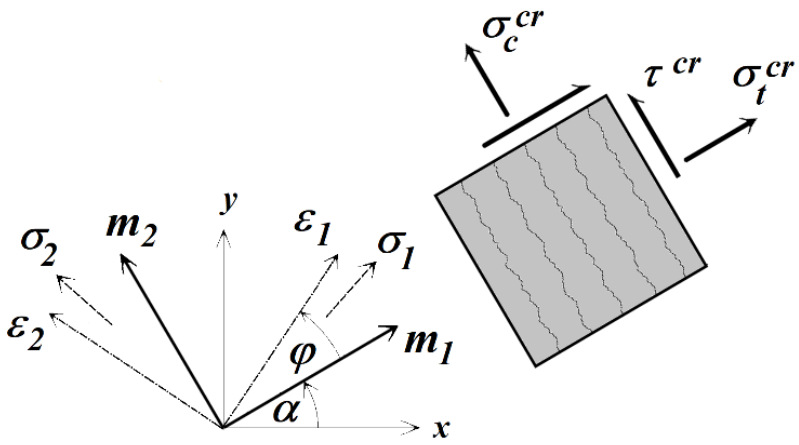


Fig. 5. Cracked concrete stress and strain state in 2D situation.

The two crack models can be combined in a mixed formulation, in which the crack direction rotates until a threshold softening stress $\sigma = c_{fix} f_t$ is reached and is fixed thereafter. This model reflects the experimentally observed crack formation process (e.g. Bazant , where the threshold softening stress reflects a transition from a fracture process zone to a localized crack.

#### 4.2.1.3 Softening laws of cracked concrete

The cohesive stresses on the crack face gradually decrease with growing deformation, which is referred as strain softening and is controlled by a nonlinear fracture mechanics. The following fracture types are recognized in cracked concrete: (1) Crack opening (mode I fracture); (2) Crack sliding in shear (mode II and III fracture); (3) Compressive failure (crushing fracture) due to the normal stress parallel with the crack direction.

(1) Crack opening law by Hordijk (1991) [36] describes the softening of the cohesive crack model, Fig. 6:

$$\frac{\sigma_t^{cr}}{f_t} = [1 + (c_1 \frac{w_t}{w_{to}})^3] \ exp(-c_2 \frac{w_t}{w_{to}}) \ - \frac{w_t}{w_{to}}(1 + {c_1}^3) \ exp(-c_2) \qquad (13)$$

$$w_{to} = 5.14 \frac{G_f}{f_t}, \ c_1 = 3, \ c_2 = 6.93.$$

where $w_t$ – crack width, $\sigma_t^{cr}$ – tensile stress on the crack face.

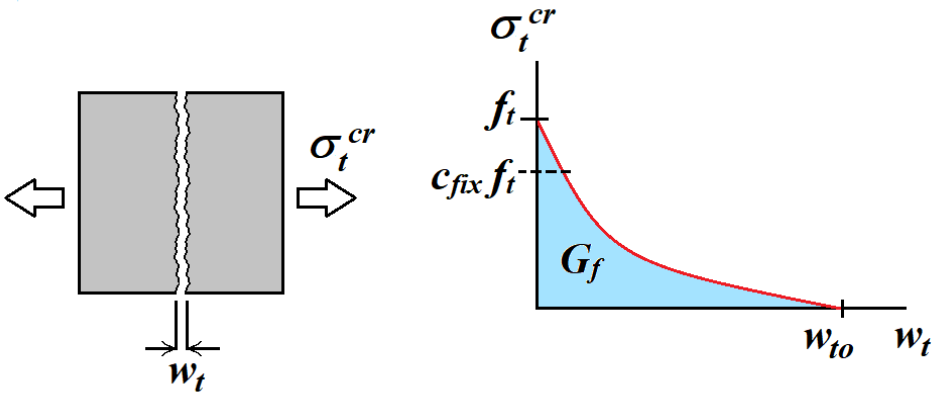


Fig. 6. Crack opening law.

(2) The shear stress in cracked concrete is described by models for stiffness and strength:

$$\tau^{cr} = s_F K_n^{cr} \gamma^{cr}, K_n^{cr} = \frac{\sigma_t^{cr}}{\varepsilon_t^{cr}} \tag{14}$$

$$\tau^{cr} \le \frac{0.18\sqrt{f_c}}{0.31 + \frac{24 w_t}{a_g + 16}} \tag{15}$$

The shear stress in Eq.(14) is based on the shear retention idea, in which $s_F$ – shear factor, $K_n^{cr}$ - stiffness of the crack opening. (For the typical value $s_F$=20 the model gives the fracture energy in shear $G_F^{II} \sim 10\, G_F^I$ , where $G_F^{II}$ and $G_F^I$ are the fracture energy is shear sliding and crack opening, respectively.)

(3) Compressive strength is reduced due to cracking damage:

$$\sigma_c^{cr} = r_c f_c \quad , \quad r_c = \frac{1}{0.8 + 170\,\varepsilon_t^{cr}}, \; r_c^{\lim} \le r_c \le 1.0 \tag{16}$$

Where $r_c$ – strength reduction factor and $r_c^{\lim}$ - limit of the reduction. Equations (15) and (16) are according to MCFT proposed by Collins (Bentz at al. 2006) [5].

A typical stress-strain curve resulting from the above formulations including tension and compression is illustrated in Fig. 7. The formulation also allows more complex forms, such as including a tension hardening, etc.

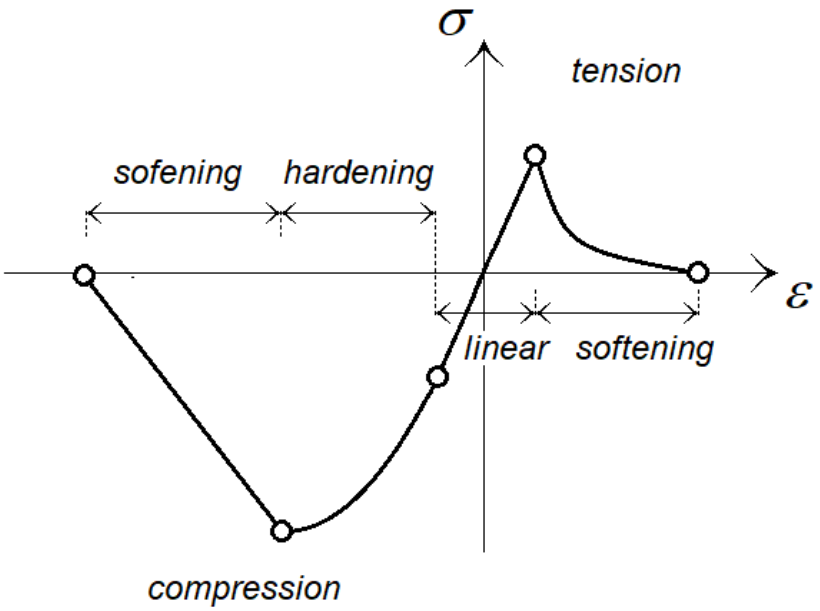


Fig. 7. Stress-strain curve of concrete.

#### 4.2.1.4 Strain localization limiter

The characteristic feature of concrete failure is the stress softening in the post-peak loading range, which is the source of instability and must be appropriately treated. In the smeared crack model this leads to a zero-energy dissipation when the constitutive model is based on strains. To remedy this malfunction an approach based on the localization limiter was introduced. The crack band concept was proposed by Bazant (1983) [3], and extended by Cervenka (1995) [11]. The idea of crack band is illustrated in Fig. 8. It is assumed, that the failure energy dissipates within the volume described by localization bands. The band size is defined as a projection of the element size. In tension:

$$w_t = \varepsilon_f L_t' \quad , \quad L_t' = \alpha\gamma L_t \ , \ \gamma = 1 + (\gamma_{\max} - 1)\frac{\theta_1}{45} \ , \ \theta_1 \in \langle 0;45 \rangle \ , \ \gamma_{\max} = 1.5 \tag{17}$$

where $w$ – displacement of the crack band (crack width), $\varepsilon_f$ – fracture strain, $L_t'$ - crack band size, and $L_t$ - element size projection parallel to the crack plane. $\alpha, \gamma$ are parameters for element type (Slobe 2013) and crack orientation angle $\theta$, respectively. The orientation factor $\gamma$ reflects the fact that for inclined cracks the crack band size exceeds the row of single elements as illustrated in Fig. 9. (For a low order iso-parametric quadrilateral element with full integration $\alpha = 1$, for the crack direction aligned with element sides $\gamma = 1$.)

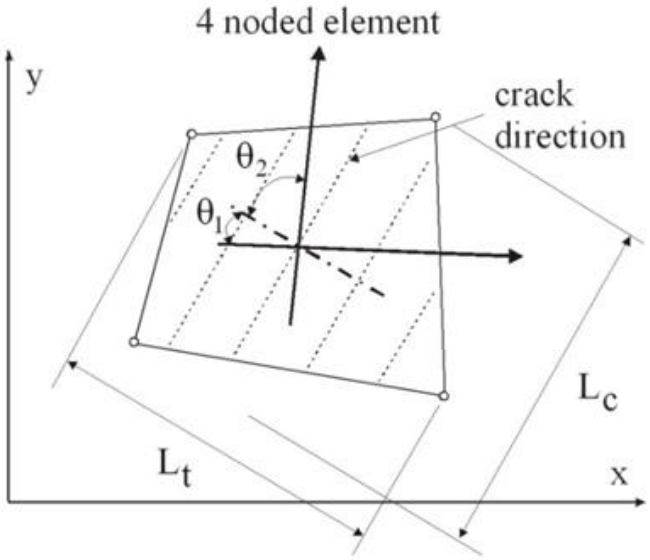


Fig. 8. Crack and crush bands in 2D situation.

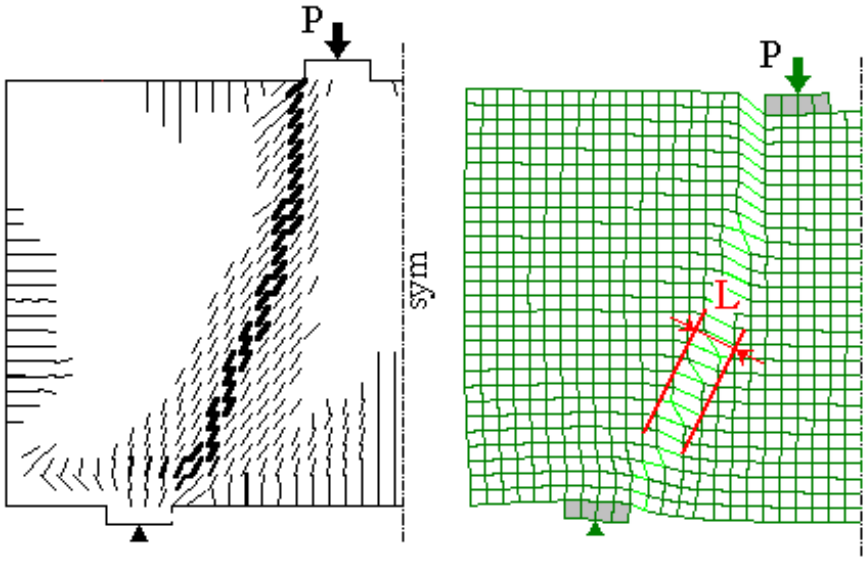


Fig. 9. Crack band for cracks not parallel with the finite element edges.

In analogy, for compression a similar model is used, Červenka J. (2014):

$$w_c = \varepsilon_p L_c' \quad , L_c' = \alpha\gamma L_c \ , \ \gamma = 1 + (\gamma_{\max} - 1)\frac{\theta_2}{45} \ , \ \theta_2 \in \langle 0;45 \rangle \tag{18}$$

where $w_d$ – displacement of the compression band, $\varepsilon_p$ – plastic strain, $L_c'$ - crush band size, and $L_c$ - element size projection normal to the crack plane.

Cervenka (2018) [15] demonstrated the limits of crack band approach for modelling of reinforced concrete when very large or very small finite elements are used.

The problem is demonstrated by experiments of cracked reinforced concrete panels, Cervenka and Gerstle (1971) [9], see Fig. 10. When very large finite elements are used, a standard assumption that a single crack will develop inside the finite element, is not valid anymore. Due to a variety of reasons, but namely due to the reinforcement, the cracks localize at distances smaller than the used finite element size. For very small element finite elements the crack spacing cannot be smaller than a non-homogeneity of material.

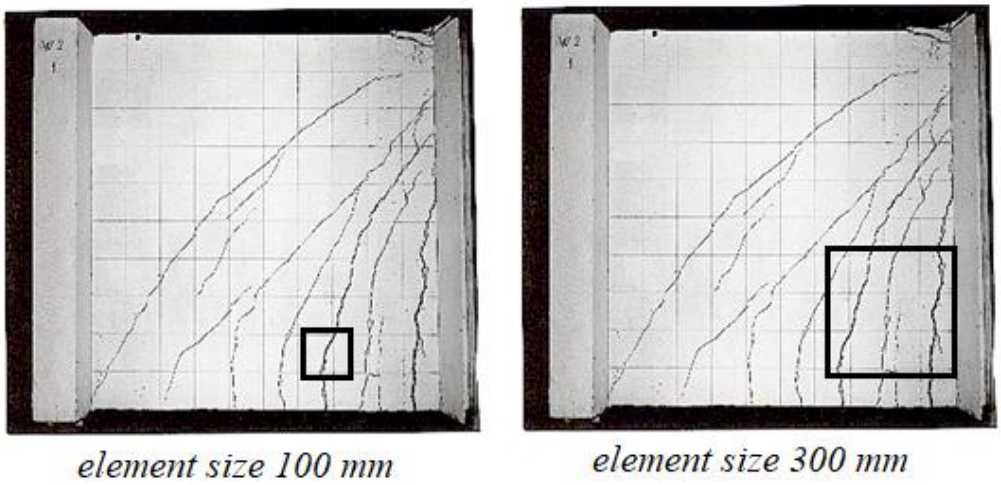


Fig. 10. Crack spacing in small and large finite elements, Cervenka&Gerstle 1971 [9].

A similar defective behavior can be observed in compression, as illustrated on the case of cylinder test subjected to compression in Fig. 11. In compression, namely due to the effects of the dilatancy and hardening, the plastic deformation may involve many elements. In this case a crush band size should be defined by the plastic deformation zone, which is approximately given by the smallest structural dimension such as the thickness.

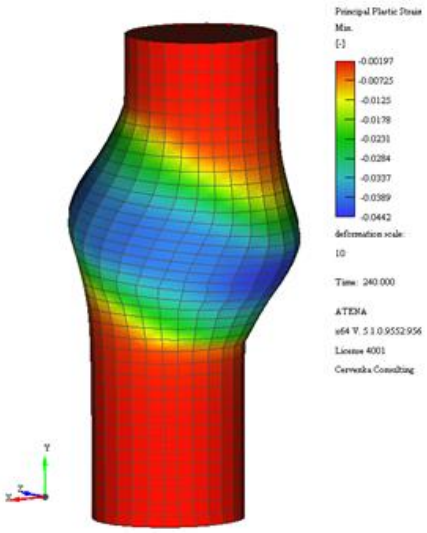


Fig. 11. Strain localization in compressive cylinder test.

In summary three localization limiters are proposed (Červenka j. 2018):

$L_{t,\min}$ - minimal crack spacing limiter in tension related to aggregate size

$L_{t,\max}$ - maximal crack spacing limiter in tension related to reinforcement arrangement

$L_{c,\min}$ - minimal crush band limiter in compression related to the minimal size of the compression zone.

#### 4.2.1.5 Crack width simulation based on the smeared crack model

The smeared crack approach simulates a real discrete crack indirectly by a damage distributed within a crack band. This simplifies the numerical analysis based on the finite elements but implies a question about the crack width. This problem was recently investigated by the authors in Cervenka et al. (2022) [21]. The study included 18 beam specimens tested at Vilnius Tech and covered a range of reinforcing arrangements and reinforcement types. Only the group of five beams representing typical RC beams is described here for brevity. The beam geometry and cross sections of 5 beams are shown in Fig. 12. The beams were subjected to a four-point bending scheme, where central span of 1000 mm was under a constant moment loading. The specimen series covers two reinforcing ratios, $\rho$ = 1, and 0.6, and bar profiles are 6, 14, 18, and 22 mm. The concrete cover of bars was 20 mm. The bars were in a single layer (except of the beam S 1-6). The concrete strength obtained in laboratory was in average $f_c$=49 MPa. The other parameters needed for the simulation were derived from the strength using the model code MC2010 relations as: elastic modulus E=35.5 GPa, tensile strength ft=3.6 MPa and fracture energy Gf=147 N/m. (The parameters slightly changed for each beam).

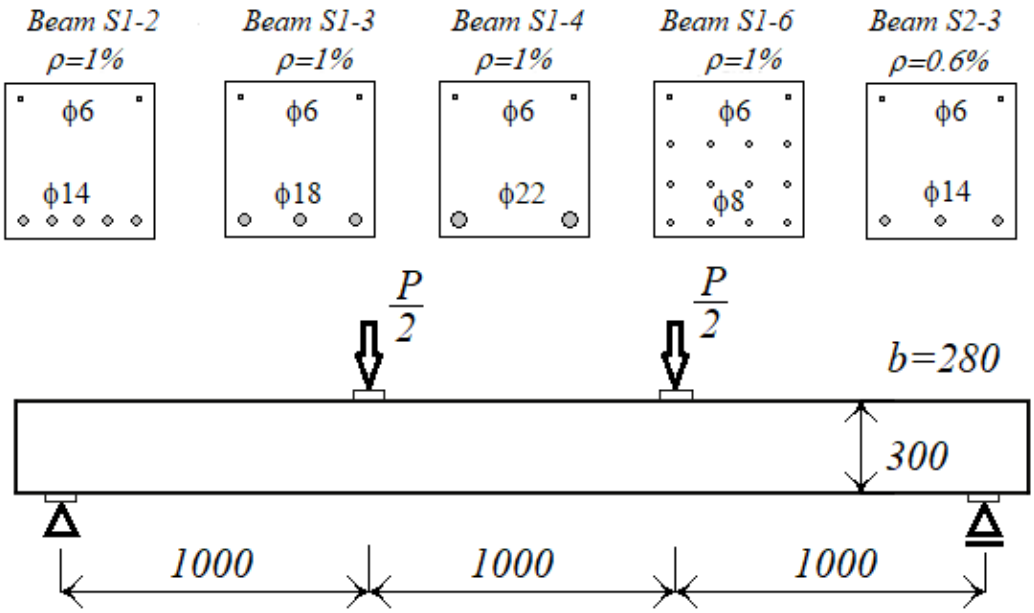


Fig. 12. Dimensions in mm and cross section reinforcement of beams.

The mean and maximum crack widths were investigated experimentally and numerically within the central span on the concrete surface along the reinforcement line. The simulation of the beam series was in 3D representation, with the smeared crack model according to the description above.

The model uncertainty of the crack width was defined as $\theta_w = w_{exp}/w_{sim}$ , where $w_{exp}$ and $w_{sim}$ are the crack widths from experiment and simulation, respectively. The summary of the model uncertainty of this series is shown in Table 1 for two mesh sizes M30, M15 (element sizes 30 and

15 mm, respectively). $\theta_{\mu}$ and $\theta_{max}$ are model uncertainties for mean and maximum crack width, respectively. The investigation revealed an excellent agreement between simulation and experiment for both meshes. (Note that $\theta$ =1 indicates a perfect match). Also the ratio between maximum and mean crack widths $w_{max}/w_{\mu}$ agrees well with the experiment. The low coefficient of variation confirms a strong validity of the conclusions.

Table 1: Crack width model uncertainty.

| | *Experiment* | | | *Simulation M30* | | | | *Simulation M15* | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Beam | $w_{\mu}$ | $w_{max}$ | $\frac{w_{max}}{w_{\mu}}$ | $w_{\mu}$ | $w_{max}$ | $\frac{w_{max}}{w_{\mu}}$ | $q_{w,mean}$ | $q_{w,max}$ | $w_{\mu}$ | $w_{max}$ | $\frac{w_{max}}{w_{\mu}}$ | $q_{w,mean}$ | $q_{w,max}$ |
| **S1-2** | 0.071 | 0.102 | 1.43 | 0.062 | 0.106 | 1.710 | 1.150 | 0.965 | 0.076 | 0.112 | 1.460 | 0.931 | 0.915 |
| **S1-3** | 0.085 | 0.140 | 1.64 | 0.075 | 0.142 | 1.899 | 1.145 | 0.987 | 0.071 | 0.152 | 2.137 | 1.199 | 0.919 |
| **S1-4** | 0.063 | 0.120 | 1.90 | 0.073 | 0.124 | 1.687 | 0.862 | 0.970 | 0.080 | 0.149 | 1.877 | 0.795 | 0.804 |
| **S1-6** | 0.038 | 0.060 | 1.57 | 0.048 | 0.070 | 1.456 | 0.789 | 0.852 | 0.042 | 0.064 | 1.535 | 0.919 | 0.941 |
| **S2-3** | 0.055 | 0.120 | 2.18 | 0.056 | 0.091 | 1.624 | 0.982 | 1.319 | 0.049 | 0.107 | 2.172 | 1.120 | 1.125 |
| **Average** | | | **1.745** | | | **1.675** | **0.985** | **1.019** | | | **1.836** | **0.993** | **0.941** |
| **COV** | | | **0.170** | | | **0.095** | **0.165** | **0.173** | | | **0.180** | **0.165** | **0.123** |

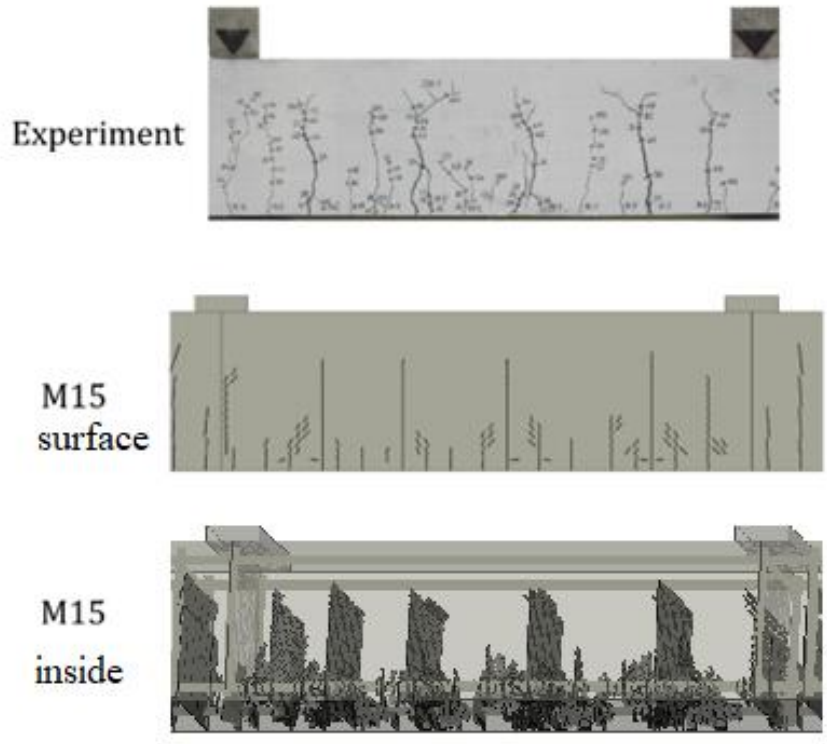


Fig. 13. Cracks in experiment and simulation.

The cracking analysis is illustrated for the beam S1-4 in Fig. 13. It compares the crack patterns on the surface between the experiment and simulation with mesh M15. It also shows the cracks inside the beam obtained by simulation, which is not available from experiments. It confirms the image of primary and secondary cracks. The primary cracks are the main open cracks extending

through the whole beam width. The secondary cracks are smaller inclined cracks near the reinforcement, which reflect the force transfer from the bars to concrete and form a bond mechanism. The research work in this project provided a conclusion, that the cracking analysis based on fracture mechanics is a sufficient model for cracking simulation and crack width prediction in reinforced concrete, and that it well describes a bond mechanism.

#### 4.2.1.6 Reinforcement, cables, contacts and other models

Reinforcement and prestressing cables are, of course, important components of the reinforced concrete structures. They are typically represented by truss elements in a uniaxial stress state and constitutive models are described by multilinear functions. They are embedded in the solid elements of concrete. The bond interaction between reinforcing bars and can be included by considering interface elements on bar surface. Interaction between solid objects can be also modelled by including contact elements reflecting the surface movements.

The issues requiring address environmental effects, durability and time require to include the classes of solutions based on transport mechanics, creep, dynamics and others. They can be coupled with the presented model and offer multi-branch solutions.

These and other features are necessary for a complete successful simulation of concrete structures. However, for brevity they are not included in this document, and reader should consult the appropriate references.

## 4.3 Safety formats for nonlinear analysis of reinforced concrete

**7.1 Design condition**

The design condition is generally formulated as:

$$E_{\mathrm{d}} < R_{\mathrm{d}} \tag{19}$$

where $E_{\mathrm{d}}$, $R_{\mathrm{d}}$ represent the design values of load and resistance, respectively. They include the specified safety margins based on a probabilistic theory as proposed by Vrouwenvelder (2002) [55] and the JCSS Probalilistic Model Code (2001) [37]. In the present approach, for simplicity, the random effects of load and resistance are considered separately.

Two types of uncertainties are recognized in the reliability analysis, Swiler (2007) [52]. The first one, referred as “aleatory” uncertainty, is due to inherent random properties of material and dimensions given by the concrete technology and is referred in the further discussion as the material uncertainty. The second one, referred a “epistemic”, is reflecting the knowledge level of the background theoretical model, and is called as the model uncertainty.

The design condition in Eq. (19) in the standard design practice is applied to critical cross-sections. The inconsistency of this concept is well known because different assumptions are used for the calculation of the load effect of internal forces $E_d$ (obtained by a linear analysis) and, the cross-section resistance $R_d$ (nonlinear material model). In general, the section forces may change due to a stress redistribution during the non-linear response. Furthermore, the local safety checks do not reflect a reliability of the entire system. In a nonlinear analysis a nonlinear material response is implicitly reflected in Eq. (4). Therefore a local check of the condition in Eq.(19) is satisfied by definition. However, a global check is required. The load effect $E_{\mathrm{d}}$ in the equation (19) is considered at the global level (typically it represents a of the relevant load combination), and analogically the resistance $R_{\mathrm{d}}$ is the ultimate load level at failure for the given load combination imposed on the structure.

MC2010 [30] introduces four methods for the global assessment using nonlinear analysis, which differ in estimate of the random parameters of the ultimate resistance. In this study only two most prominent methods will be treated. i.e. the partial factor (PFM) (Section 7.11.3.4) and ECoV method (Section 7.11.3.3 of fib model code 2010). These two methods are expected to be included in the new version of Eurocodes, and therefore they will be treated in more detail. A full probabilistic approach on the other hand can be used as an exact reference solution.

### 7.2 Partial safety factor method (PFM)

This approach is most appealing for practicing engineers as it is a quite natural extension of the current design practice. The design resistance $R_d$ is obtained as:

$$R_{\mathrm{d}} = \frac{R\left(X_{\mathrm{d}}; a_{\mathrm{nom}}\right)}{\gamma_{\mathrm{Rd}}}, \quad X_{\mathrm{d}} = \frac{X_{\mathrm{k}}}{\gamma_{\mathrm{m}}} \tag{20}$$

Where,

$R\left(X_{\mathrm{d}}; a_{\mathrm{nom}}\right)$ - structural resistance by numerical simulation,

$\mathrm{X_d}$ - design value of the material property considering material and geometric uncertainties, but excluding the model uncertainty,

$X_k$ - characteristic value of material parameter,

$\gamma_{\mathrm{m}}$ - partial safety factor of material (for concrete, reinforcement, etc.) without model uncertainty,

$a_{\mathrm{nom}}$ - nominal value of the geometric property, i.e. for instance reinforcement depth or element dimension,

$\gamma_{\mathrm{Rd}}$ - partial safety factor for the model uncertainty of NLFEA (to be described later).

### 7.3 GFM - ECoV method - an estimate of the coefficient of variation

ECoV method originally proposed by Červenka (2008 [16], 2013 [17]) is a semi probabilistic approach assuming that the statistical distribution of resistance due to the variability of materials can be estimated by Eq.(21)

$$V_{\mathrm{m}} = \frac{1}{1.65} \ln\left(\frac{R_{\mathrm{m}}}{R_{\mathrm{k}}}\right), \tag{21}$$

where the coefficient of variation $V_m$ is estimated by two resistance values, the mean resistance $R_{\mathrm{m}}$ (based on the mean material parameters) and characteristic resistance $R_{\mathrm{k}}$. These two points describe a scatter of the resistance. The underlying assumption is that the statistical distribution of the resistance is according to a lognormal distribution:

The global safety factor for material uncertainty of the resistance can be calculated as:

$$\gamma_{\mathrm{m}} = \exp(\alpha_{\mathrm{R}} \beta V_{\mathrm{m}}) \cong \exp(3.04\, V_{\mathrm{m}}) \tag{22}$$

Typical values are, for sensitivity factor $\alpha_R$=0.8 and reliability index $\beta$=3.8 (50 years reference period). The design value of resistance is calculated as:

$$R_{\mathrm{d}} = \frac{R_{\mathrm{m}}}{\gamma_{\mathrm{m}} \gamma_{\mathrm{Rd}}} \tag{23}$$

Alternatively, depending on data availability, a total coefficient of variation of resistance $V_R$ can be estimated as:

$$V_{\mathrm{R}} = \sqrt{\sum_{i}^{n} V_i^2}\,, \tag{24}$$

Where $V_i$ stands for the coefficient of variation of the random effect *i* of material, geometry, model uncertainty, etc. Then, Eq. (22) and (23) read:

$$R_{\mathrm{d}} = \frac{R_{\mathrm{m}}}{\gamma_{\mathrm{R}}}\,,\quad \gamma_{\mathrm{R}} = \exp(\alpha_{\mathrm{R}} \beta V_{\mathrm{R}}) \tag{25}$$

The mean and characteristic values of resistance are estimated from two separate nonlinear analyses using mean and characteristic values of the input material parameters, respectively.

The method is quite general and a suitable reliability level can be chosen by β if required. Variety of failure models and the sensitivity to a random variation of the material parameters is adequately captured in most cases of practical relevance.

### 7.4 Model uncertainty

The evaluation of model uncertainty of the resistance based on NLFEA is critical for a robust and reliable design. The model uncertainty can be determined from the comparison of the model predictions to experimental data. In this respect it should be acknowledged that the obtained partial factors of model uncertainty will be valid only for the investigated material model or simulation software. The model uncertainty is usually defined as the ratio:

$$\theta = R_{\mathrm{exp}} / R_{\mathrm{sim}} \tag{26}$$

where $R_{\mathrm{exp}}$ and $R_{\mathrm{sim}}$ are the resistances obtained from experiment and numerical simulation, respectively. The model uncertainty covers the imperfection of the model (lack of knowledge). The best agreement between a model and an experiment is obtained when $\theta_w$ =1.

The experimental resistance is considered as a reference, i.e. true value. Therefore, to investigate pure model uncertainties, it is essential to reduce other effects such as aleatory uncertainties to minimum. Material properties of concrete are typically identified by the concrete compressive strength tested on accompanying concrete samples (e.g. cylinders). Other material parameters (elastic modulus, tensile strength, fracture energy, etc.) are usually determined indirectly by formulas available in codes or should be provided as guidelines for a particular model Furthermore, there is a random variability of material properties within the tested structure. These effects are therefore included in the model uncertainties.

The experimental data base for the calibration of model uncertainty for a particular material model or modelling approach should include results of experiments relevant to the considered resistance model. Parameters of experiments, such as reinforcement, size or concrete strength class, determine the relevant range of validity. Failure mode is often also suggested as a classification parameter. However, it may be useful to include more failure modes in one group, since in general a failure mode identification is not unique and straightforward.

For a database containing *n* samples (experiments) the central moment characteristics of model uncertainty can be estimated for mean $\mu_\theta$, standard deviation $\sigma_\theta$ and coefficient of variation $V_\theta$. Considering a log-normal statistical distribution (according to *fib* MC2010) a safety factor for model uncertainty can be obtained as:

$$\gamma_{Rd} = \frac{\exp(\alpha_R \beta \times V_\theta)}{\mu_\theta}\,, \tag{27}$$

The reliability parameters should be in accordance with those applied for the material safety in equation (22).

A validation of the model uncertainty is by its nature dependent on the used constitutive model, its implementation in a particular software or even on the analyst or engineer himself. The human factor can be eliminated or at least addressed by providing modelling guidelines, training and education. Several investigations of model uncertainty for various nonlinear finite element software have been published recently.

Engen (2017) [26] investigated 38 RC members under monotone loading analyzed by several authors. Rather low partial safety factor for the model uncertainty was proposed based on the failure mode in the range 1.02-1.04.

Castaldo (2018) [7] considered 25 structural members from various literature sources including deep beams, shear panels and walls. The investigated members had statically determined static scheme and were tested up to failure with a monotonic incremental loading process. The tests were reproduced by non-linear analysis adopting 9 different modelling hypotheses distinguishing between the software platform and concrete tensile response. A total number of 225 simulations has been performed, and the resulting value of the model uncertainty partial factor was 1.15.

Castaldo (2020) [8] investigated also the model uncertainty for cyclic loading of 17 shear walls with statically determined scheme. 18 different modelling hypothesis were considered and altogether 306 simulations have been performed. This study proposes a model uncertainty factor $\gamma_{Rd}$ = 1.35.

Gino (2021) [31] focused on the problem of model uncertainty related to nonlinear analysis of slender RC members, A total number of 40 experiments of concrete columns with slenderness ratio between 15-275 are considered, and the model uncertainty factor $\gamma_{Rd}$ is proposed in the range 1.15 – 1.19.

Finally the authors of this paper also performed a study in Červenka V. (2018b) [18] including 33 RC members, slabs and beams, with failure modes ranging from ductile up to brittle failure modes. The results of this study are summarized in **Error! Reference source not found.**.

It should be noted that the model uncertainties parameters in **Error! Reference source not f ound.** are valid only for the used software (Cervenka et al, 2020) [20] and the constitutive model (Cervenka & Papanikolaou 2008) [13].

# 5 Programming Manual

This chapter describes how to define and simulate **New-MRCS column-beam connections** using the **Python programming interface** provided in **ATENA 2025**. The scripting interface allows for flexible model generation, parametric studies, automation of load scenarios, and integration with external data sources (e.g., optimization, machine learning, or validation frameworks).

The ATENA 2025 Python interface provides a powerful scripting environment for defining, analyzing, and evaluating complex hybrid systems such as the New-MRCS column-beam connection. This programmatic access enables full automation of simulation workflows, customization of reinforcement layouts, and systematic exploration of sustainable structural design options.

The New-MRCS system is characterized by:

- **Multi-spiral Reinforced Concrete (MRC) columns**
- **Steel (S) beams**
- Prefabricated assembly logic
- Green concrete and SCM integration

This section demonstrates how to:

- Create geometry and material definitions
- Assign reinforcement layouts
- Configure loading and boundary conditions
- Run the simulation and extract key outputs

---

## 5.1 Setting Up the Environment

Before scripting, ensure:

- ATENA 2025 Python interface is installed
- The atena2025 Python package is available (via ATENA installer)
- Your working environment is configured (e.g., VSCode, PyCharm, or a Jupyter notebook)

## 5.2 XML part

The user interface is created using XML syntax. For example, it is possible to read a real number from the user via RealParameter, an integer via IntegerParameter, more values via EnumParameter etc. In addition, it is possible to supplement the dialog with comments via CommentParameter. What such a notation looks like is shown in the image below.

```
<CommentParameter TabSection="Geometry" Name="BaseGeometry_Comment" Comment="Base dimensions:" FontWeight="bold"
<RealParameter TabSection="Geometry" Label="Column side a" Name="a" Value="600" Unit="mm" LimitRange="[50; 5000]"
<RealParameter TabSection="Geometry" Label="Height of concrete part v1" Name="v1" Value="1000" Unit="mm" LimitRan
<RealParameter TabSection="Geometry" Label="Height of steel part v2" Name="v2" Value="400" Unit="mm" LimitRange="

<CommentParameter TabSection="Geometry" Name="BeamGeometry_Comment" Comment="Beam dimensions:" FontWeight="bold"
<RealParameter TabSection="Geometry" Label="Beam length L" Name="L" Value="3505" Unit="mm" LimitRange="[100; 1000
<RealParameter TabSection="Geometry" Label="Beam width b" Name="b" Value="240" Unit="mm" LimitRange="[50; 5000]"
<RealParameter TabSection="Geometry" Label="Beam height h" Name="h" Value="800" Unit="mm" LimitRange="[50; 5000]"
<RealParameter TabSection="Geometry" Label="Thickness of flange t_f" Name="tf" Value="55" Unit="mm" LimitRange="[
<RealParameter TabSection="Geometry" Label="Thickness of web t_w" Name="tw" Value="20" Unit="mm" LimitRange="[1;
```

Fig. X: Sample XML code to define the user interface

## 5.3 Python part

User input processing takes place using a python functions written in the same file, but outside the xml part. First, we need to read the user input and create global parameters. The parameters from the sample above are read in this way:

```
a = pluginProxy.Parameters("a").GetValue()
v1 = pluginProxy.Parameters("v1").GetValue()
v2 = pluginProxy.Parameters("v2").GetValue()
L = pluginProxy.Parameters("L").GetValue()
b = pluginProxy.Parameters("b").GetValue()
h = pluginProxy.Parameters("h").GetValue()
tf = pluginProxy.Parameters("tf").GetValue()
tw = pluginProxy.Parameters("tw").GetValue()

globalParameter(equation = "a = " + str(a))
globalParameter(equation = "v1 = " + str(v1))
globalParameter(equation = "v2 = " + str(v2))
globalParameter(equation = "L = " + str(L))
globalParameter(equation = "b = " + str(b))
globalParameter(equation = "h = " + str(h))
globalParameter(equation = "tf = " + str(tf))
globalParameter(equation = "tw = " + str(tw))
```

Fig. X: Loading global parameters

After creating global parameters, you can start creating geometry using scripting commands. These commands are created automatically when creating geometry in the preprocessor, so you don't need to remember them, just copy them from the scripting history.

The image below shows an simple example of creating a surface in the Preprocessor with automatic script generation. This script can be copied and replaced with numeric values for global parameters. and replace the specific values with global parameters.

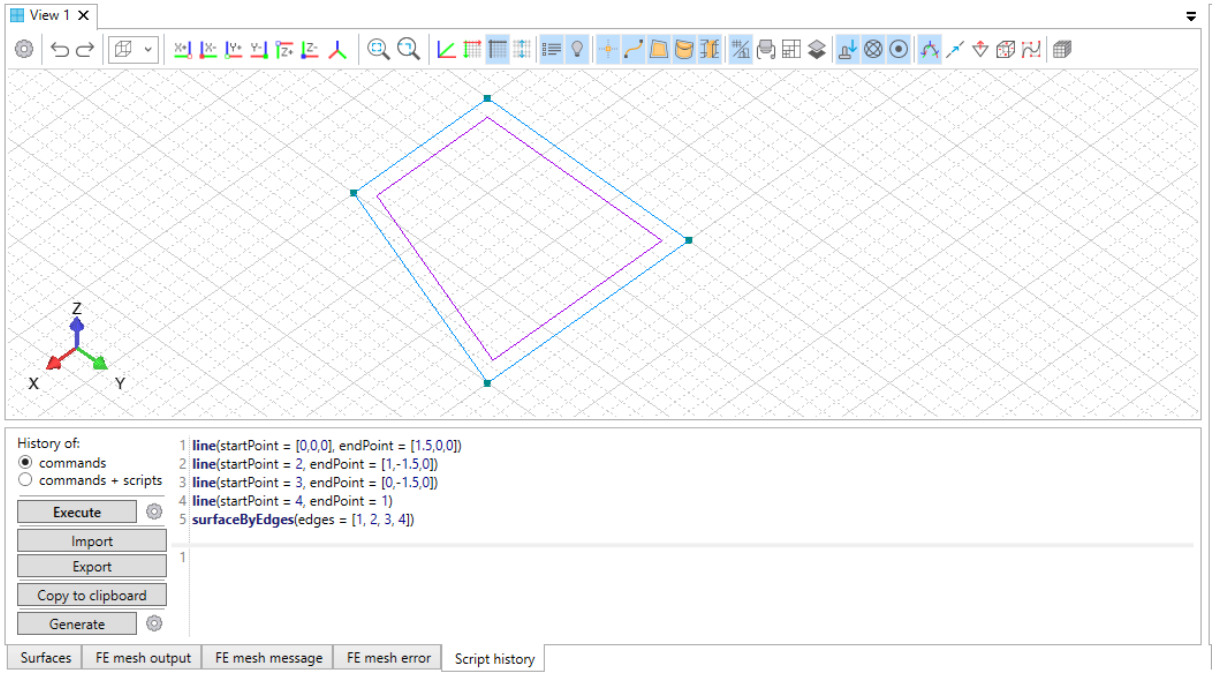


Fig. X: Automatic script history generation

Below you can see part of the script from our example, complete with global parameters.

```
layer(name = "Steel cross")
line(startPoint = ["b/2","-b/2",0], endPoint = ["b/2","-tw/2",0])
line(startPoint = 2, endPoint = ["b/2","tw/2",0])
line(startPoint = 3, endPoint = ["b/2","b/2",0])
line(startPoint = 4, endPoint = ["b/2","b/2","tf"])
line(startPoint = 5, endPoint = ["b/2","tw/2","tf"])
line(startPoint = 6, endPoint = ["b/2","-tw/2","tf"])
line(startPoint = 7, endPoint = ["b/2","-b/2","tf"])
line(startPoint = 8, endPoint = 1)
line(startPoint = 7, endPoint = 2)
line(startPoint = 6, endPoint = 3)
surfaceByEdges(edges = [1, 7, 8, 9])
surfaceByEdges(edges = [6, 2, 9, 10])
surfaceByEdges(edges = [3, 4, 5, 10])
```

Fig. X: Script for creating geometry with global parameters

In this way, it is possible to create all the necessary geometric data. It is not surprising that the script is also generated when creating materials, element types, FE mesh parameters and completely everything that we do in the preprocessor. Similarly, we can create a material, element type and assign it to the geometry.

```
material(name = "Steel", prototype = "BiLinearSteelVonMises", generator = "EuroCode 2")
materialGeneratorRun(material = "Steel")
materialSetName(name = "Steel", newName = "FBP")
elementType(name = "Plate", type = "PlateType", material = "Beam")
elementTypeEdit(name = "Plate", parameter = "AutomaticName", value = "False")
elementTypeEdit(name = "Plate", parameter = "Name", value = "FBP")
elementTypeMaterialSet(name = "FBP", material = "FBP")
elementTypeToGeometryAssign(name = "FBP", solids = [80, 81, 82, 83, 84, 85, 86, 87])
```

Fig. X: Creating a material and element type and assigning it to geometry

Each script command returns a created object that can be used if needed. The example below shows the connection of individual parts of a spiral reinforcement using the curvesCompose command into a single polycurve. The returned object is stored in the spiral variable (see highlighted part) and is later used to assign an element type with a material to this geometry

```
layer(name = "Central spiral")
circ1 = circularArc(method = "StartInteriorEnd", startPoint = ["0","-r2",-v1], interiorPoi
circ2 = circularArc(method = "StartInteriorEnd", startPoint = circ1[1].Id, interiorPoint =
n = int((2*v1+h)//s2)
translate(startPoint = circ1[2].Id, endPoint = circ2[1].Id, copies = n, collapse = "True",
spiral = curvesCompose(curves = list(range(circ1[0].Id, circ1[0].Id + 2*(n+1))))
elementType(name = "Bar", type = "BarType", material = "Reinforcement")
elementTypeEdit(name = "Bar", parameter = "ReBondType", value = "Bond")
elementTypeEdit(name = "Bar", parameter = "AutomaticName", value = "False")
elementTypeEdit(name = "Bar", parameter = "Name", value = "Central spiral")
elementTypeEdit(name = "Central spiral", parameter = "ProfileDiameter", value = d2)
elementTypeEdit(name = "Central spiral", parameter = "ReBondType", value = "PerfectBond")
elementTypeToGeometryAssign(name = "Central spiral", curves = [spiral.Id])
```

Fig. X: Saving the return object into a variable.

Since this is a regular python script, we can use standard tools of the programming language. In addition to storing in variables, this includes frequent program branching using if-else blocks or using loops. The for-loop was also used to create many intervals within the Task dialog - see below.

```
task(name = "Task")
interval(task = "Task", interval = "Interval 1")
intervalParametersSet(task = "Task", interval = "Interval 1", parameters = "Newton-Raphson - Optimal")
intervalEdit(task = "Task", interval = "Interval 1", parameter = "Name", value = "Interval - Force load
intervalLoadCasesAdd(task = "Task", interval = "Interval - Force load", loadCases = "Supports")
intervalLoadCasesAdd(task = "Task", interval = "Interval - Force load", loadCases = "Forces")

n = 0
oldDrift = 0
intervalCount = pluginProxy.Parameters("Intervals").Columns[0].GridValues.Count
for i in range(intervalCount):
  drift = pluginProxy.Parameters("Intervals").Columns[1].GridValues[i].Value
  steps = pluginProxy.Parameters("Intervals").Columns[3].GridValues[i].Value

  n += 1
  interval(task = "Task", interval = f"Interval {n}")
  intervalParametersSet(task = "Task", interval = f"Interval {n}", parameters = "Newton-Raphson - Optim
  intervalEdit(task = "Task", interval = f"Interval {n}", parameter = "IntervalMultiplier", value = dri
  intervalEdit(task = "Task", interval = f"Interval {n}", parameter = "NumberOfSteps", value = steps)
  intervalLoadCasesAdd(task = "Task", interval = f"Interval {n}", loadCases = "Supports")
  intervalLoadCasesAdd(task = "Task", interval = f"Interval {n}", loadCases = "Prescribed deformation")

  n += 1
  interval(task = "Task", interval = f"Interval {n}")
  intervalParametersSet(task = "Task", interval = f"Interval {n}", parameters = "Newton-Raphson - Optim
  intervalEdit(task = "Task", interval = f"Interval {n}", parameter = "IntervalMultiplier", value = -2*
  intervalEdit(task = "Task", interval = f"Interval {n}", parameter = "NumberOfSteps", value = 10)
  intervalLoadCasesAdd(task = "Task", interval = f"Interval {n}", loadCases = "Supports")
  intervalLoadCasesAdd(task = "Task", interval = f"Interval {n}", loadCases = "Prescribed deformation")
```

Fig. X: For loop for creating intervals.

The described method allows you to create user plugins. They are loaded into the preprocessor automatically if you place them in the "Plugins" directory in the AppData Roaming folder. The complete path to the folder may look like this:

C:\Users\Mike\AppData\Roaming\Cervenka_Consulting\AtenaStudio\Preprocessor_2025\MyTemplates\Plugins

## 5.4 Programming of Material Models

### 5.4.1 Concrete Material with Variable Properties – class header

This is the material class allowing the definition of a general material with time dependent material properties. It is derived from CCMaterial class. Any material can be used as a base Material, such as for instance CC3DNonLinCementitious2 for time dependent concrete modelling.

```
#ifndef CC_MATERIAL_WITH_VAR_PROPERTIES_DEFINED
#define CC_MATERIAL_WITH_VAR_PROPERTIES_DEFINED

// >>>>>>>>>>>>>>>>>>>>>>>>>>>> INCLUDE FILES <<<<<<<<<<<<<<<<<<<<<<<<<<<<

#include "CCUtils\iostream.h"
#include "CCMaterial\CCMaterial.h"
#include "CCFEModel\CCFEModel.h"

// >>>>>>>>>>>>>>>>>>>>>>>>>>>> PREPROCESSOR'S CONSTANTS <<<<<<<<<<<<<<<<<<

// >>>>>>>>>>>>>>>>>>>>>>>>>>>> CLASS TYPE DEFINITIONS <<<<<<<<<<<<<<<<<<<<

/**************************************************************************
 *                                                                        *
 *  Class: CCMaterialWithVariableProperties                               *
 *                                                                        *
 *  Purpose: This class can be used to create a material, whose selected  *
 *         material parameters will change according to a specified       *
 *         function depending on time or step number.                     *
 *                                                                        *
 *  Usage: This material is derived from the base material class.         *
 *                                                                        *
 **************************************************************************/

class CCMaterials_Basic_DECL_SPEC CCMaterialWithVariableProperties : public CCMaterial
{
```

```
public:

   enum VersionId {VERSION_ID=3};
   DECLARE_SERIAL(CCMaterialWithVariableProperties)

   CCMaterialWithVariableProperties();

   CCMaterialWithVariableProperties( const CCObject &src );
   CCMaterialWithVariableProperties( const CCMaterialWithVariableProperties &mat );

public:
  /*
   * Destructor
   */
   virtual ~CCMaterialWithVariableProperties ();

  /*
   * Other
   */

   virtual CCObject& operator = (const CCObject& src);

   virtual void write() const;
   virtual CCInt read();
   virtual void delete_contents();

   // Set of virtual function for changing material parameters
   // Returns CCTRUE if parameter was changed. Otherwise CCFALSE
   virtual CCBool change_parameter( CCString &name, CCReal&   value );
   virtual CCBool change_parameter( CCString &name, CCInt&    value );
   virtual CCBool change_parameter( CCString &name, CCString&  value );

   // Creates an appropriate material state, initializes it and
   // returns its pointer
   // !!! User must explicitly delete it !!!
   virtual CCMaterState* create_state() const;

   // Resets material state to initial values
   virtual void reset_state( CCMaterState &mstate ) const;

   // Compute tangent matrix
   virtual void tangent_stiff( CCMaterPoint &mp,
                               CCRealMatrix &d );
   // Compute secant matrix
   virtual void secant_stiff( CCMaterPoint &mp,
                              CCRealMatrix &d );
   // Compute tangent matrix
   virtual void tangent_compl( CCMaterPoint &mp,
                               CCRealMatrix &c );
   // Compute secant matrix
   virtual void secant_compl( CCMaterPoint &mp,
                              CCRealMatrix &c );
   // Update of state variables based on strain increment
   // current state (state is updated if requested)
   virtual const CCReal calculate_response( const CCRealColumnVector &deps,
                                            UpdateFlag update,
                                            CCMaterPoint &mp,
                                            CCRealColumnVector &sigma );
   // Transform state variables to account for large
   // deformations. For instance 2nd Piola Kirchhof stresses
   // to Cauchy stresses
   // (called typically after UPDATE_MATERIAL)
   virtual void transform_state( const CCRealMatrix& deformation_gradient,
                                 CCMaterPoint &mp ) const;

   // Compute elastic matrix
   virtual void calculate_elast_stiff();
   virtual void elast_stiff( CCMaterPoint &mp, CCRealMatrix &d );

   // Compute elastic matrix
   virtual void calculate_elast_compl();
   virtual void elast_compl( CCMaterPoint &mp, CCRealMatrix &d );

   // This method is called to register output data available
   // for this material law.
   virtual void register_output_data( LPCSTR label_extension = NULL ) const;

   // Get material weight density
   virtual CCReal get_weight_density( CCMaterPoint &mp ) const;
   // Get material young modulus density
   virtual CCReal get_young_modulus( CCMaterPoint &mp ) const;
   // Get material mu
   virtual CCReal get_mu( CCMaterPoint &mp ) const;
   // Get material expansion coeff
   virtual CCReal get_expansion_coeff( CCMaterPoint &mp ) const;
   // Get material idealisation type
   virtual CCInt get_material_ideal_type() const;
   // Get material fc (compressiv                                          e strength)
   virtual CCReal get_fc(CCMaterPoint &mp) const;
   // Get material ft (tensile strength)
   virtual CCReal get_ft(CCMaterPoint &mp) const;
   // Get material Gf (fracture energy)
   virtual CCReal get_Gf(CCMaterPoint &mp) const;
   // Get material damping alpha coeff
   virtual CCReal get_damping_alpha_coeff() const;
   // Get material damping_beta coeff
   virtual CCReal get_damping_beta_coeff() const;

   virtual void check( CCFEModel* model_ptr  );

   // dynamic type creation

   CCStatic CCObject* copy_constructor(const CCObject& src) ;
   virtual CopyConstructor get_copy_constructor_ptr() ;

// Info helpers
   virtual CCString description() const  ;
   virtual const CCInt type_of_items() const ;
   virtual CCString item_label(CCInt item) const ;
   virtual const CCInt item_type(CCInt item_id) const;
   virtual const CCInt data_format() const ;
   virtual const CCString item_units(CCInt item) const ;

   virtual void* item_ptr(CCInt item) const;
   virtual CCInt number_of_items() const;
```

```
// MFC stuff
   virtual void Serialize( CArchive& archive ); // override

#ifdef _DEBUG  // Functions for _DEBUG version only
   virtual void AssertValid() const; // override
   virtual void Dump( CDumpContext& dc ) const; // override
#endif

public:
   // Virtual function for setting material parameters
   // Returns true if parameter was successfully set
   virtual CCBool set_parameter( CCString &name, CCReal& value );
   virtual CCBool get_parameter(CCString &name, CCReal &value);
   CCReal get_value(CCInt ipar, CCFEModel *model_ptr=nullptr, CCMaterPoint *mp_ptr=nullptr) const;

   // Data members

   // The id of the base material, whose material properties can
   // be varied. This material utilizes the change_parameter
   // function of the base material, so it is important that
   // the base material is capable of changing its parameters.
   CCMaterialSmartId base_material_id;

   // Heap of material properties names, which will be varied from
   // step to step based on time or step number.
   CStringHeap param_names;

   // Parameter for LITS
   CCReal parameter_a;

   // Heap of function ids that define the evolution of each
   // material parameter, which is specified in the list of
   // param_names
   CCFunctionSmartIdHeap function_t_ids;
   CCFunctionSmartIdHeap function_temp_ids;
   CCRealHeap const_values;   // the final mater. parameter = const_values * function_ids(t)

   CCInt tmp_elem_total_temperature_data_id=-1; // temporary, used only in value(...)
   CCInt tmp_elem_temperature_incr_data_id=-1; // temporary, used only in value(...)
};

// >>>>>>>>>>>>>>>>>>>>>>>>>>>> INLINE FUNCTIONS <<<<<<<<<<<<<<<<<<<<<<<<<

// >>>>>>>>>>>>>>>>>>>>>>>>>>>> EXTERN VARIABLES AND FUNCTION <<<<<<<<<<<<

// >>>>>>>>>>>>>>>>>>>>>>>>>>>> END OF FILE <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
#endif // CC_MATERIAL_WITH_VAR_PROPERTIES_DEFINED
```

### 5.4.2 Variable Material Calculate Response Method

This is the base method controlling the evaluation of stresses based on given strain increment and a set of time dependent functions for the selected material parameters.

```
/*************************************************************************
 *                                                                       *
 *                                                                       *
 * Function: update material point state based on the increment of strain*
 *          and return material response in sigma. Depending on the      *
 *        value of variable update material point state variables.    *                                                                  *
 *                                                                       *
 * Return: returns performance index, which in this case is zero since   *
 *         function does nothing.                                        *
 *                                                                       *
 *************************************************************************/
const CCReal
CCMaterialWithVariableProperties::calculate_response
(
  /*
   * Arguments
   */
   const CCRealColumnVector &deps,   // strain increment
   CCMaterial::UpdateFlag upd_flag,  // CCTRUE is state data are to be
                                     // updated
   CCMaterPoint &mp,                 // Material point with state data
   CCRealColumnVector &sigma         // Calculated stress
)
{
   CCFEModel* model_ptr = FE_Model_Ptr.operator ->();
   CCInt n_param = param_names.GetSize();
   CCMaterial* mater_ptr = model_ptr->get_material_ptr( base_material_id );

   for( CCInt ipar=1; ipar<=n_param; ipar++ )
      {
      CCReal value = get_value(ipar, model_ptr, &mp);
      mater_ptr->change_parameter( param_names( ipar ), value );
      }

   return mater_ptr->calculate_response( deps, upd_flag, mp, sigma );
}
```

### 5.4.3 Reinforcement Cycling Bond Model

Cycling reinforcement bond model is simulated using an appropriate class derived from CCExternalCable_2D class for FE geometry.

```
// Calculates element nodal slips for a cable element. This routine is
// called for all elements (of the group) from the first element's calculate_element() routine.
void CCBarWithMemoryBond_2D::calculate_slips
   (
   CCElementGroup *element_group_ptr, // [in] current element group pointer
   CCGlobalVector &global_vector,     // [in] displs. within step (for calc. strain)
   CCExternalCableGeometry *geometry_ptr, // [in] ptr. to cable geometry description
   CCMaterial *default_material_ptr,          // [in] pointer to material
   CCFEModel *model_ptr_,               // [in] pointer to CCFEModel
   CCRealMObject*& deviators_data_ptr,    // [in,out] total and delta slip in deviators
   CCBool reset_dslips_at_end,         // [in] flag for reset delta slips (at 1st iteration)
   CCRealMObject &r_coord,            // [tmp] r coords in IPs
```

```
   CCRealMObject &dh_dr_value,        // [tmp] diff(nodal functions, r)
   CCRealMObject &dh_dx_value,        // [tmp] diff(nodal functions, x)
   CCRealMObject &coords_transf,      // [tmp] not used (no transformation global->local coord. system)
   CCRealMObject &dofs_transf,        // [tmp] not used (no transformation global->local coord. system)
   CCRealMObject &element_coords,     // [tmp] local element coordinates
   CCRealMObject &b_l,                // [tmp] B_l matrix
   CCRealMObject &deformation_gradient, // [tmp] matrix of def. gradient
   CCRealCVObject &elem_displs,       // [tmp] element displacements
   CCRealCVObject &nodal_vector,      // [tmp] temporary array (displs. in a node)
   CCRealCVObject &tau,               // [tmp] stress increment
   CCRealCVObject &eps,               // [tmp] strain increment
   CCRealCVObject &deps_ini,          // [tmp] initial strain increment
   CCRealHeap &init_dsig_deps,        // [tmp] temporary storage for initial strain
   CCReal lambda_prev                 // [in] current loading factor lambda (not used here, needed for external cable only)
   )
   {
   CCStructures *model_ptr = dynamic_cast<CCStructures *>(model_ptr_);
   ASSERT(model_ptr);

   // Deal with backward compatibility
   if(is_equal(geometry_ptr->bond_cohesion, -CCREAL_HUGE))
      {
      // ensure compatibility with CCExternalCableGeometry::VERSION_ID<=4
      geometry_ptr->bond_cohesion = geometry_ptr->deviator_friction_constant;
      geometry_ptr->deviator_friction_constant = 0.;
      }
   if(is_equal(geometry_ptr->slip_coeff_unload, CCREAL_HUGE))
      {
      // ensure compatibility with CCExternalCableGeometry::VERSION_ID<=5
      geometry_ptr->slip_coeff_unload = !is_zero(geometry_ptr->deviator_friction_constant) ?
         geometry_ptr->deviator_friction_coefficient/geometry_ptr->deviator_friction_constant                                    :
geometry_ptr->deviator_friction_coefficient;
      geometry_ptr->deviator_friction_coefficient = 0;
      }

   CCReal deviator_friction_coefficient_save =  geometry_ptr->deviator_friction_coefficient;
   CCReal slip_coeff_unload_save = geometry_ptr->slip_coeff_unload;
   CCReal deviator_friction_constant_save = geometry_ptr->deviator_friction_constant;
   CCReal bond_cohesion_save = geometry_ptr->bond_cohesion;
   CCReal wobble_coefficient_save = geometry_ptr->wobble_coefficient;

   if(geometry_ptr->release_cable)
      {
      // temporarily release fix of external cable in pre-stressing phase
      geometry_ptr->deviator_friction_coefficient = 0.;
      geometry_ptr->slip_coeff_unload = 0;
      geometry_ptr->deviator_friction_constant = 0.;
      geometry_ptr->bond_cohesion = 0;
      geometry_ptr->wobble_coefficient = 0;
      }

   // Calculate cable geometry
   CCObjectPtrHeap &group_data = (element_group_ptr->group_data);
   CCRealCVObject *element_lengths_ptr = NULL;
   allocate_object_ptr(group_data, "ExternalCableLengths", element_lengths_index,
      element_lengths_ptr, element_lengths_index);
   ASSERT(element_lengths_ptr);
   CCRealCVObject &element_lengths = *element_lengths_ptr;

   CCRealCVObject *deviator_angles_ptr = NULL;
   allocate_object_ptr(group_data, "ExternalCableAngles", deviator_angles_index,
      deviator_angles_ptr, deviator_angles_index);
   ASSERT(deviator_angles_ptr);
   CCRealCVObject &deviator_angles = *deviator_angles_ptr;

   CCReal deviator_radius_save = geometry_ptr->deviator_radius;
   geometry_ptr->calculate_cable_geometry(element_group_ptr, deviator_angles, element_lengths);


   //   CCBool init_mpoints;
   //   init_mpoints= (model_ptr->step_id == 1 && model_ptr->iteration_id ==1);
   CCFEModel::Switches &switches= model_ptr->switches;

   CCBool fix_the_bar;
   long number_of_elements=element_group_ptr->elements.GetSize();
   long number_of_slips=number_of_elements+1;
   CCString msg;
   CCString element_group_label= " in " + element_group_ptr->label(model_ptr);
   CCFixedSize<CCRealHeap, 3> xx, yy; xx.SetSize(3); yy.SetSize(3);

   if(allocate_resize_object_ptr(group_data,"ExternalCableData", deviators_data_index,
      number_of_slips, MINIMUM_SLIP, deviators_data_ptr, deviators_data_index))
      deviators_data_ptr->zero();
   ASSERT(deviators_data_ptr);
   CCRealMObject &deviators_data=*deviators_data_ptr;

   CCRealCVObject *slip_diagonal_ptr=NULL, *slip_super_diagonal_ptr=NULL,
      *slip_sub_diagonal_ptr=NULL, *slips_ptr=NULL;

   CCObjectPtrHeap &group_tmp_heap = CCElementGroup::get_threads_group_tmp_heap().get_container();

   get_object_ptr(group_tmp_heap,slip_diagonal_index, slip_diagonal_ptr);
   get_object_ptr(group_tmp_heap,slip_super_diagonal_index, slip_super_diagonal_ptr);
   get_object_ptr(group_tmp_heap,slip_sub_diagonal_index,slip_sub_diagonal_ptr);
   get_object_ptr(group_tmp_heap,slips_index,slips_ptr); // rhs or delta slips increment

   CCReal x1=0, y1=0;  // coords of the first node of the cable (in local coordinate system)
   CCReal xn=0, yn=0;  // coords of the current node of the cable, for which we calculate slip (in local coordinate system)
   // ccassert allocations
   ASSERT(slip_diagonal_ptr->length() == number_of_slips);
   ASSERT(slip_super_diagonal_ptr->length() == number_of_slips);
   ASSERT(slip_sub_diagonal_ptr->length() == number_of_slips);
   ASSERT(slips_ptr->length() == number_of_slips);
   ASSERT(deviators_data.columns() == 4); // slip increment, total slip
   ASSERT(deviators_data.rows() == number_of_slips); // the above data arranged by deviators

   CCExternalCablePrestress *step_related_prestress_ptr = NULL, *group_prestress_ptr = NULL;
   CCReal prestress = 0.;
   CCInt& prestress_incr_load_id = TLS_allocate_cell_CCInt(CCExternalCable_2D___calculate_slips___prestress_incr_load_id);
   CCInt& prestress_tot_load_id = TLS_allocate_cell_CCInt(CCExternalCable_2D___calculate_slips___prestress_tot_load_id);

   CCObjectPtrHeap &element_step_related_data = element_group_ptr->elements(number_of_elements).
      element_step_related_data;    // grabbing load from the last cable element!! even in the case, when the first element in
the cable is prestressed; see actualize_bcs(CCStructures.cpp)
```

```
   // Get element pre-stressing (increment !!)
   if((prestress_incr_load_id   =   element_step_related_data.FindClassF(RUNTIME_CLASS(CCExternalCablePrestress),   -1,   -1,
prestress_incr_load_id))
      > 0) // check if CCExternalCablePrestress is applied at all. if not, return
      {
      step_related_prestress_ptr = // CCExternalCablePrestress::prestress_incr object
         (CCExternalCablePrestress*)*(element_step_related_data(prestress_incr_load_id)); // ptr to step prestress increment
      }

   // Get element pre-stressing (total with respect to the previous step )
   CCInt load_id;
   if(allocate_object_ptr(group_data, "ExternalCablePrestress",
      load_id, group_prestress_ptr, prestress_tot_load_id))  // check if CCExternalCablePrestress is applied at all. if not,
return
      {
      prestress_tot_load_id = load_id; // save load index for the next search
      }
   ASSERT(group_prestress_ptr);

   if(group_prestress_ptr->reset_dslips)
      {
      group_prestress_ptr->prestress_tot += group_prestress_ptr->prestress_incr;
      group_prestress_ptr->prestress_incr = 0.;
      }

   if(step_related_prestress_ptr)
      {   // prestress total specified, increment specified
      group_prestress_ptr->prestress_incr =
         step_related_prestress_ptr->prestress_incr*lambda_prev;
      }
   else
      {    // prestress total specified, increment not specified
           // do nothing
      }

   prestress = group_prestress_ptr->prestress_tot+group_prestress_ptr->prestress_incr;

   CCReal e_a_stiff_left = 0, e_a_stiff_right = 0;

   // Check cable end nodes BCs

   CCUInt fixed_end_geometry_map = 0;
   if(geometry_ptr->fix_the_bar_left)   // BCs based on geometry
      set_bit(fixed_end_geometry_map, CCExternalCablePrestress::FIX_LEFT);
   if(geometry_ptr->fix_the_bar_right)
      set_bit(fixed_end_geometry_map, CCExternalCablePrestress::FIX_RIGHT);

   CCUInt fixed_end_total_load_map = 0;       // BCs based on previous (total) pre-stressing
   if(is_set_bit(group_prestress_ptr->flag, CCExternalCablePrestress::PRESTRESS_RIGHT))
      set_bit(fixed_end_total_load_map, CCExternalCablePrestress::PRESTRESS_RIGHT);   // = FIX_LEFT
   if(is_set_bit(group_prestress_ptr->flag, CCExternalCablePrestress::PRESTRESS_LEFT))
      set_bit(fixed_end_total_load_map, CCExternalCablePrestress::PRESTRESS_LEFT); // = FIX_RIGHT

   if( step_related_prestress_ptr)
      {
      CCUInt fixed_end_incr_load_map = 0;  // BCs based on current pre-stressing
      if(is_set_bit(step_related_prestress_ptr->flag, CCExternalCablePrestress::PRESTRESS_RIGHT))
         set_bit(fixed_end_incr_load_map, CCExternalCablePrestress::PRESTRESS_RIGHT); // = FIX_LEFT
      if(is_set_bit(step_related_prestress_ptr->flag, CCExternalCablePrestress::PRESTRESS_LEFT))
         set_bit(fixed_end_incr_load_map, CCExternalCablePrestress::PRESTRESS_LEFT); // = FIX_RIGHT

      if(fixed_end_incr_load_map==0)
         {
         fixed_end_incr_load_map = fixed_end_geometry_map; // prestress incr.LCs not defined; used those from geometry
         step_related_prestress_ptr->flag = fixed_end_geometry_map;
         }

      if(fixed_end_total_load_map==0)
         {
         fixed_end_total_load_map = fixed_end_incr_load_map; // prestress total (=previous) LCs not defined; used those from
prestress incr.LCs
         group_prestress_ptr->flag = step_related_prestress_ptr->flag;
         }

      if(fixed_end_total_load_map!=fixed_end_incr_load_map)
         throw new CCFEModelExc(THROW_EXC_PLACE, "CCFEModel", IDS_CCFEModelExc_INCOMPAT_PRESTRESS, 1,
            element_group_ptr->label(model_ptr));
      }
   else
      {
      if(fabs(group_prestress_ptr->prestress_tot+group_prestress_ptr->prestress_incr)>1.e-3)
         {
         set_bit(fixed_end_geometry_map, CCExternalCablePrestress::FIX_LEFT);
         set_bit(fixed_end_geometry_map, CCExternalCablePrestress::FIX_RIGHT);
         }
      }

   CCReal slip_displs_error=CCREAL_HUGE, slip_left, slip_right=0.;
   CCInt slip_iteration_id=1;

   CCReal element_force_left, element_force_right;
   CCReal cable_length_right=0, cable_length_left=0., total_cable_length=0.;
   CCReal cohesion_left=0, cohesion_right=0;
   CCReal wobble_cohesion_left = 0, wobble_cohesion_right = 0;
   CCReal variable_cohesion_left, variable_cohesion_right;
   CCReal reduction_factor_left, reduction_factor_right;
   CCReal temperature_left=0, temperature_right=0;

   CCReal fi;
   CCReal deviator_a, deviator_b, deviator_a_left, deviator_b_left,
      deviator_a_right, deviator_b_right;

   if(group_prestress_ptr->reset_dslips)
      {  // zeroize delta slips
      memset( &deviators_data(1,DELTA_SLIP), 0, deviators_data.rows()*sizeof(CCReal));
      group_prestress_ptr->reset_dslips = CCFALSE;
      }

   CCInt mater_prop_id=0;
   CCRealColumnVector *h_value_ptr =  (CCRealCVObject*) get_object_ptr(group_tmp_heap, h_value_index);
   ASSERT(h_value_index);
   CCReal area = geometry_ptr->get_area();

   while(slip_iteration_id <= switches.iteration_limit &&
      slip_displs_error > switches.displacement_error)
      {
```

```
      CCReal scoord = 0; // longitudinal s coordinates of prestress bas, (from start to end)

      fix_the_bar=CCTRUE; // assume the bar need fixing;

      if(is_set_bit(fixed_end_total_load_map, CCExternalCablePrestress::PRESTRESS_LEFT))
         {
         CCElement *element_ptr = &element_group_ptr->elements(1);
         reduction_factor_left = get_reduction_factor(element_ptr, geometry_ptr, group_tmp_heap);
         element_force_left = prestress*area*reduction_factor_left;
         }
      else
         {
         reduction_factor_left = 1;
         element_force_left = 0.;
         }

      cohesion_left=0.;
      wobble_cohesion_left = 0.;
      slip_left=deviators_data(1, DELTA_SLIP);

      temperature_right = model_ptr->base_temperature_ref;

      for(CCInt slip_id = 1; slip_id<= number_of_slips; slip_id++)
         {
         if(slip_id>1)
            scoord += element_lengths(slip_id-1);
         else
            scoord = 0;

         // solving node, i.e. slip_id, i.e. left node of element slement_id=slip_id

         if(slip_id< number_of_slips)
            {
            CCInt element_id = slip_id;

            // element exists, calculate its normal force

            CCElement *element_ptr = &(element_group_ptr->elements(element_id));
            CCMaterPoint *mater_point_ptr = element_ptr->element_state->mpoints(1);

            reduction_factor_right = get_reduction_factor(element_ptr, geometry_ptr, group_tmp_heap);
            CCReal reduced_area = area * reduction_factor_right;

             // Obtain reference to current material at element
            CCMaterial *material_ptr;
            if(get_mater_ip_prop_id(1, element_group_ptr, element_ptr, mater_prop_id))
               material_ptr = model_ptr->get_material_ptr( mater_prop_id );
            else
               material_ptr=default_material_ptr;
            // if(init_mpoints) element_ptr->set_mpoints( element_group_ptr, material_ptr);
            //         if(init_mpoints) create_element_state( element_group_ptr,
            //                                                element_ptr );
            // right element slip
            slip_right= deviators_data(slip_id+1, DELTA_SLIP);
            // calculate right element force
            element_force_right=calculate_cable_force
               ( element_group_ptr, element_ptr, global_vector,
               material_ptr, model_ptr, slip_left, slip_right,
               r_coord, dh_dr_value, dh_dx_value, coords_transf, dofs_transf,
               element_coords, b_l, deformation_gradient,
               elem_displs, nodal_vector, tau, eps, deps_ini,
               init_dsig_deps);

            // right element stiffness
            e_a_stiff_right = reduced_area * material_ptr->get_young_modulus(*mater_point_ptr);
            e_a_stiff_right *= geometry_ptr->reduction_factor(element_ptr, h_value_ptr);

            // calculate cable length
            cable_length_right = element_lengths(slip_id);
            ASSERT(cable_length_right>0);

            if(slip_id==1)
               {
               x1=element_coords(1,1);
               y1=element_coords(2,1);
               }
            xn=element_coords(1,1);
            yn=element_coords(2,1);

            // calculate cohesion
            if(geometry_ptr->bond_cohesion > 0)
               cohesion_right = geometry_ptr->perimeter*0.5*cable_length_right*   // surface reduced_area of the bar
                  geometry_ptr->bond_cohesion;  // cohesion force per unit reduced_area
            else // negative value input directly * (-1.)
               cohesion_right=-geometry_ptr->bond_cohesion;

            wobble_cohesion_right = geometry_ptr->perimeter*0.5*cable_length_right*   // surface reduced_area of the bar
               geometry_ptr->wobble_coefficient*max(0.,  element_force_right)/reduced_area;    //  cohesion  force  per  unit
reduced_area
            }
         else
            {
            CCInt element_id = slip_id-1;

            // element does not exist, the very last slip of the cable
            if(is_set_bit(fixed_end_total_load_map, CCExternalCablePrestress::PRESTRESS_RIGHT))
               {
               CCElement *element_ptr = &(element_group_ptr->elements(element_id));
               reduction_factor_right = get_reduction_factor(element_ptr, geometry_ptr, group_tmp_heap);
               element_force_right = prestress*area*reduction_factor_right;
               }
            else
               {
               element_force_right = 0.;
               reduction_factor_right = 1;
               }

            xn=element_coords(1,2);
            yn=element_coords(2,2);

            cohesion_right=0;
            wobble_cohesion_right = 0;
            }
         variable_cohesion_right=cohesion_right;
         variable_cohesion_left=cohesion_left;
```

```
         CCReal slip_cf_load = 1.;
         CCReal slip_cf_load_max = 1.;
         CCReal slip_cf_load_min = 1.;
         CCReal dcoh_dslip = CCREAL_ZERO;
         CCReal slip_cf_unload = geometry_ptr->slip_coeff_unload;
         CCReal slip_cf;

         CCReal SMOOTH_CF=0.2;  // smoothing between  frict_coeff and frict_cf_unload
         CCReal tot_slip= deviators_data(slip_id,TOTAL_SLIP);
         CCReal max_slip= deviators_data(slip_id,MAXIMUM_SLIP);
         CCReal min_slip= deviators_data(slip_id,MINIMUM_SLIP);
         ASSERT(max_slip+CCREAL_SMALL>=min_slip);
         CCReal delta_slip = max_slip-min_slip;
         CCReal dforce = element_force_right - element_force_left;

         if (geometry_ptr->slip_function_id)
         {
            slip_cf_load     = (**model_ptr->functions(geometry_ptr->slip_function_id))(fabs(tot_slip));
            slip_cf_load_max = (**model_ptr->functions(geometry_ptr->slip_function_id))(fabs(max_slip));
            slip_cf_load_min = (**model_ptr->functions(geometry_ptr->slip_function_id))(fabs(min_slip));
            dcoh_dslip       = (**model_ptr->functions(geometry_ptr->slip_function_id)).df_dx(fabs(tot_slip));
            if (dcoh_dslip < CCREAL_ZERO)
               dcoh_dslip = CCREAL_ZERO;
         }
         slip_cf_unload = min(geometry_ptr->slip_coeff_unload, slip_cf_load);

         if (is_equal(delta_slip, CCREAL_ZERO))
         {
            slip_cf = slip_cf_load;
         }
         else if (is_within_range(tot_slip, max_slip - SMOOTH_CF*delta_slip, max_slip, CCMIN_ABS, delta_slip*0.001))
         {  // transition interval near max_slip,
            // pushing towards max_lip use interpolated value between slip_cf_unload,  slip_cf_load
            if (dforce > CCREAL_ZERO)
            {
               xx(1) = max_slip - SMOOTH_CF*delta_slip; xx(3) = max_slip; xx(2) = (xx(1) + xx(3)) / 2.;
               yy(1) = slip_cf_unload; yy(3) = slip_cf_load_max; yy(2) = (yy(1) + yy(3)) / 2.;
               slip_cf = interpolate_lagrange(tot_slip, xx.GetSize(), xx.GetData(), yy.GetData());
            }
            else // unloading
            {
               slip_cf = slip_cf_unload;
            }
         }
         else if (is_within_range(tot_slip, min_slip, min_slip + SMOOTH_CF*delta_slip, CCMIN_ABS, delta_slip*0.001))
         {  // transition interval near min_slip
            // pushing towards min_slip, use interpolated value between slip_cf_load,  slip_cf_unload
            if (dforce < CCREAL_ZERO)
            {
               xx(1) = min_slip; xx(3) = min_slip + SMOOTH_CF*delta_slip; xx(2) = (xx(1) + xx(3)) / 2.;
               yy(1) = slip_cf_load_min; yy(3) = slip_cf_unload; yy(2) = (yy(1) + yy(3)) / 2.;
               slip_cf = interpolate_lagrange(tot_slip, xx.GetSize(), xx.GetData(), yy.GetData());
            }
            else // unloading
            {
               slip_cf = slip_cf_unload;
            }
         }
         else if (is_within_range(tot_slip, min_slip + SMOOTH_CF*delta_slip, max_slip - SMOOTH_CF*delta_slip, CCMIN_ABS,
delta_slip*0.001))
         { // tot_slip inside of <min_slip+cf*delta_slip...max_slip-cf*delta_slip, use frict_cf_unload
            slip_cf = slip_cf_unload;
         }
         else // in any other case it must be loading
         {
            slip_cf = slip_cf_load;
         }

         CCReal dcoh_dslip_left  = dcoh_dslip * variable_cohesion_left;
         CCReal dcoh_dslip_right = dcoh_dslip * variable_cohesion_right;

         variable_cohesion_right *= slip_cf;
         variable_cohesion_left *= slip_cf;

         variable_cohesion_right += wobble_cohesion_right;
         variable_cohesion_left += wobble_cohesion_left;

         CCReal loc_cf=1;

         if(geometry_ptr->loc_function_id)
            {
            CCReal location=sqrt(pow(xn-x1,2)+pow(yn-y1,2));
            loc_cf = (**model_ptr->functions(geometry_ptr->loc_function_id))(location);
            variable_cohesion_right *= loc_cf;
            variable_cohesion_left  *= loc_cf;
            }

         CCReal prestress_loss = 0;
         if(geometry_ptr->prestress_loss_function_id)
            {
            CCReal element_dforce = element_force_right-element_force_left;
            CCFunction *prestress_loss_function_ptr = model_ptr->functions(geometry_ptr->prestress_loss_function_id);
            CCReal       prestress_loss_cf       =      (*prestress_loss_function_ptr)(scoord+cable_length_right/2.)       -
(*prestress_loss_function_ptr)(scoord-cable_length_left/2.);
            prestress_loss = fabs(prestress_loss_cf)*prestress*area /* *(reduction_factor_left+reduction_factor_right)/2.) */;
            prestress_loss = min(prestress_loss, fabs(element_dforce));
            prestress_loss = -sign_transfer(prestress_loss, element_dforce);
            }

         if(geometry_ptr->temperature_function_id)
            {
            if(slip_id==1)
               temperature_left = model_ptr->base_temperature_ref;
            if(slip_id<number_of_slips)
               {
               CCElement *element_ptr = &(element_group_ptr->elements(slip_id));

               CCReal  aver_humid_at_start_time  =  0,  aver_temp_at_start_time  =  0,  aver_humid_at_target_time  =  0,
aver_temp_at_target_time = 0;
               CCTimeTempHumid::mean_humid_temp_at_element(element_ptr, aver_humid_at_start_time, aver_temp_at_start_time,
                  aver_humid_at_target_time, aver_temp_at_target_time);
               temperature_right = (aver_temp_at_start_time+aver_temp_at_target_time)/2.;
               }
            else
               temperature_right = model_ptr->base_temperature_ref;
```

```
            CCReal                                                  temper_cf                                                                       =
(**model_ptr->functions(geometry_ptr->temperature_function_id))((temperature_left+temperature_right)/2.);
            variable_cohesion_right *= temper_cf;
            variable_cohesion_left *= temper_cf;
            }

         if(geometry_ptr->corrosion_function_id)
            {
            CCReal corr_level = 1. - (reduction_factor_left+reduction_factor_right)/2.;
            CCReal corr_cf = (**model_ptr->functions(geometry_ptr->corrosion_function_id))(corr_level);
            variable_cohesion_right *= corr_cf;
            variable_cohesion_left *= corr_cf;
            }

         if(geometry_ptr->aging_function_id)
            {
            CCReal        constr_time=        model_ptr->t        -        element_group_ptr->group_construct_time        -
element_group_ptr->elements(slip_id).elem_construct_time;
            CCReal age_cf = (**model_ptr->functions(geometry_ptr->aging_function_id))(constr_time);
            variable_cohesion_right *= age_cf;
            variable_cohesion_left *= age_cf;
            }

         variable_cohesion_right=max(variable_cohesion_right,CCREAL_SMALL);  // there must be at least some cohesion there
         variable_cohesion_left=max(variable_cohesion_left, CCREAL_SMALL);

         if(slip_iteration_id==1) total_cable_length += cable_length_right;

         fi = deviator_angles(slip_id);
         /*
         Testing - for suppressing action of deviators set

         deviator_a = 1;
         deviator_b = 0;

         which produces

         deviator_a_left = deviator_a_right = 1;
         deviator_b_left = deviator_b_right = 0;
         */


         if(geometry_ptr->deviator_friction_coefficient > -CCREAL_SMALL) // correct calculation
            deviator_a = exp(-fi*geometry_ptr->deviator_friction_coefficient*geometry_ptr->perimeter /* *slip_cf */ *loc_cf);
         else   // negative value input directly * (-1.)
            deviator_a = -geometry_ptr->deviator_friction_coefficient /* *slip_cf */ *loc_cf;

         if(geometry_ptr->deviator_friction_constant > -CCREAL_SMALL)
            deviator_b = geometry_ptr->perimeter*fi*geometry_ptr->deviator_friction_constant*geometry_ptr->deviator_radius /*
*slip_cf */ *loc_cf;
         else // negative value input directly * (-1.)
            deviator_b = -geometry_ptr->deviator_friction_constant /* *slip_cf */ *loc_cf;

         if(element_force_right >= element_force_left)
            { // use F_right
            deviator_a_left = 1.;
            deviator_b_left = 0.;
            deviator_a_right = deviator_a;
            deviator_b_right = deviator_b;
            }
         else
            { // use F_left
            deviator_a_left = deviator_a;
            deviator_b_left = deviator_b;
            deviator_a_right = 1.;
            deviator_b_right = 0.;
            }

         // CCReal tensile_element_force_left = max(0, element_force_left);
         // CCReal tensile_element_force_right = max(0, element_force_right);
         CCReal tensile_element_force_left  = element_force_left;
         CCReal tensile_element_force_right = element_force_right;

         CCReal bond_strength = variable_cohesion_left + variable_cohesion_right
                              + (1. - deviator_a_left)* tensile_element_force_left + deviator_b_left
                              + (1. - deviator_a_right)*tensile_element_force_right + deviator_b_right;

         //CCReal d_slip = deviators_data(slip_id, DELTA_SLIP);

         if  ((fabs(element_force_right  -  element_force_left+prestress_loss)  >  bond_strength)  /*  ||  (  !is_equal(d_slip,
CCREAL_ZERO)) */)
            { // slip activated, should be used also if the slip is activated in the current step, in this case
              // bond stress must be exactly on the slip law curve

            if(slip_id == 1)
               { // the very first slip

               (*slip_diagonal_ptr)(slip_id) = e_a_stiff_right/cable_length_right + dcoh_dslip_right;  // diagonal
               (*slip_sub_diagonal_ptr)(slip_id) = 0.;  // sub_diagonal
               (*slip_super_diagonal_ptr)(slip_id) = -e_a_stiff_right/cable_length_right; // super_diagonal

               (*slips_ptr)(slip_id) = element_force_right-element_force_left+prestress_loss-
                  sign_transfer(variable_cohesion_right+
                  (1-deviator_a_right)*tensile_element_force_right+deviator_b_right,
                     tensile_element_force_right-tensile_element_force_left);  // rhs
               }
            else if(slip_id == number_of_slips)
               { // the very last slip
               (*slip_diagonal_ptr)(slip_id) = e_a_stiff_left/cable_length_left + dcoh_dslip_left;  // diagonal
               (*slip_sub_diagonal_ptr)(slip_id) = -e_a_stiff_left/cable_length_left;  // sub_diagonal
               (*slip_super_diagonal_ptr)(slip_id) = 0.; // super_diagonal

               (*slips_ptr)(slip_id) = element_force_right-element_force_left+prestress_loss-
                  sign_transfer(variable_cohesion_left+
                  (1.-deviator_a_left)*tensile_element_force_left+deviator_b_left,
                     tensile_element_force_right-tensile_element_force_left);  // rhs
               }
            else
               { // intermediate slips

               (*slip_diagonal_ptr)(slip_id)= e_a_stiff_left/cable_length_left
                  + e_a_stiff_right/cable_length_right
                  + dcoh_dslip_left + dcoh_dslip_right;  // diagonal
               (*slip_sub_diagonal_ptr)(slip_id)=
                  -e_a_stiff_left/cable_length_left;  // sub_diagonal
               (*slip_super_diagonal_ptr)(slip_id)=
```

```
                  -e_a_stiff_right/cable_length_right; // super_diagonal
               (*slips_ptr)(slip_id) = element_force_right - element_force_left+prestress_loss -
                  sign_transfer(bond_strength, tensile_element_force_right-tensile_element_force_left);  // rhs
               }
            }
         else
            { // slip not activated, ie. it will not change

            if(slip_id == 1)
               { // the very first slip
               (*slip_diagonal_ptr)(slip_id)= e_a_stiff_right/cable_length_right; // any nonzero number
               }
            else if(slip_id == number_of_slips)
               { // the very last slip
               (*slip_diagonal_ptr)(slip_id)= e_a_stiff_left/cable_length_left;  // any nonzero number
               }
            else
               { // intermediate slips
               (*slip_diagonal_ptr)(slip_id)= e_a_stiff_left/cable_length_left+
                  e_a_stiff_right/cable_length_right; // any nonzero number
               }
            (*slip_super_diagonal_ptr)(slip_id)= 0.;
            (*slip_sub_diagonal_ptr)(slip_id)= 0.;
            (*slips_ptr)(slip_id) = 0.;
            fix_the_bar=CCFALSE;
            }

         // copy some vital data for the next element
         cable_length_left=cable_length_right;
         element_force_left=element_force_right;
         cohesion_left=cohesion_right;
         wobble_cohesion_left = wobble_cohesion_right;
         reduction_factor_left = reduction_factor_right;
         temperature_left = temperature_right;
         e_a_stiff_left=e_a_stiff_right;
         slip_left= slip_right;
         }

      // solve the system for new slip increments

      if(is_set_bit(fixed_end_geometry_map,    CCExternalCablePrestress::FIX_LEFT)    &&    is_set_bit(fixed_end_geometry_map,
CCExternalCablePrestress::FIX_RIGHT))
         {  // fixing of the left and right end points required by user
         (*slips_ptr)(1)=0.;
         (*slips_ptr)(number_of_slips)=0.;
         long number_of_eqns=number_of_elements-1;
         if(number_of_eqns>0)
            MY_DLSLTR(&number_of_eqns, &((*slip_sub_diagonal_ptr)(2)),   // number_of_elements (=number_of_slips-1) dofs
            &((*slip_diagonal_ptr)(2)), &((*slip_super_diagonal_ptr)(2)),
            &((*slips_ptr)(2)));
         }
      else if(is_set_bit(fixed_end_geometry_map, CCExternalCablePrestress::FIX_LEFT))
         { // fixing of the left end points required by user
         (*slips_ptr)(1)=0.;
         if(number_of_elements>0)
            MY_DLSLTR(&number_of_elements, &((*slip_sub_diagonal_ptr)(2)),   // number_of_elements (=number_of_slips-1) dofs
            &((*slip_diagonal_ptr)(2)), &((*slip_super_diagonal_ptr)(2)),
            &((*slips_ptr)(2)));
         }
      else if(is_set_bit(fixed_end_geometry_map, CCExternalCablePrestress::FIX_RIGHT))
         {  // fixing of the right end points required by user
         (*slips_ptr)(number_of_slips)=0.;
         if(number_of_elements>0)
            MY_DLSLTR(&number_of_elements, &((*slip_sub_diagonal_ptr)(1)),   // number_of_elements (=number_of_slips-1) dofs
            &((*slip_diagonal_ptr)(1)), &((*slip_super_diagonal_ptr)(1)),
            &((*slips_ptr)(1)));
         }
#ifdef FIX_1ST_SLIP_IF_ALL_NODES_SLIP
      else if(fix_the_bar)
         {  // all segments of the bar are slipping, fix the 1st. slip incr. (within the iteration) in order to avoid singularity
            // (much the same as for geometry_ptr->fix_the_bar_left)
         (*slips_ptr)(1)=0.;
         if(number_of_elements>0)
            MY_DLSLTR(&number_of_elements, &((*slip_sub_diagonal_ptr)(2)),   // number_of_elements (=number_of_slips-1) dofs
            &((*slip_diagonal_ptr)(2)), &((*slip_super_diagonal_ptr)(2)),
            &((*slips_ptr)(2)));
         }
#endif
      else
         {  // normal processing, the bar somewhere fixed due to not activated slip
         if(number_of_slips>0)
            MY_DLSLTR(&number_of_slips, &((*slip_sub_diagonal_ptr)(1)),   // number_of_slips dofs
            &((*slip_diagonal_ptr)(1)), &((*slip_super_diagonal_ptr)(1)),
            &((*slips_ptr)(1)));
         }

      // solve the system for new slip increments
      slip_displs_error = pow(dot_product(
         (const CCRealColumnVector&) *slips_ptr,
         (const CCRealColumnVector&) *slips_ptr),0.5)
         * geometry_ptr->precision_factor / total_cable_length;


      for(CCInt slip_id = 1; slip_id <= number_of_slips; slip_id++)
         {
         deviators_data(slip_id,TOTAL_SLIP) += geometry_ptr->damping_factor *
            (*slips_ptr)(slip_id); // update deviator slips
         deviators_data(slip_id,DELTA_SLIP) += geometry_ptr->damping_factor *
            (*slips_ptr)(slip_id); // update deviator slips
         if(deviators_data(slip_id,MAXIMUM_SLIP)<deviators_data(slip_id,TOTAL_SLIP))
            deviators_data(slip_id,MAXIMUM_SLIP) = deviators_data(slip_id,TOTAL_SLIP);
         if(deviators_data(slip_id,MINIMUM_SLIP)>deviators_data(slip_id,TOTAL_SLIP))
            deviators_data(slip_id,MINIMUM_SLIP) = deviators_data(slip_id,TOTAL_SLIP);
         }

      slip_iteration_id++;

//       afxDump << "\n step_id: " << model_ptr->step_id << " iteration_id: " << model_ptr->iteration_id << " slip_iteration_id:
" << slip_iteration_id << "\n";
//       dRM(deviators_data);
      }

   if(slip_iteration_id > switches.iteration_limit)
      {
      //       message(MSG_EXC_PLACE, "CCStructures", IDS_CCStructuresWar_FAIL_CONV, 0); // divergence
      msg.Format("iteration %5i, error %10.2g", slip_iteration_id, slip_displs_error);
```

```
   message(MSG_PLACE_HIDDEN /*MSG_EXC_PLACE*/, "CCStructures", IDS_CCStructuresWar_EXTERNAL_CABLE_SLIPS_HDR, 2, msg,
      element_group_label);
   if(slip_iteration_id == 1) element_group_label = ""; // delete cable id for the subsequent iterations
   }

if(reset_dslips_at_end)
   group_prestress_ptr->reset_dslips=CCTRUE;

geometry_ptr->deviator_radius = deviator_radius_save;
if(geometry_ptr->release_cable)
   {
   // restore after temporary release fix of external cable in pre-stressing phase
   geometry_ptr->deviator_friction_coefficient = deviator_friction_coefficient_save;
   geometry_ptr->slip_coeff_unload = slip_coeff_unload_save;
   geometry_ptr->deviator_friction_constant = deviator_friction_constant_save;
   geometry_ptr->bond_cohesion = bond_cohesion_save;
   geometry_ptr->wobble_coefficient = wobble_coefficient_save;
   }
}
```

## 5.5 Run Time Analysis, Output and Post-Processing

Simulation execution is done via the standard ATENA run-time and post-processing Tool AtenaStudio, see ATENA Studio User's Manual [20].

## 5.6 Advanced Usage (Optional)

- Parametric studies with different SCM ratios
- Integration with NumPy/Pandas for optimization
- Automated generation of spiral configurations
- Scripting experimental comparison plots

# 6 Examples and Validation

## 6.1 MRCS Joint Experiments

This section summarizes the modelling of MRCS column-steel joints tested in Taiwan. The following is an excerpt from the work of the Taiwanese side. The two experiments that were carefully designed, prepared and carried out were performed on the same column geometry and displacement load, but different values of axial preload:

**EBL (low axial load)—**this variant was axially pre-loaded by $0.1 \times f'_c A_g$ (3018 kN)

**EBH (high axial load)—**this variant was axially pre-loaded by $0.5 \times f'_c A_g$ (15556 kN)

The tested reinforced concrete + steel column (RCS) was 600 × 600 mm in its cross-section and encased an 800 × 240 steel beam to form a joint, as shown in Fig. 14 and Fig. 15.

Numerical simulations were performed in the ATENA software using advanced material models and experience from solutions of former CeSTaR projects (CeSTaR I and CeSTaR II) and also experience from other forms of blind predictions with similar setups. The Taiwanese side provided the necessary experimental data for preparation of an advanced numerical model and its validation against the experiment. The numerical model was developed with respect to the correct geometry, column fixing, load, material interfaces and materials used.

During the first project year, there was detailed communication between the two sides about performed material tests, reinforcement geometry, concrete strength etc. The Taiwanese side provided experimental stress-strain diagrams for all steel reinforcement materials and for the beam web, flange, doubler plate and face bearing plate materials, which were incorporated into ATENA either as a steel elastic material or as a multilinear reinforcement material.

Reinforcement is modelled as embedded, and its mesh is made as 1 element for each 1D member and subsequently division is made after the creation of the hexahedral elements for concrete. Steel and concrete materials use linear hexahedral elements and contact between individual bodies is created either by fixed contacts for neighboring bodies for fixed entities or via an interface material for the steel beam-concrete interface that only allows for transfer of compressive forces but allows separation of the materials in the tensile direction.

To accurately represent the boundary conditions of the experiment, the column was fixed by a stiff metal slab on both ends and allowed to rotate along a surface-dividing line on top and allowed to translate in the direction of the *x* axis.

Loading was performed as described in the loading diagram in Fig. 16 by displacing the tip of the steel beam in the vertical direction.

Since the inside of the joint is a complicated geometry with multiple material interfaces, the contacts between individual bodies must be prepared carefully. shows the internal parts of the joint without concrete members.

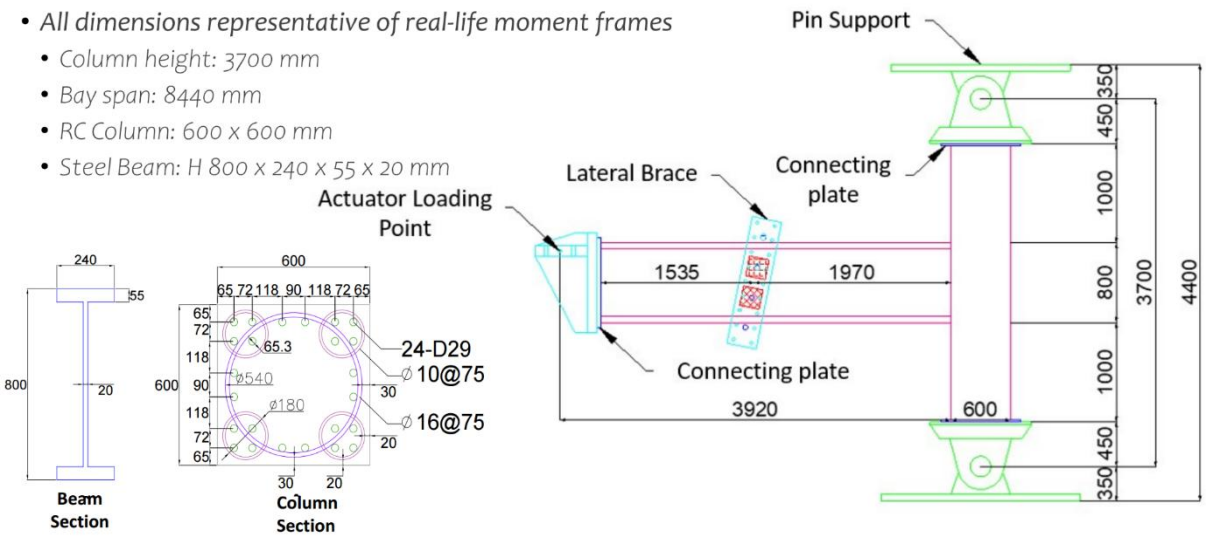


Fig. 14. All dimensions (mm) of both analyzed MRCS specimens

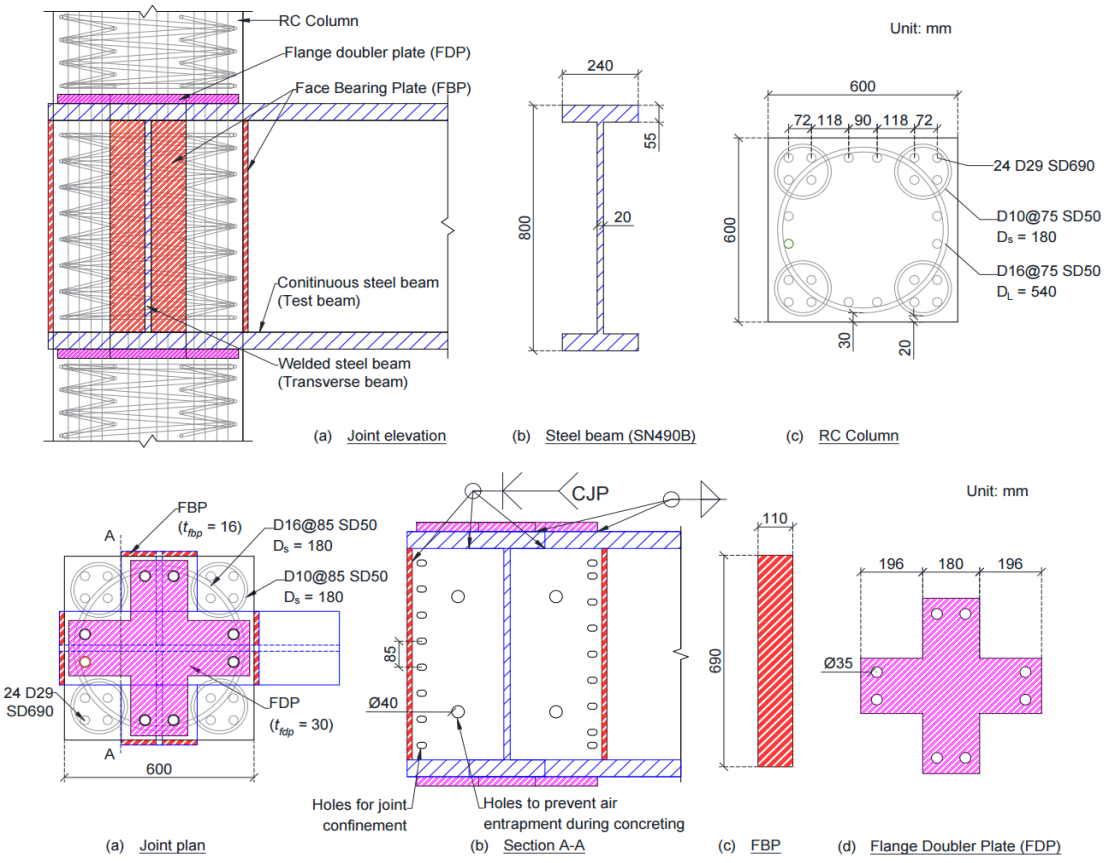


Fig. 15. Detail of the steel beam, its openings and reinforcing members

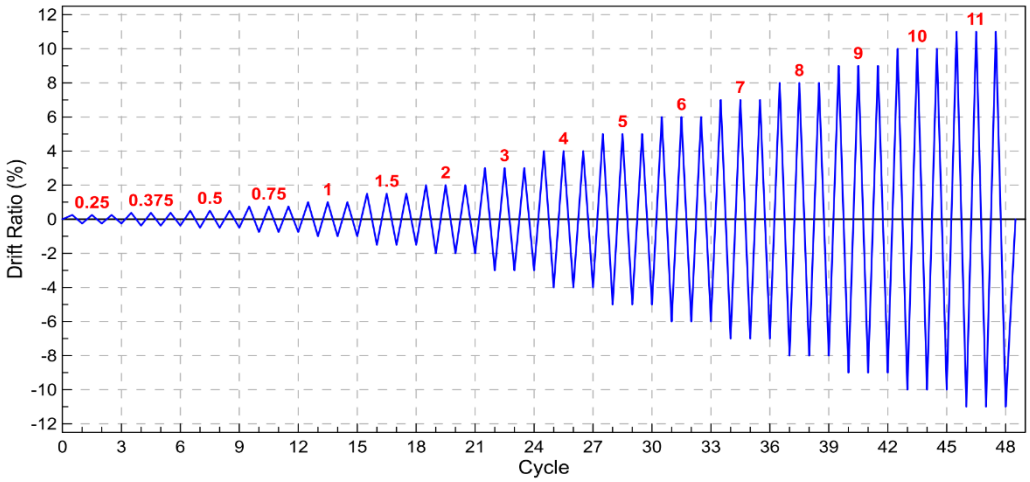


Fig. 16. The cyclic load program. Vertical displacement shown as drift ratio (fraction of vertical displacement and beam length, which is 3500 mm)

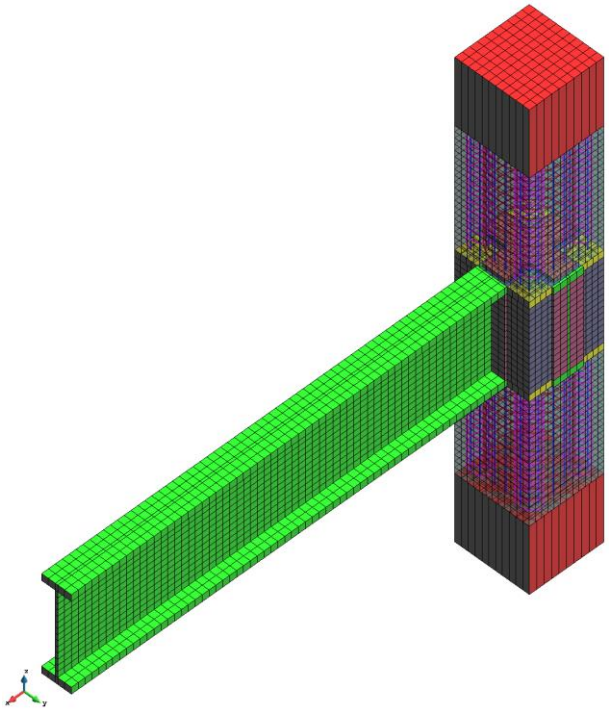

Fig. 17. Isometric view of the ATENA model with all used components and hexahedral mesh elements for steel and concrete.

We can observe many important details in the joint. Separation of the reinforcement between the joint and column was modelled (as performed in the experiment), column concrete (not visible in Fig. 17) was connected to steel beam members via a compression-only interface, connection of beam web and flanges as well as the front bearing plates was done by using compatible hexahedral mesh and the flange doubler plate was modelled to be in fixed contact with both steel and concrete neighboring volumes. The openings for reinforcement and concreting, which were

present in the real experiment, were not modelled for sake of simplicity, reduction of comp. times and to allow good quality meshing with hexahedral elements.

The lateral brace seen in Fig. 18 was modelled simply by fixing the *y* axis displacement of some of the elements after meshing.

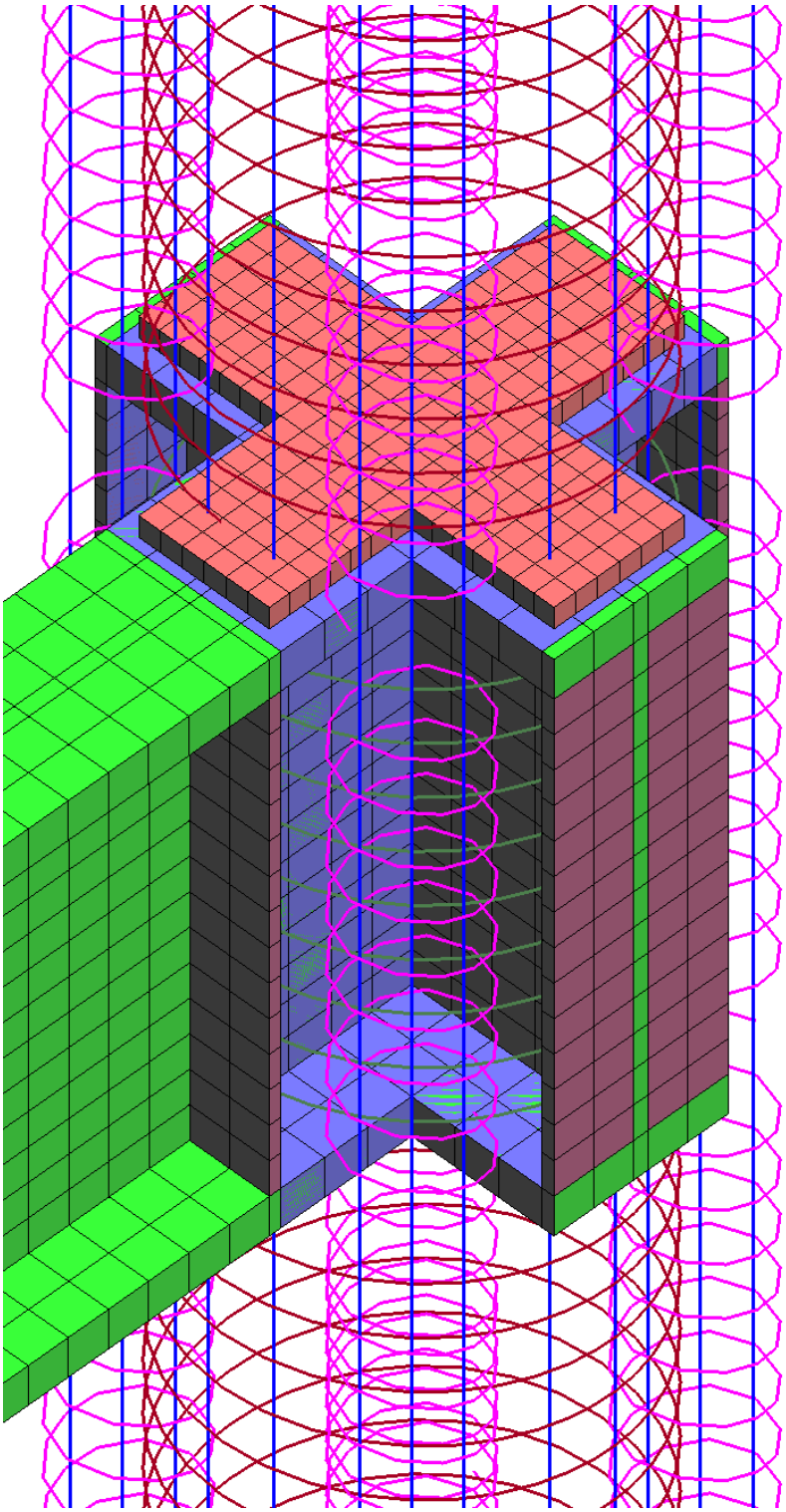

Fig. 18. Joint detail without column concrete

### 6.1.1 Static Simulation – Push-over Analysis

To assess the structural performance and deformation capacity of the proposed MRCS system under lateral loading, nonlinear static pushover analyses were conducted using the ATENA software. These simulations provide insight into the strength, stiffness degradation, and ductility of the system under monotonic loading, and serve as a preparatory step before undertaking full cyclic analysis. The goal of this initial phase was to evaluate the modelling assumptions and material behavior, and to calibrate the numerical model against experimental observations.

The load–displacement response of the MRCS system was evaluated by recording the total reaction force at the tip of the steel beam and plotting it against the drift ratio—defined as the beam tip displacement divided by the beam length (3500 mm). This approach enabled the assessment of global lateral stiffness, strength, and post-peak behavior of the structural system.

Two column configurations were considered, distinguished by their axial preload levels:

- ELJ variant: Low axial load, $0.1 \times f'_c A_g$ (3018 kN)
- EHJ variant: High axial load, $0.5 \times f'_c A_g$ (15556 kN)

Initial simulations were performed for the ELJ configuration, with the EHJ variant subsequently analyzed in 2024.

The first set of results under monotonic static loading—using baseline material parameters—are shown Fig. 19 and Fig. 20. The numerical model demonstrated excellent agreement in the pre-peak region Fig. 19, capturing stiffness and load-bearing behavior accurately. However, post-peak softening was more ductile in the numerical predictions compared to the experimental results. This discrepancy is likely due to the inability of the current model to capture flange rupture in the steel beam, particularly the top flange, which failed in the experiment but is not yet modelled in ATENA.

These findings underscore the need for refinements in the modelling strategy before extending the simulations to **cyclic loading scenarios**.

The pilot simulations reveal some limitations of the current **steel material model in ATENA**, which relies solely on a **plasticity-based formulation**. To improve realism—especially in capturing post-peak fracture and failure mechanisms—it is proposed to extend the steel model using a concept analogous to the **fracture-plastic model already used for concrete** in ATENA. This existing model combines:

- **Smeared crack modelling** (via crack band theory) for tensile fracture
- **Plasticity-based compressive behavior** for concrete crushing

Applying a similar **fracture–plasticity hybrid approach** to the steel material is particularly relevant for **high-strength steel**, where **tensile fracture following plastic deformation** can become a dominant failure mode.

Additionally, several other components of the simulation framework may require revision to accurately capture the MRCS system's behavior under cyclic loading:

- **Interface/contact elements**: Especially critical for joint behavior during load reversals
- **Cyclic bond–slip model** for reinforcement anchorage
- **Material models for SCM concrete**: Enhancements to capture hardening/softening behavior under repeated loading

These upgrades will improve the fidelity of the simulation results and allow for the reliable prediction of MRCS system performance under both monotonic and cyclic loads.

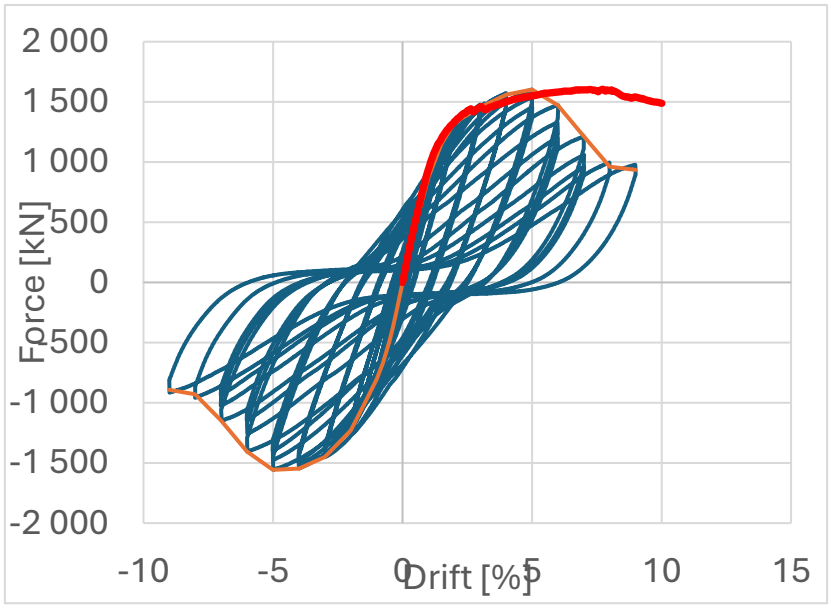


Fig. 19. Results for non-cycling response from ATENA simulation, equivalent lateral column force versus drift.

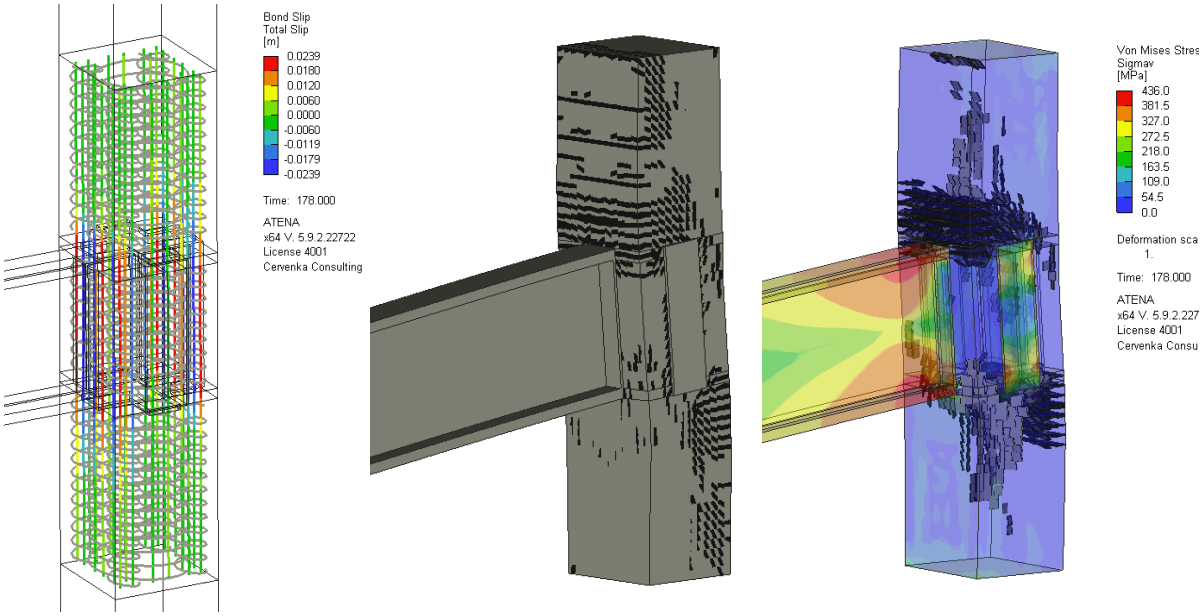


Fig. 20. Crack pattern (Top), interior major cracks > 0.5 mm with display of Von Mises stress in steel elements (bottom), loss of bond in longitudinal reinforcement (left).

### 6.1.2 Cycling Simulation

In 2024, the simulations in ATENA were performed and validated using the experimental data peak column shear force (beam load force) and the corresponding achieved drift, as described in the figures below in this section. Due to experiment budget restrictions on the Taiwanese side, we had two specimens already mentioned in the 2023 report to work with—the EJL and EJH experiments, which are identical in geometry but vary in the prescribed axial pre-load (10% and 50%, respectively). The 2023 report describes their design, fabrication, loading regime, and results in detail. The geometries are also shown here in Figures 2.1.1 and 2.1.2.

During 2023, we performed simulations of a static envelope instead of the full cyclic load and showed results in good agreement with the experimental envelope. However, the cyclic behavior

was not investigated, and the static envelope simulation mode was only performed for the EJL experiment. During 2024, we focused on developing the numerical model that describes cyclic behavior. To capture the fracturing of the beam's flange that occurred for the EJH specimen (Fig. 23), we introduced interface elements that allow the transfer of forces in the compressive direction but also allow for the separation of adjacent surfaces in tension.

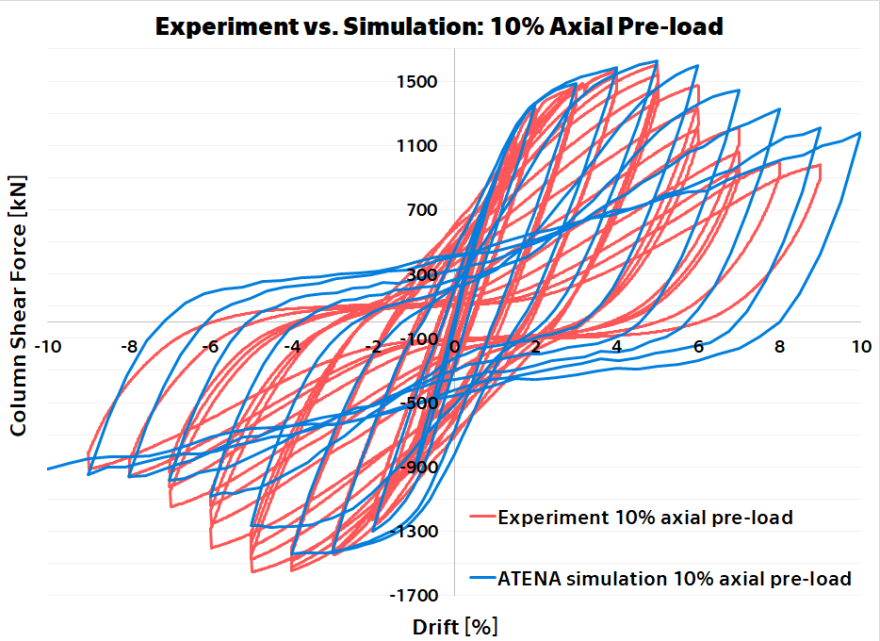


Fig. 21. The force-drift diagram of the EJL experiment in red (10% axial pre-load) and the ATENA simulation in blue, showing a moderately good agreement between the experiment and simulation. The ATENA simulation predicts slightly stiffer behavior in the post-peak region, but the peak force is a very good match and the softening trend follows.

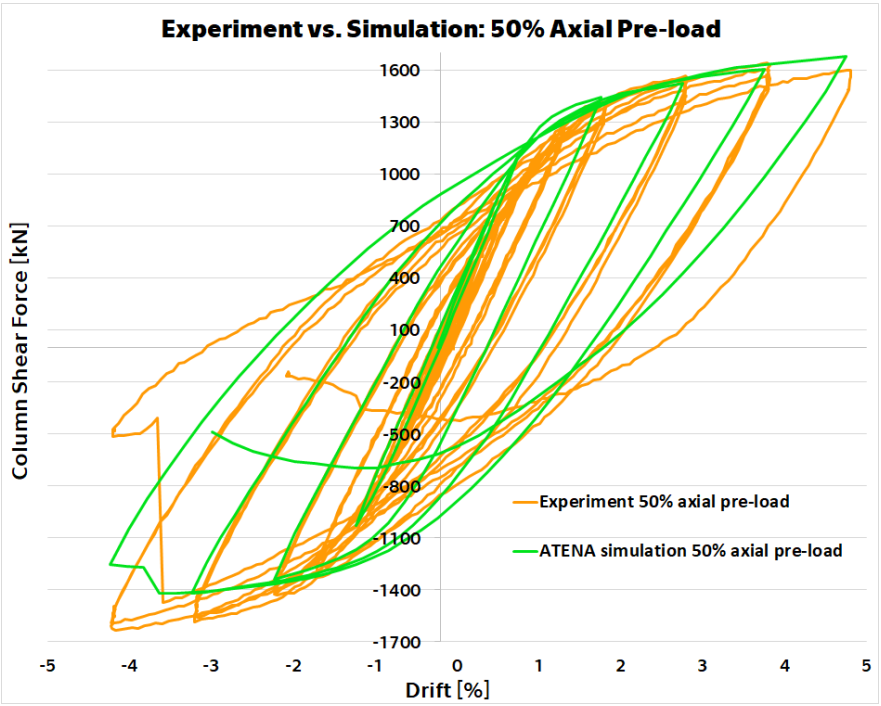


Fig. 22. The force-drift diagram of the EJH experiment in orange (50% axial pre-load) and the ATENA simulation in green. The experiment was terminated by rupturing of the beam's flange (Fig. 23) at only ca. half the drift of the EJL experiment. The simulation follows this trend by

predicting failure sooner, but only at ca. 3% drift, before the peak force achieved during the experiment. The rupturing of the flange during the experiment is shown in Fig. 23.


Fig. 23: Rupturing of the beam's flange (EJH specimen) during the experiment.

The simulations that compare various levels of axial pre-load. There is no further available experimental data due to the budget reductions on the Taiwanese side.

## 6.2 Cycling Tests of Multi-Story Structure

This section contains a description of additional independent testing and validation of the numerical model for simulating cycling behavior. This is part of the effort of continuous testing and improvement of numerical models for the modeling of reinforced concrete structures in cycling and dynamic type of loading. The best test is the participation in blind competitions organized by various institutions. The competition was organized by Bowen Laboratory at Purdue University, USA, and involved a cycling test of full-scale two-story RC structure (see Fig. 24). The full results of the Purdue competition are still to be announced after the experimental program completion in March 2025. The submitted predictions by the project team are shown in Fig. 25.

Fig. 24. Two blind competitions used for FEM model tunning and validation in Task 2.4, (left) prestressed RC girder from Swiss test, (right) experiment of full scale two-story RC frame from Purdue University.

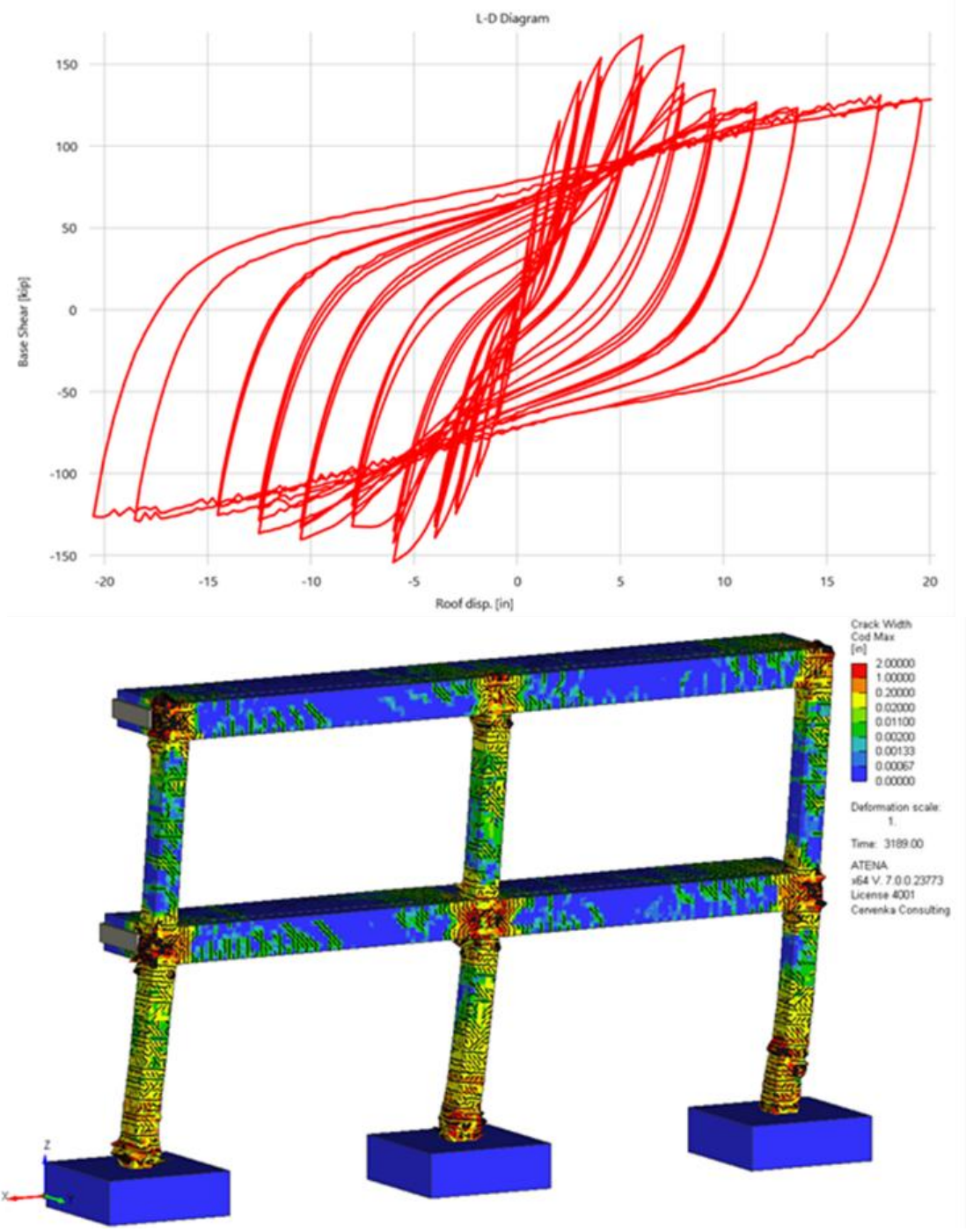


Fig. 25. Prediction submitted by CeSTaR-3 project partners for Purdue University competition. The results are still expected to be published in March 2025, model without retrofitting.

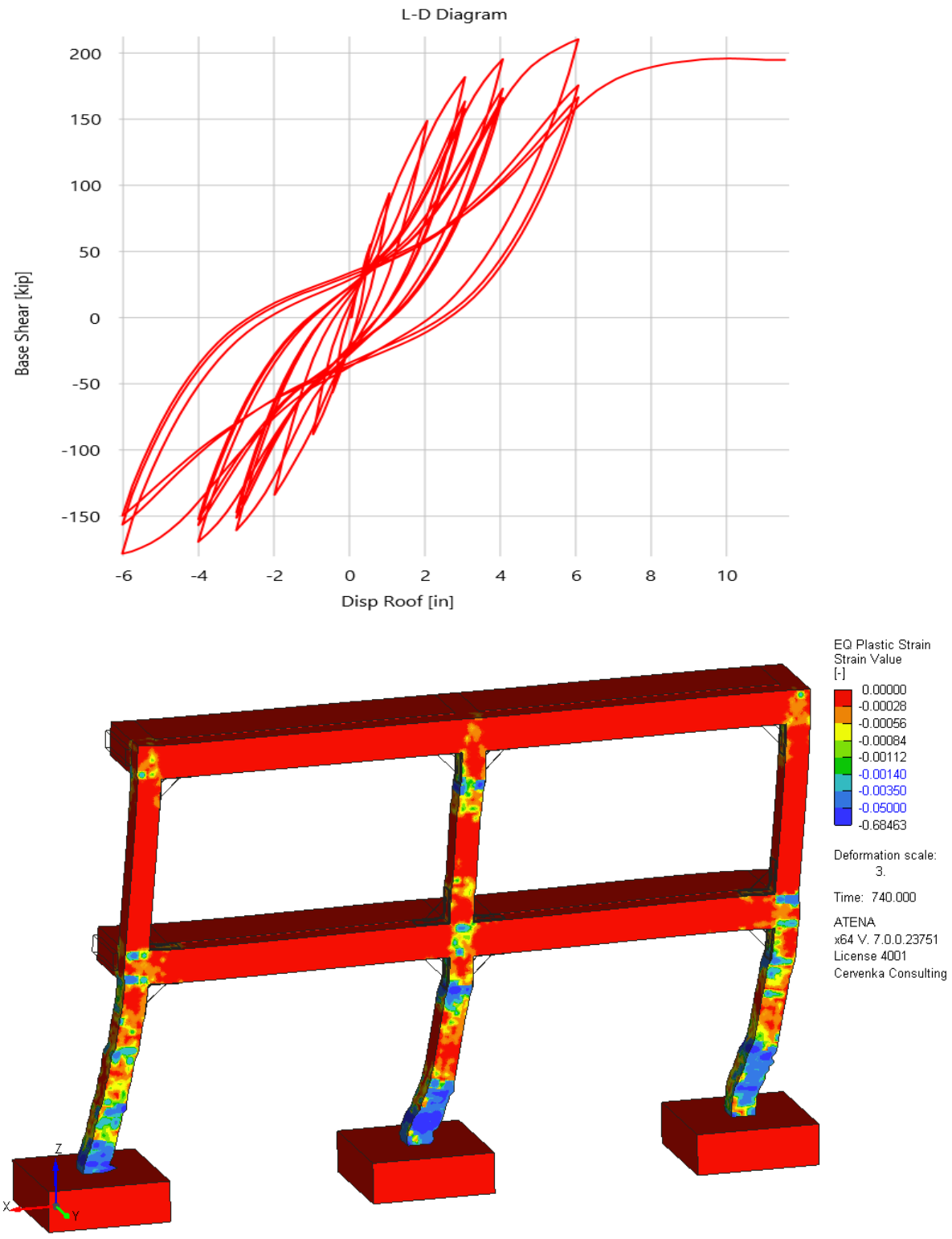


Fig. 26. Prediction submitted by CeSTaR-3 project partners for Purdue University competition. The results are still expected to be published in March 2025, model without retrofitting.

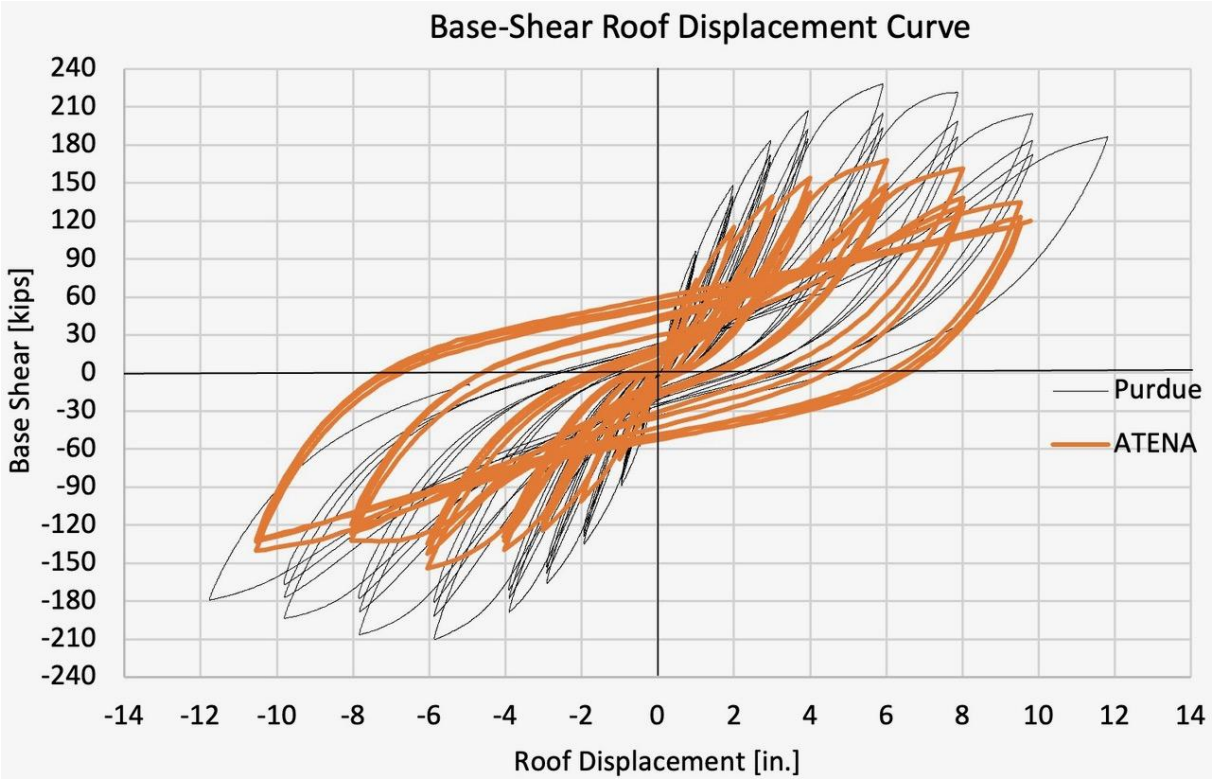


Fig. 27. Comparison of the predicted and experimental load-displacement curves from Purdue benchmark.

The CeSTaR-3 team aims to develop robust numerical models with high predictive accuracy, and these blind competitions provide a unique platform for testing the models as well as learning from failures and mistakes.

## 6.3 Shear Failure in Pre-stressed Girder

Last year the tunning of FEM numerical models was validated in two blind competitions: the first one was organized by XDEEA during a seminar on NLFEM in Wildegg, Switzerland on the shear behavior of a pre-stressed girder.

The prediction by models used and developed in CeSTaR-3 had the best prediction in the Swiss competition as is shown in the announcement in Figure 2.4.2.

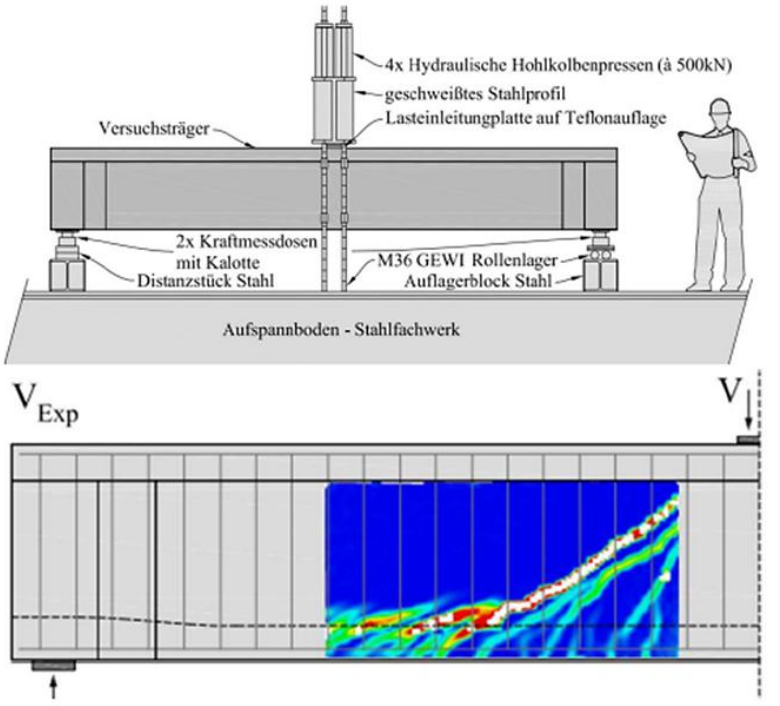


Fig. 28. Prestressed RC girder from Swiss blind competition.

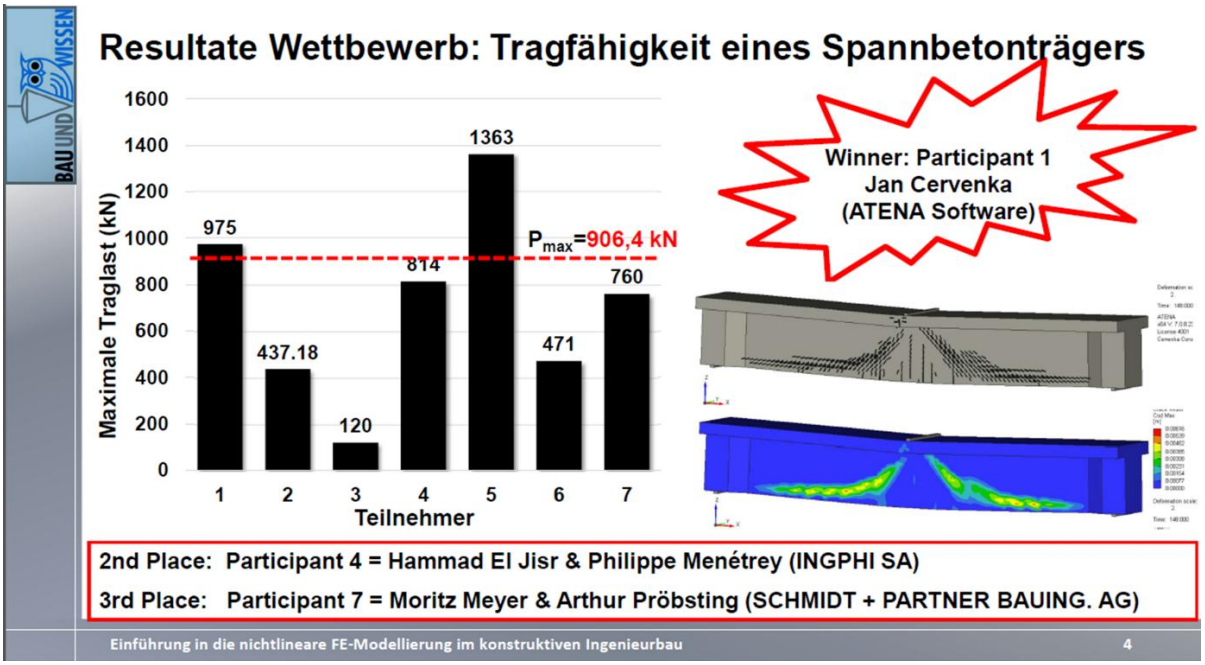


Fig. 29. A slide from the announcement of the winning prediction by CeSTaR-3 project participants.

## 6.4 Punching Shear fib WG 2.4.1 Validation

This section describes additional blind competition organized by fib WG 2.4.1 for punching-shear problems. This is a very good testing problem, as it can evaluate the behavior of the ATENA concrete model in similar stress states with high confinement, as expected in the experiments performed in this project. Fig. 30 shows the geometry of the experiment. It was organized as a blind prediction, so the participants knew only the geometric data, loading process and basic material properties. The blind prediction by the project members is reported in Fig. 32, where it is indicated as a red line. The experimental result is indicated by blue color. The predictions by the other participants are in grey color.

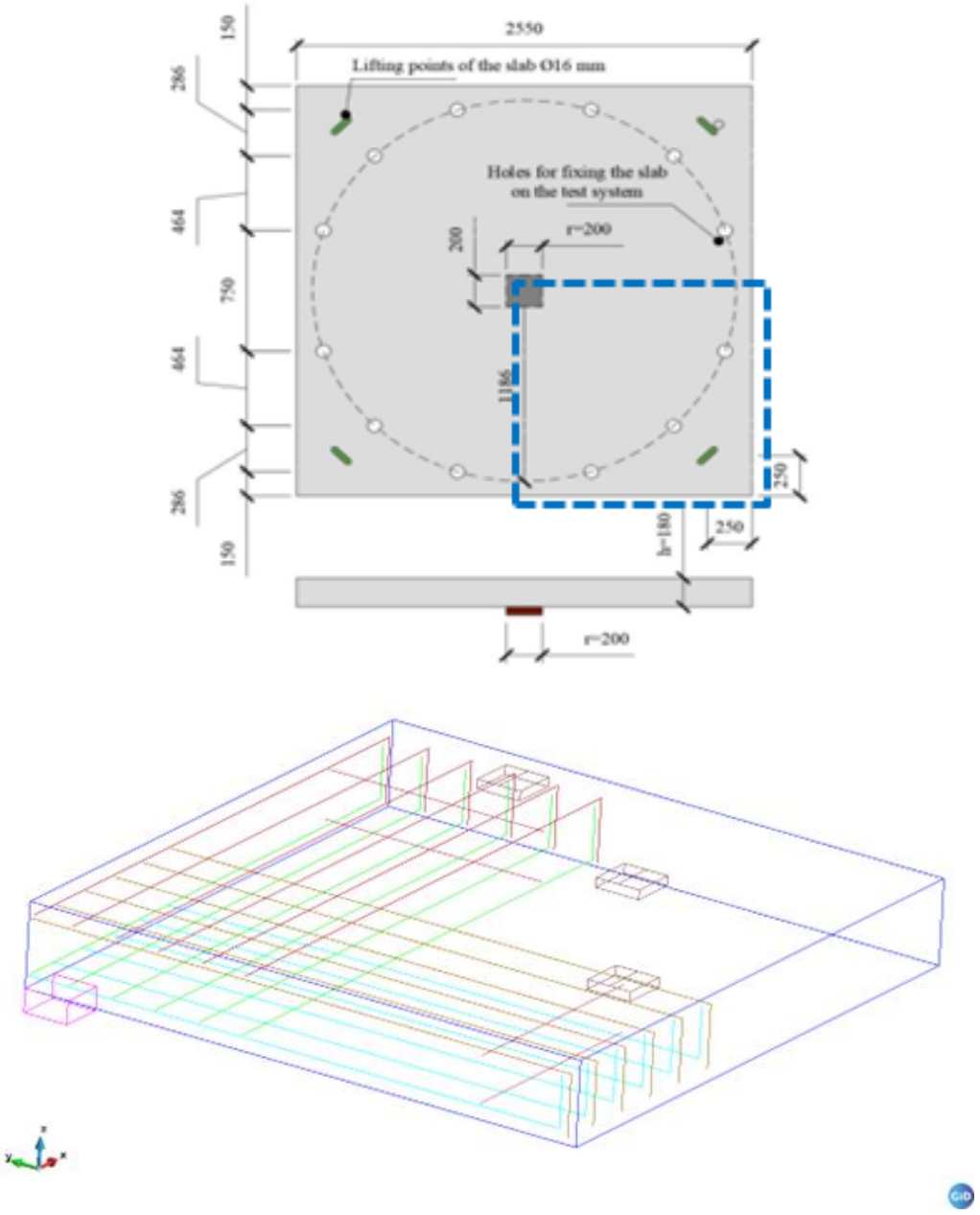


Fig. 30. fib WG 2.4.1 punching benchmark problem and the geometry of the numerical model.

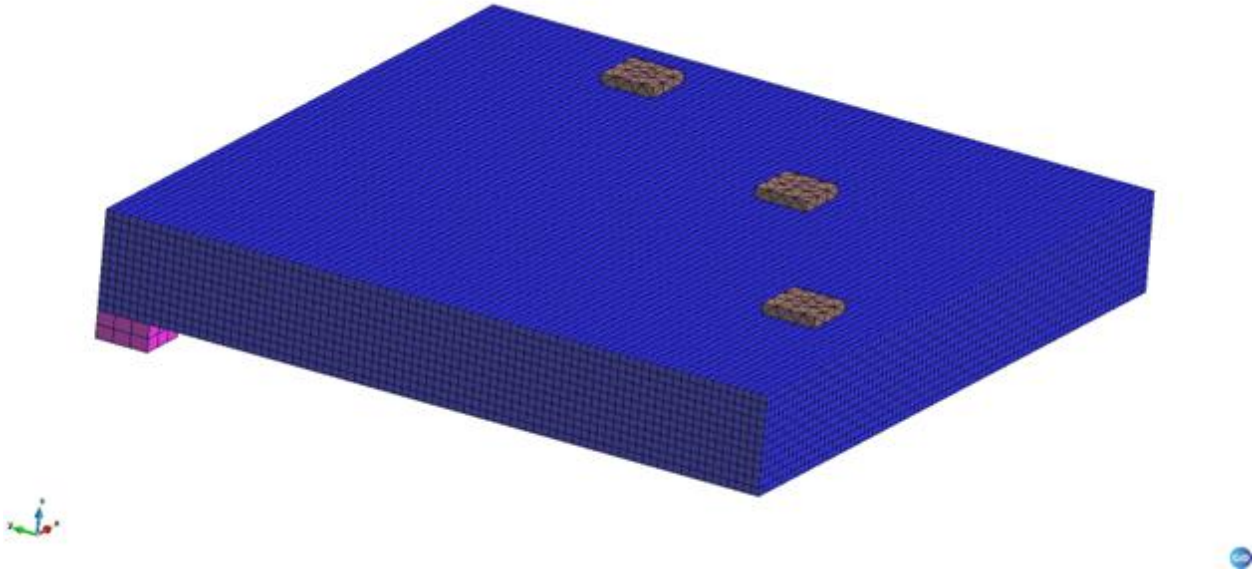

Fig. 31. FE mesh of fib WG 2.4.1 punching benchmark

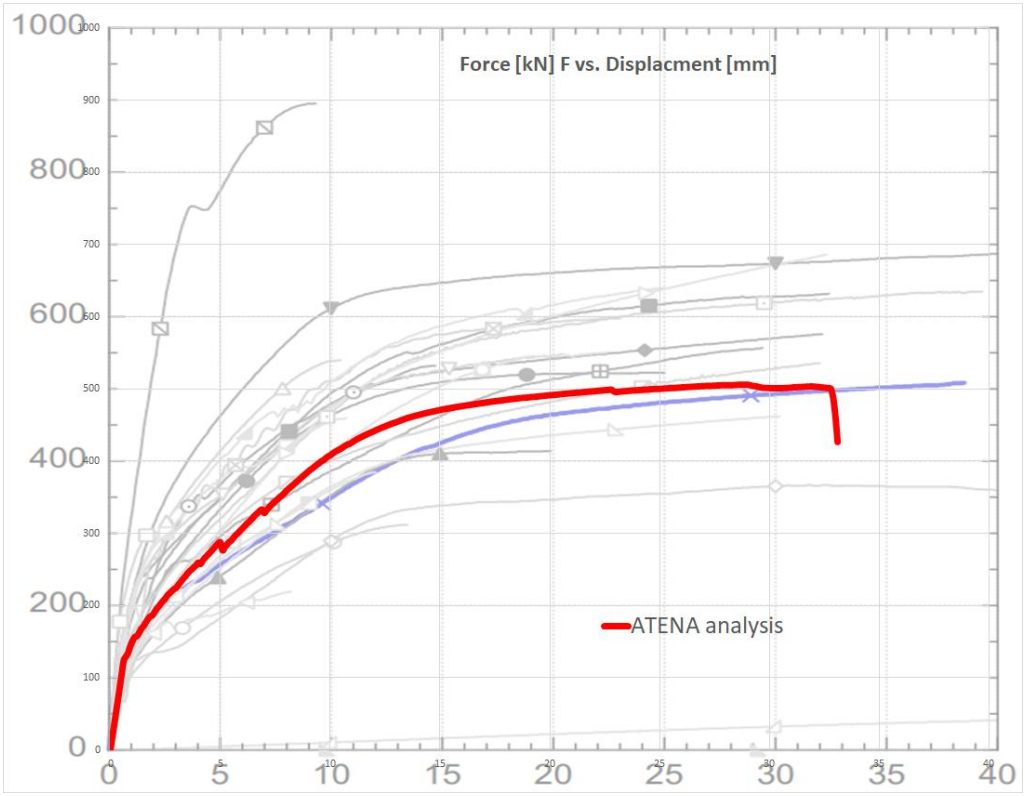


Fig. 32. Comparison of fib WG 2.4.1 punching benchmark (blue line: experiment)

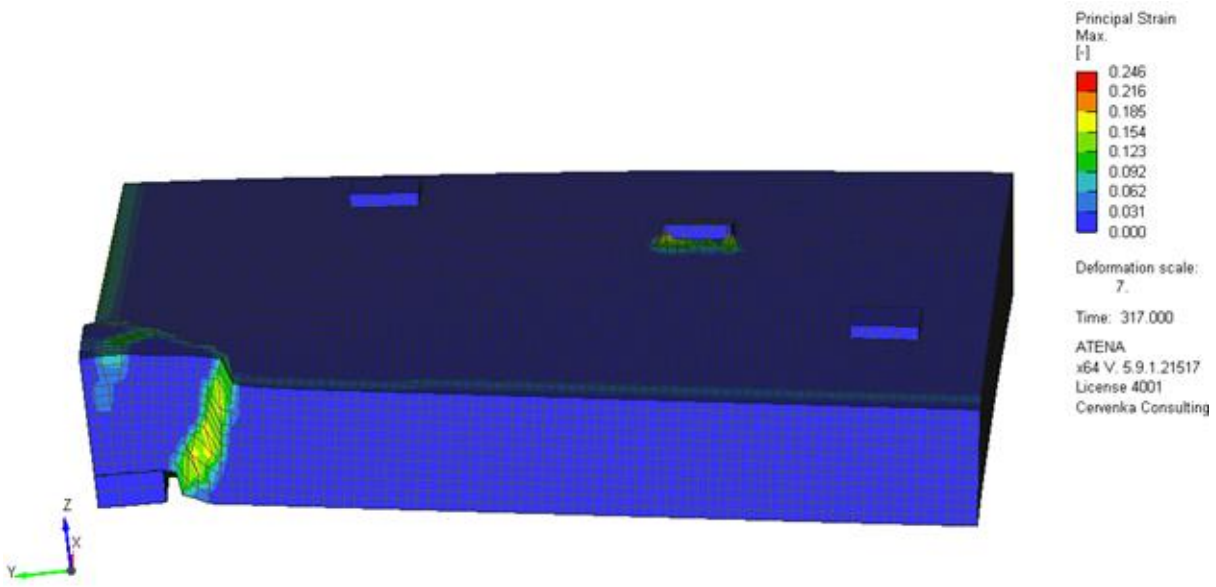


Fig. 33. Development of shear failure mode in the simulation of fib WG 2.4.1 blind prediction competition.

Considering the typical scatter of concrete material properties when for instance the coefficient of variation of concrete compressive strength is typically around 15%, the results of the blind prediction show very good match in the peak load as well as reasonable prediction of the stiffness and pre-peak response (Fig. 32).

## 6.5 Summary of all Recent ATENA Validation Benchmarks

This section summarizes additional blind competition results of ATENA software. Verification of simulation models for concrete structures is typically conducted through comparison with experimental data. Interesting insight can be obtained from blind predictions in international competitions, when the experimental results are not known at the time of the analysis. Fig. 34 to Fig. 39 summarizes several such contests and benchmarks in which the authors participated, for more details see Cervenka et. al. (2024) [22]. The overall summary is prodived in Fig. 39, where the predicted strength is normalized by the ratio $F_{sim}/F_{exp}$ with 22 cases from seven benchmark contests displayed on the horizontal axis. The vertical bars represent the prediction scatter, while the author's results are marked in green.

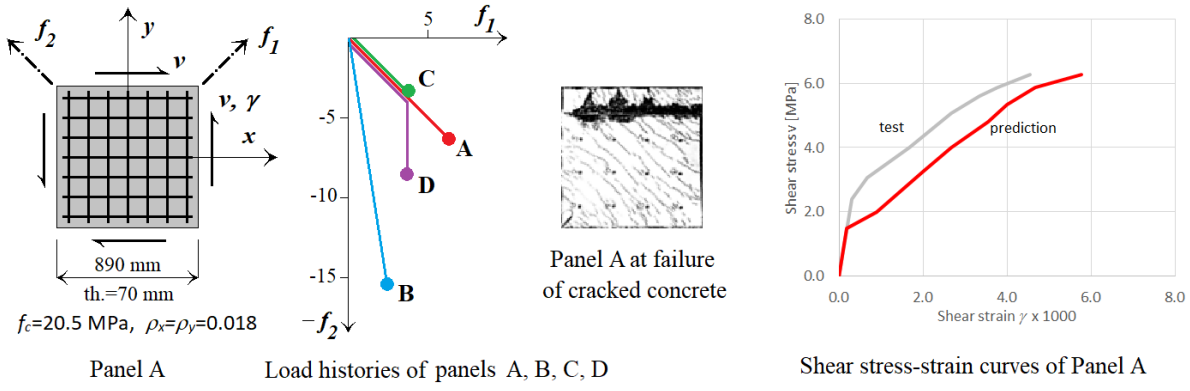


Fig. 34. Reinforced concrete panels, Toronto 1985.

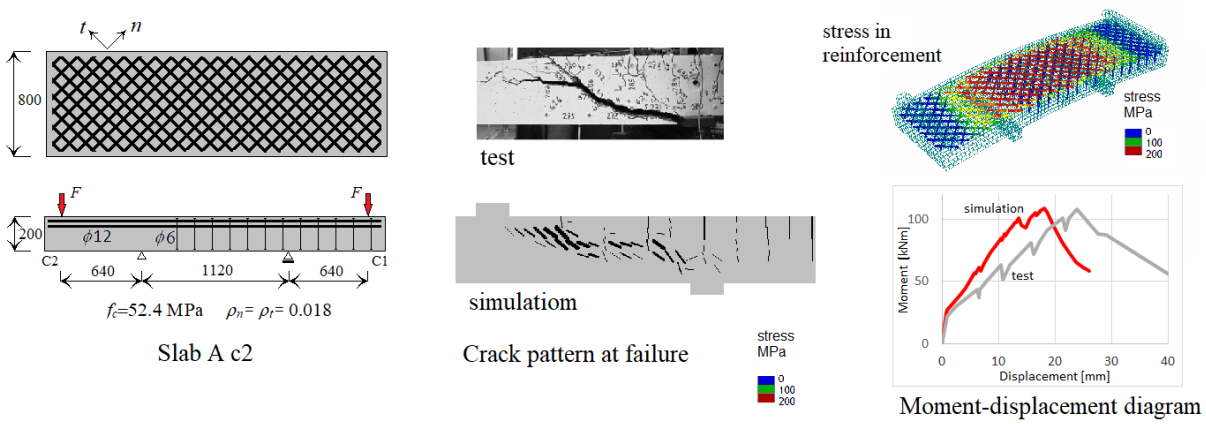


Fig. 35. Reinforced concrete slab, ETH 2005.

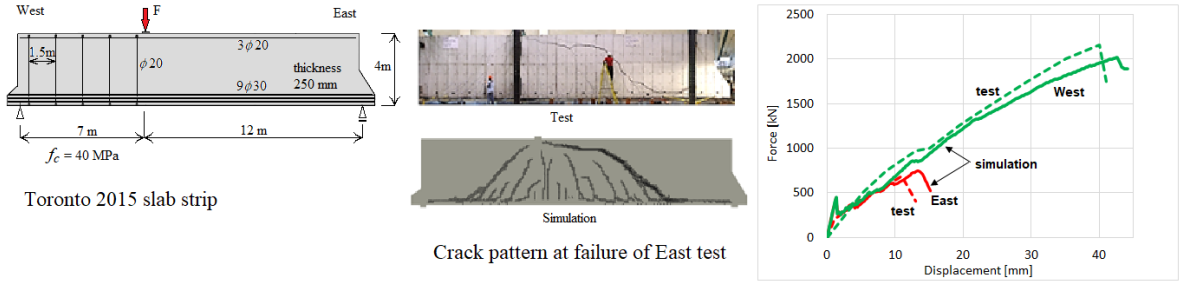


Fig. 36 Shear strength of very thick slab strip, Toronto 2015.

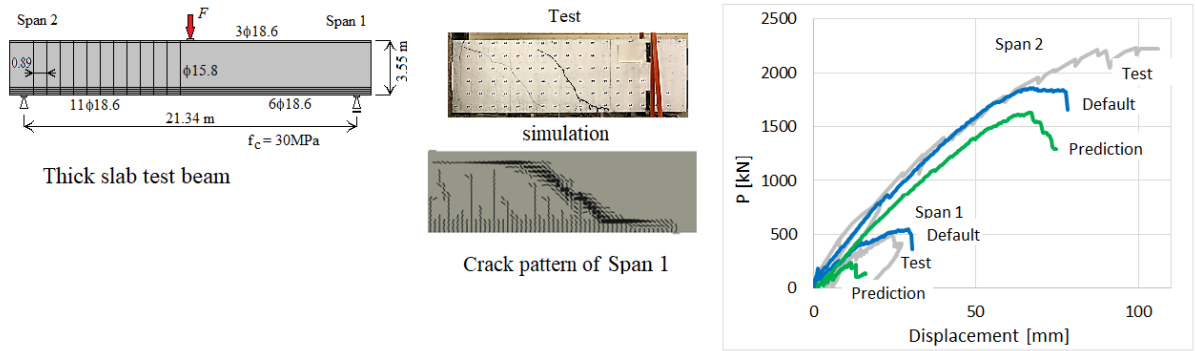


Fig. 37 Shear strength of reinforced concrete foundation, Berkeley 2021.

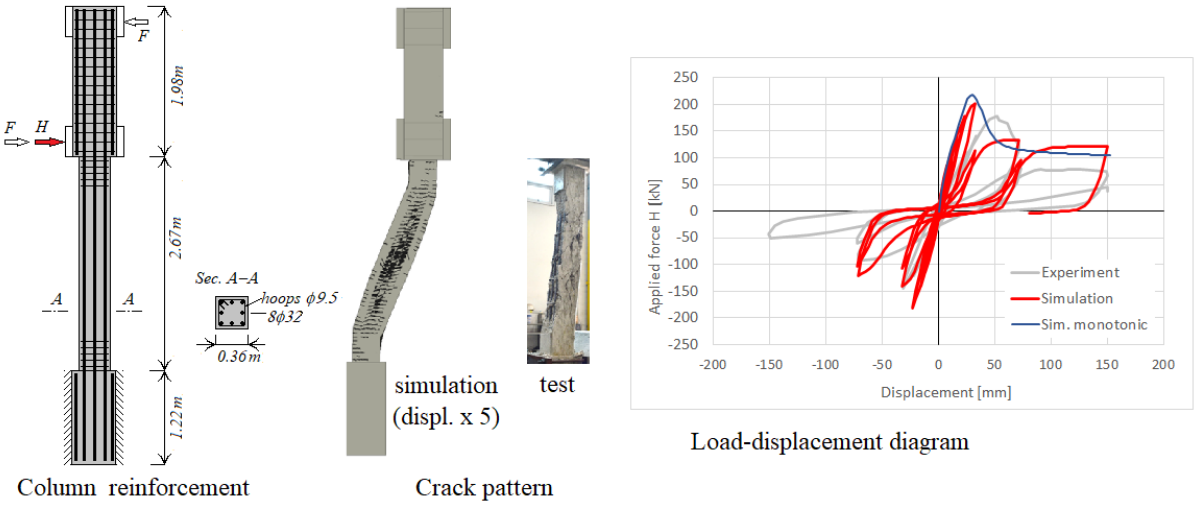


Fig. 38 Cyclic loaded reinforced concrete column.

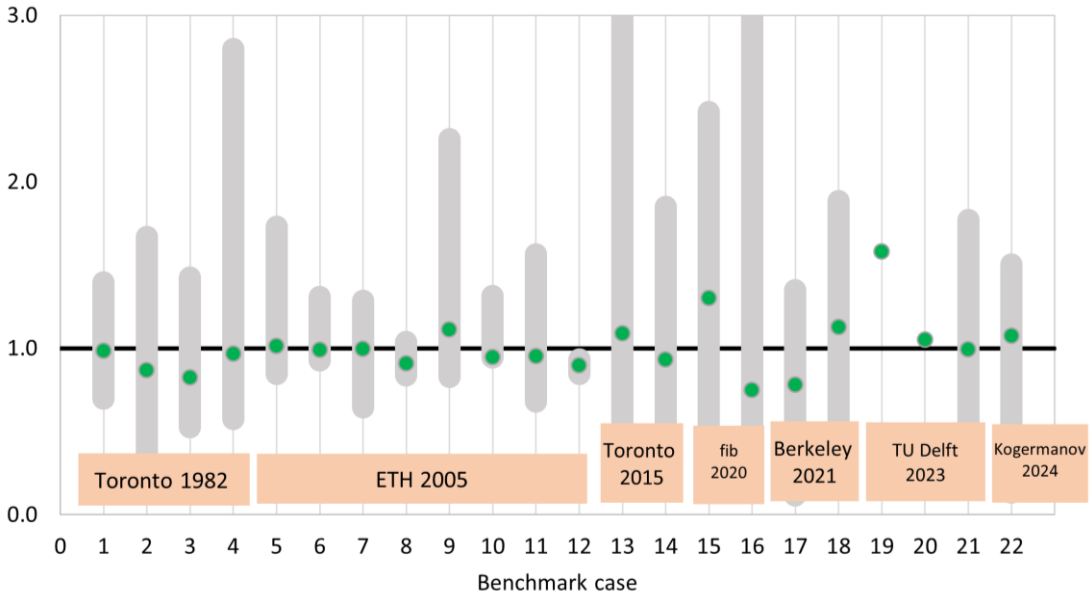


Fig. 39. Benchmark summary.

Over the past 40 years, the engineering community has shown sustained interest in improving simulation tools; however, no clear trend toward reduced uncertainty has emerged. The benchmarks primarily focus on shear or bending strength, and the wide prediction scatter reflects a limited understanding of shear failure. In addition, strength, stiffness, deformations, and crack patterns were also considered in the evaluation.

## 6.6 Parametric Generation of New-MRCS Joints

In this task, the new ATENA software module, which is developed in Task 2.4 is being applied to develop numerical models for extrapolation to other geometries and structural elements that can be applied to virtual test, verify, and develop new elements or joint connections. The objective is to develop a tool for rapid and user-friendly development of numerical models that can be used

mainly by Taiwanese partners to verify and validate structural elements or construction joints that cannot be tested since the experimental program is always expensive and time-consuming.

This initiative is closely connected to the objectives of Tasks 3.3 and 5.3, which emphasize the parametric modeling and construction sequence optimization aspects of the new ATENA module. Preliminary results and visualizations from the two ATENA modules, demonstrating the parametric input and model definition for geometric and material data, are detailed in the the module User's Manual in Section 2. This alignment ensures a streamlined approach to both development and optimization efforts.

The following figures show the user interfaces of the new software module for the parametric definition of the key dimensions of the MRCS joints as well as selected results of the generated numerical models with different geometries and dimensions of the individual structural elements, such as: column width, I-profile dimensions, spiral diameter or reinforcement type.

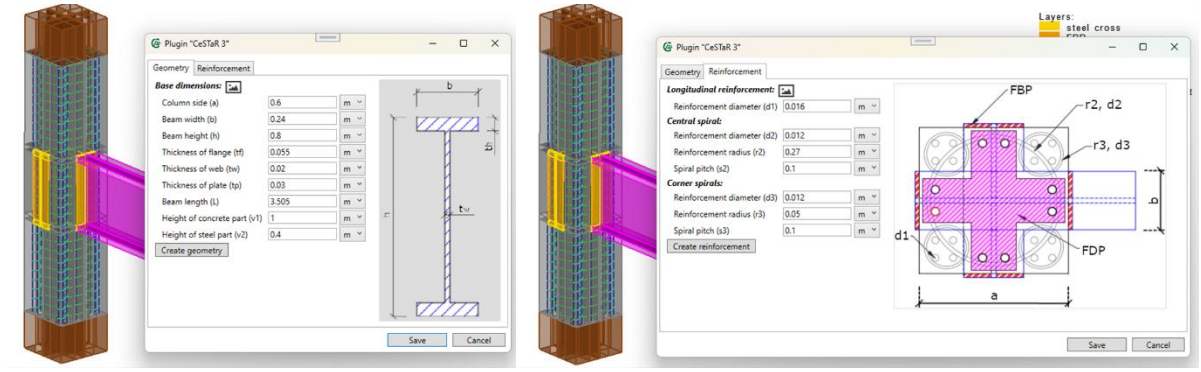


Fig. 40. Parametric definition of new MCRS structural elements, definition of steel member dimensions or main cross-sectional properties.

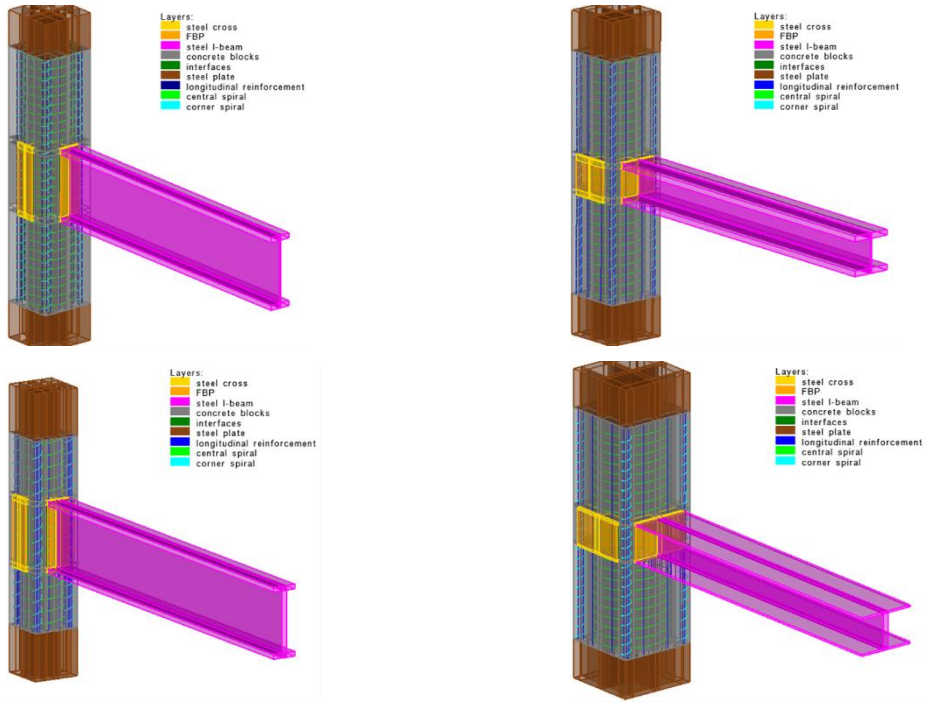


Fig. 41. Results of parametric automatic generation of numerical models of MRCS type of joints with spiral reinforcements with different dimensions and sizes of individual structural elements.

The new module enables easy customization and development of user defined scripts for the rapid generation of specialized numerical models with parametric input. An example of the script code is shown in Fig. 42 below.

```
<?xml version="1.0" encoding="utf-8" ?>
<Plugins>
  <Plugin Name="CeSTaR 3" Image="plugin.png">

    <!-- Geometry -->
    <CommentParameter TabSection="Geometry" Name="Geometry_Comment" Comment="Base dimensions:" FontWeight="bold" Image="cross_section.png"/>
    <RealParameter TabSection="Geometry" Label="Column side (a)" Name="a" Value="0.6" Unit="m" LimitRange="[0.01; 5]" StandardRange="[0.1; 2]" MarginLabel="10,0,0,0" AlignmentLabel="left" />
    <RealParameter TabSection="Geometry" Label="Beam width (b)" Name="b" Value="0.24" Unit="m" LimitRange="[0.01; 5]" StandardRange="[0.1; 2]" MarginLabel="10,0,0,0" AlignmentLabel="left" />
    <RealParameter TabSection="Geometry" Label="Beam height (h)" Name="h" Value="0.8" Unit="m" LimitRange="[0.01; 5]" StandardRange="[0.1; 2]" MarginLabel="10,0,0,0" AlignmentLabel="left" />
    <RealParameter TabSection="Geometry" Label="Thickness of flange (tf)" Name="tf" Value="0.055" Unit="m" LimitRange="[0.001; 2]" StandardRange="[0.01; 0.2]" MarginLabel="10,0,0,0" AlignmentLabel="left" />
    <RealParameter TabSection="Geometry" Label="Thickness of web (tw)" Name="tw" Value="0.02" Unit="m" LimitRange="[0.001; 2]" StandardRange="[0.01; 0.2]" MarginLabel="10,0,0,0" AlignmentLabel="left" />
    <RealParameter TabSection="Geometry" Label="Thickness of plate (tp)" Name="tp" Value="0.03" Unit="m" LimitRange="[0.001; 2]" StandardRange="[0.01; 0.2]" MarginLabel="10,0,0,0" AlignmentLabel="left" />
    <RealParameter TabSection="Geometry" Label="Beam length (L)" Name="L" Value="3.505" Unit="m" LimitRange="[0.01; 10]" StandardRange="[0.1; 5]" MarginLabel="10,0,0,0" AlignmentLabel="left" />
    <RealParameter TabSection="Geometry" Label="Height of concrete part (v1)" Name="v1" Value="1" Unit="m" LimitRange="[0.01; 10]" StandardRange="[0.1; 3]" MarginLabel="10,0,0,0" AlignmentLabel="left" />
    <RealParameter TabSection="Geometry" Label="Height of steel part (v2)" Name="v2" Value="0.4" Unit="m" LimitRange="[0.01; 5]" StandardRange="[0.1; 2]" MarginLabel="10,0,0,0" AlignmentLabel="left" />

    <ApplyButtonParameter TabSection="Geometry" SectionType="geometry" Label="Create geometry" Name="Generate_Geometry" AlignmentValue="left" ApplyFunction="generate_Geometry" JumpToNextSection="False" ShowMessageBox="False"/>

    <!-- Reinforcement -->
    <CommentParameter TabSection="Reinforcement" Name="Longitudinal_Comment" Comment="Longitudinal reinforcement:" FontWeight="bold" Image="floor_plan.png"/>
    <RealParameter TabSection="Reinforcement" Label="Reinforcement diameter (d1)" Name="d1" Value="0.016" Unit="m" LimitRange="[0.001; 0.2]" StandardRange="[0.004; 0.032]" MarginLabel="10,0,0,0" AlignmentLabel="left" />

    <CommentParameter TabSection="Reinforcement" Name="Central_Spiral_Comment" Comment="Central spiral:" FontWeight="bold" />
    <RealParameter TabSection="Reinforcement" Label="Reinforcement diameter (d2)" Name="d2" Value="0.012" Unit="m" LimitRange="[0.001; 0.2]" StandardRange="[0.004; 0.032]" MarginLabel="10,0,0,0" AlignmentLabel="left" />
    <RealParameter TabSection="Reinforcement" Label="Reinforcement radius (r2)" Name="r2" Value="0.27" Unit="m" LimitRange="[0.01; 5]" StandardRange="[0.01; 2]" MarginLabel="10,0,0,0" AlignmentLabel="left" />
    <RealParameter TabSection="Reinforcement" Label="Spiral pitch (s2)" Name="s2" Value="0.1" Unit="m" LimitRange="[0.002; 5]" StandardRange="[0.01; 2]" MarginLabel="10,0,0,0" AlignmentLabel="left" />

    <CommentParameter TabSection="Reinforcement" Name="Corner_Spiral_Comment" Comment="Corner spirals:" FontWeight="bold" />
    <RealParameter TabSection="Reinforcement" Label="Reinforcement diameter (d3)" Name="d3" Value="0.012" Unit="m" LimitRange="[0.001; 0.2]" StandardRange="[0.004; 0.032]" MarginLabel="10,0,0,0" AlignmentLabel="left" />
    <RealParameter TabSection="Reinforcement" Label="Reinforcement radius (r3)" Name="r3" Value="0.05" Unit="m" LimitRange="[0.01; 2]" StandardRange="[0.01; 1]" MarginLabel="10,0,0,0" AlignmentLabel="left" />
    <RealParameter TabSection="Reinforcement" Label="Spiral pitch (s3)" Name="s3" Value="0.1" Unit="m" LimitRange="[0.002; 2]" StandardRange="[0.01; 1]" MarginLabel="10,0,0,0" AlignmentLabel="left" />

    <ApplyButtonParameter TabSection="Reinforcement" SectionType="reinf" Label="Create reinforcement" Name="Generate_Reinforcement" AlignmentValue="left" ApplyFunction="generate_Reinforcement" JumpToNextSection="False" ShowMes

    <ApplyPython>

def generate_Geometry():

  globalParameter(equation = "a = " + str(pluginProxy.Parameters["a"].GetValue()))
  globalParameter(equation = "b = " + str(pluginProxy.Parameters["b"].GetValue()))
  globalParameter(equation = "h = " + str(pluginProxy.Parameters["h"].GetValue()))
  globalParameter(equation = "tf = " + str(pluginProxy.Parameters["tf"].GetValue()))
  globalParameter(equation = "tw = " + str(pluginProxy.Parameters["tw"].GetValue()))
  globalParameter(equation = "tp = " + str(pluginProxy.Parameters["tp"].GetValue()))
  globalParameter(equation = "L = " + str(pluginProxy.Parameters["L"].GetValue()))
  globalParameter(equation = "v1 = " + str(pluginProxy.Parameters["v1"].GetValue()))
  globalParameter(equation = "v2 = " + str(pluginProxy.Parameters["v2"].GetValue()))

  layer(name = "steel cross")
  line(startPoint = ["b/2","-b/2",0], endPoint = ["b/2","-tw/2",0])
  line(startPoint = 2, endPoint = ["b/2","tw/2",0])
  line(startPoint = 3, endPoint = ["b/2","b/2",0])
  line(startPoint = 4, endPoint = ["b/2","b/2","tf"])
  line(startPoint = 5, endPoint = ["b/2","tw/2","tf"])
  line(startPoint = 6, endPoint = ["b/2","-tw/2","tf"])
  line(startPoint = 7, endPoint = ["b/2","-b/2","tf"])
```

Fig. 42. A segment of the script header for the parametric definition of the geometries and models shown above in Fig. 40 and Fig. 41.

# 7 Conclusion

This document contains the technical as well as user's documentation of ATENA software module for the advanced nonlinear modelling and simulation of the New-MRCS concrete-steel joints developed by Taiwanese partners.

The software has been developed as a special pluggin to the simulation system ATENA. It cannot be used independently, but ATENA license is required to fully utilize this new module.

The module has been developed with the financial support of the Technology Agency of the Czech Republic under the project number TM04000013 "Virtual modelling of green concrete - structures with novel multi-spiral reinforcement and steel members".

# References


[1] Argyris JH. 1954 Energy Theorems and Structural Analysis. Aircraft Engineering and Aerospace Technology. Emerald, 1954.

[2] Bažant, Z.P. 1976. Instability, Ductility and Size Effect in Strain Softening Concrete, J. Engrg. Mech., ASCE, Vol. 102, No. 2, pp. 331-344.

[3] Bažant, Z.P. & Oh, B.H., 1983. Crack band theory for fracture of concrete. *Materials and Structures, RILEM* 16 (3), 155–177.

[4] Bažant, Z.P., Pijaudier-Cabot, G. 1987. Nonlocal continuum damage, localization instability and convergence. Journal of Applied Mechanics, ASME 55 (2), 287-293.

[5] Bentz, E.C., Vecchio, F.J., Collins, M.P. 2006. Simplified Modified Compression Field Theory for Calculating Shear Strength of Reinforced Concrete Elements. ACI Material Journal, Jul/Aug 2006.

[6] Calatrava S. 2020. The UAE Pavilion at EXPO 2020. https://aasarchitecture.com/2021/09/the-uae-pavilion-at-expo-2020-by-santiago-calatrava/

[7] Castaldo, P., Gino, D., Bertagnoli, G., Mancini, G. 2018. Partial safety factor for resistance model uncertainties in 2D non-linear analysis of reinforced concrete structures, Engineering Structures, 176, 746-762. https://doi.org/10.1016/j.engstruct.2018.09.041.

[8] Castaldo, P., Gino, D., Bertagnoli, G., Mancini, G. 2020. Resistance model uncertainty in non-linear finite element analyses of cyclically loaded reinforced concrete systems, Engineering Structures, 211(2020), 110496, https://doi.org/10.1016/j.engstruct.2020.110496

[9] Cervenka, V., Gerstle, K., 1971. Inelastic analysis of reinforced concrete panels. Part I , Theory. Part II, Experimental verification and Application. Publication IABSE 31 (11), 32-45.

[10] Cervenka, V., 1985, Constitutive Model for Cracked Reinforced Concrete. ACI Journal Nov.-Dec. 1985, 877-882.

[11] Cervenka, V., Margoldova, J. 1995, Tension Stiffening Effect in Smeared Crack Model, Engineering Mechanics, Stain Sture (Eds), Proc. 10th Conf., ASCE EMD, May 21-24, 1995, Boulder, Colorado, pp. 655-658

[12] Cervenka J, Cervenka V, Eligehausen R, 1998 Fracture-plastic material model for concrete. Application to analysis of powder actuated anchors, FraMCoS 3, Gifu, Japan, eds. H. Mihashi and K. Rokugo, Aedificatio Publishers, Freiburg, Germany, Vol. 2, pp. 1107-1116.

[13] Cervenka J. & Papanikolaou V. 2008. Three Dimensional Combined Fracture-Plastic Material Model for Concrete. *Int Journal of Plasticity*. 24:2192-2220. doi:10.1016/j.ijplas.2008.01.004

[14] Cervenka, J., Cervenka, V., Laserna S., 2014, On finite element modelling of compressive failure in brittle materials, Computational Modeling of Concrete Structures. Bicanic et al.(Eds),Euro-C 2014, St. Anton

[15] Cervenka, J., Cervenka, V., Laserna. S., 2018a. On crack band model in finite element analysis of concrete fracture in engineering practice, Engineering Fracture Mechanics, Volume 197, 2018, Pages 27-47, ISSN 0013-7944, https://doi.org/10.1016/j.engfracmech.2018.04.010.

[16] Cervenka V. Global safety format for nonlinear calculation of reinforced concrete. Beton- und Stahlbetonbau, 103, special edition. Berlin: Ernst & Sohn; 2008. p. 37–42.

[17] Cervenka V. Reliability-based non-linear analysis according to model code 2010. J fib Struct Concr. 2013;14(1):19–28.

[18] Cervenka V, Cervenka J, Kadlec L. 2018b. Model uncertainties in numerical simulations of reinforced concrete structures. Structural Concrete; 2018: 2004–2016. https://doi.org/10.1002/suco.201700287

[19] Cervenka, V., Cervenka, J., Jendele, L., 2025. ATENA Program Documentation, Part 1: Theory, Cervenka Consulting s.r.o., www.cervenka.cz

[20] Benes S., Mikolášková J., Altman T., 2025. ATENA Program Documentation, User's Manual for ATENA Studio, Cervenka Consulting s.r.o., www.cervenka.cz

[21] Cervenka, V., Rimkus, A., Gribniak, V., Cervenka, J., 2022. Simulation of the Crack Width in Reinforced Concrete Beams Based on Concrete Fracture. Theoretical and Applied Fracture Mechanics, 121 (2022) 103428. https://doi.org/10.1016/j.tafmec.2022.103428
[22] Cervenka, V., Cervenka, J., Rymes, J., Numerical simulation of concrete structures-From research to engineering application, Structural Concrete. 2024;1–22., 2024 fib. International Federation for Structural Concrete, DOI: 10.1002/suco.202300016
[23] Collins, M.P., Vecchio, F.J., Mehlhorn, G., 1985. An international competition to predict the response of reinforced concrete panels. Canadian Journal of Civil Engineering, Vol.12, 624-644, 1985.
[24] Collins, M.P., Bentz, E.C., Quach, P., Proestos, G.T., 2015. The Challenge of Predicting the Shear Strength of Very Thick Slabs. Concrete International, V.37,No.11, Nov. 2015, pp 29-37.
[25] Costa, D.D., Cervenka, V., Costa, R.G., 2018. Model uncertainty in discrete and smeared crack prediction in RC beams under flexural loads. Engineering Fracture Mechanics 199, Elsevier, (2018) 532–543.
[26] Engen M, Hendriks M., Köhler J, Øverli JA, Åldtstedt E. 2017. A quantification of modelling uncertainty for non-linear finite element analysis of large concrete structures. Structural Safety 2017; 64: 1-8.
[27] Etse G. Theoretische und numerische untersuchung zum diffusen und lokalisierten versagen in beton. Ph.D. Thesis. University of Karlsruhe. 1998
[28] EN 1990, 2002. Eurocode 0: Basis of structural design. European Committee for Standardization (CEN). Brussels.
[29] EN 1992-1-1, 2004. Eurocode 2: Design of Concrete Structures - Part 1-1: General ruels and rules for buildings. European Committee for Standardization (CEN). Brussels.
[30] *fib* MC 2010 . International Federation for Structural Concrete. Model Code for Concrete Structures 2010. Berlin: Wilhelm Ernst & Sohn; 2013.
[31] Gino D, Castaldo P, Giordano L, Mancini G. 2021. Model uncertainty in nonlinear numerical analyses of slender reinforced concrete members. Structural Concrete. 1–26. https://doi.org/10.1002/suco.202000600
[32] de Borst, R. 1986. Non-linear analysis of frictional materials. PhD Thesis, Delft University of Technology, The Netherlands.
[33] de Borst, R., Benallal, A., and Heeres, O.M. 1996. A gradient-enhanced damage approach to fracture. J. de Physique IV, C6, pp. 491-502.
[34] de Borst R, Mühlhaus HB. Gradient dependant plasticity: formulation and algorithmic aspects. Int J Numer Meth Eng. 1992;35(3):521–39.
[35] de Borst R, Rots JG. Occurence of spurious mechanisms incomputations of strain-softening solids. Eng Comput. 1989;6:272–80.
[36] Hordijk, D.A. 1991. Local approach to fatigue of concrete. PhD Thesis, Delft University of Technology, The Netherlands.
[37] JCSS Probabilistic Model Code, 2001.https://www.jcss-lc.org/jcss-probabilistic-model-code/
[38] Jeager, T., Marti, P., 2009. Reinforced Concrete Slab Shear Prediction: Entries and Discussion. ACI Journal May-June 2009. Pp. 309-318.
[39] Jendele L, Phillips DV, 1992. Finite Element Software for Creep and Shrinkage in Concrete, Computer and Structures, 45 (1), 113-126.
[40] Jirásek, M., Bažant, Z.P. 2001. Inelastic Analysis of Structures, John Willey & Sons, LTD, Baffins Lane, Chichester, England, ISBN 0-471-98716-6
[41] Lee J, Fenves, GL. Plastic-damage model for cyclic loading of concrete structures. J. Eng. Mech, ASCE. 1998;124(8):892–900.
[42] Lin CS., Scordelis A. Nonlinear Analysis of RC Shells of General Form, ASCE, J. of Struct. Eng., Vol. 101, No. 3, 1975, pp. 152–63.

[43] Ngo, D., Scordelis, A.C. 1967. Finite element analysis of reinforced concrete beams, J. Amer. Concr. Inst. 64, pp. 152-163.
[44] Menétrey, P., Willam, K.J. 1995. Triaxial failure criterion for concrete and its generalization. ACI Structural Journal 92 (3), 311-318.
[45] Moehle, J.P., Zhai, J. 2021. Blind Prediction Competition of Shear Strength for Thick Concrete Foundation Elements With and Without Shear Reinforcement. Private communication. 2021.
[46] Ngo, D., Scordelis, A.C. 1967. Finite element analysis of reinforced concrete beams, J. Amer. Concr. Inst. 64, pp. 152-163.
[47] Pramono, E., Willam, K.J. 1989. Fracture energy-based plasticity formulation of plain concrete. Journal of Engineering Mechanics, ASCE 115 (6), 1183-1204.
[48] Rashid, Y.R. 1968. Analysis of prestressed concrete pressure vessels. Nuclear Engineering and Design 7 (4), 334-344.
[49] Rots, J.G., Blaauwendraad, J. 1989. Crack models for concrete : Discrete or smeared ? Fixed, multi-directional or rotating ? Heron 34 (1).
[50] Schlangen, E. 1993. Experimental and Numerical Analysis of Fracture Processes in Concrete, Ph.D. dissertation, Delf University of Technology. 1993.
[51] Suidan, M., Schnobrich, W.C. 1973. Finite Element Analysis of Reinforced Concrete, ASCE, J. of Struct. Div., Vol. 99, No. ST10, pp. 2108-2121
[52] Swiler L.P., Giunta A.A., 2007. Aleatory and Epistemic Uncertainty Quantification for Engineering Applications. Sandia National Laboratories. American Statistical Association, July 30 - August 2, 2007 in Salt Lake, UT. https://www.osti.gov/servlets/purl/1147526
[53] Terzic, V. et al, 2021. Quasi-static cyclic test of an RC column. Private communication. Nov. 2021.
[54] Turner M., Clough RW. 1956 Stiffness and deflection analysis of complex structures. Journal of the Aeronautical Science. Vol.23, September 1956, No. 9.
[55] Vrouwenvelder A.C.W.M., 2002. Developments towards full probabilistic design codes. Structural Safety, Volume 24, Issues 2–4, April–October 2002, Pages 417-432.
[56] Zienkiewicz O.C., Cheung Y.K. 1967. The Finite Element Method in Structural and Continuum Mechanics. McGraw-Hill, London, 1967. p 274.
[57] Zienkiewicz O.C., Valliappan S., King I.P. 1969. Elasto-plastic solutions of engineering problems 'initial stress' finite element approach. Int. Journal for Numerical Methods in Engineering. Volume1, Issue1, January/March 1969, pp. 75-100 https://doi.org/10.1002/nme.1620010107