Software for Solving Problems in   
Structural Engineering

using Finite Element Analysis

FE Analysis

Prof. Dr.-Ing. Karl E. Beucke, Ettersburg

2022

**Goals and Motivation**

1. **Structural Engineering Education**

A primary goal of educating students in civil engineering for the design and execution of structures is associated with developing an understanding of the load carrying characteristics of structures under external influences. Furthermore, students should be educated in learning to understand the effects of modifications and adjustments in structural design on the behavior of a structure.

For these purposes an idealization of the structure must be developed that captures the most important design characteristics. This will be referred to as a **model**.

Today, a model is generally a **digital model** for a complete structure or individual structural elements. Digital models can be generated and analyzed using a multitude of software systems that are normally specific to a specific model definition. As a consequence, engineering projects often consist of various different models for different specific purposes.

The relatively new **BIM**-technology (**B**uilding **I**nformation **M**odel) aims to unify or at least to standardize such models into a unified project model or at least into a set of compatible partial models.

It is still current state-of-the-art that complex projects in civil engineering are prepared and executed on the basis of a multitude of Technical Documents - in large projects these can consist of thousands of documents. Although these are almost always generated today in digital formats, they are usually independent of each other with consistency simply insured by the engineers responsible for generating the information. It is safe to say that **still today no major construction project is executed on the basis of a completely consistent set of Technical Documents**, i.e. contradictions and inconsistencies in project data are inevitable. In the end, resulting problems often only emerge on the construction site and need to be resolved by experienced engineers on the site. However, quite often this will be too late in order to find an appropriate solution for the problem. Frequently, such inconsistencies lead to the often portrayed construction defects inherent in manz construction projects today.

Technical Documents in construction projects are generally prepared by draughtsmen or engineers. For these purposes they must develop a complete understanding of a construction to be completed including individual construction steps and intermediate load carrying states of construction. In addition, so-called Technical Reports need to be prepared for static and dynamic behavior of structures. These are normally prepared by Structural Engineers who have been trained in understanding and performing corresponding analyses.

At this stage the two worlds of modeling and analyzing are closely dependent upon each other. It must be ensured that any changes or modifications in a model must be synchronized with the corresponding analyses. At a minimum, consequences of changes to a model must be estimated in its consequences on subsequent analyses. Previously, it was important to decouple a model from corresponding analyses as much as possible because it was a major effort to do a complete reanalysis. Today, with powerful computers and interactive software tools available it is much easier to synchronize a model with its analysis.

Most curricula in structural engineering currently will require students to learn a number of different, specialized analysis techniques for the analysis of the behavior of structures (strength, deformation, heat transfer, etc.). These can be carried out „by-hand“ or with some simple, general software tools (e.g. spreadsheets, math tools). Most of these have a very limited scope but will allow students to follow and understand each individual step in the solution process. Powerful, generalized analysis techniques in structural engineering, however, consist of algorithms with many tedious, time consuming steps in the solution process. This fact had motivated a structural engineer, Konrad Zuse, many years ago to develop “mechanized” solution techniques for these purposes. He developed what now is called computer science and is regarded as the inventor of the computer. The use of computer software in structural engineering is now state-of-the-practice.

Later, in professional problem solving however, engineers are confronted with highly complex and complicated software systems for the solution of such problems. These systems normally have a general scope with the aim of solving even the most demanding problems in structural engineering. They are, however, not suitable for the education of engineering students as they do not support any insight in functionality and execution of its solution procedures. Such systems are sometimes referred to as “Black Box Systems”. Input data must be prepared, a solution process is started and results will be shown. On this basis it is hardly possible for future engineers to develop an understanding in the prerequisites, limitations and execution of solution algorithms in structural engineering.

The objectives of education in structural engineering should be for students to develop an understanding of the behavior of structures under various external influences and of consequences of changes in the model. This requires tools that will allow to monitor and follow each individual step in a solution process. Furthermore, such tools should support interactive changes to model definitions with immediate reanalysis and display of results in interactive graphical formats that can easily be interpreted and queried for specific results required. Consequently, such tolls must be

* **transparent** in their internal solution approaches and procedures,
* **open** without any restrictions to users in readable formats and
* **traceable** in its execution process step-by-step including all relevant intermediate results as defined by a user.

An interactive control of model parameter definitions and presentation of results by the users will mandate the availability of a modern interactive User Interface (**UI**) in engineering software that will support a flexible variety of powerful general purpose events for defining and handling engineering user interactions. This requires the availability of a

* powerful object-oriented, event driven user (**UI**) interface that will separate as much as possible the interface design from its functionality.

Today it is totally indisputable to use software in structural engineering in education as well as in professional use. Classical solution techniques that can be carried out „by-hand“ or with simple, general software tools are still relevant but only for general, rough estimates of the behavior of structures or for checking of results produced by software systems.

1. **Software Systems for Structural Engineering problem solving**

Most curricula in Structural Engineering require students to analyze predefined structural models. A given design is either analyzed “by-hand” or using a specific software solution. For these purposes simple idealizations like bending beams or frame structures with specific support conditions and external influences, e.g. loads, are being utilized.

Originally, the process of setting up a computational model for practical purposes was tedious and time consuming. The execution of the analysis could not be performed on a “Personal Computer” but rather had to be done in a Computer Center which was mot always available in house. The interpretation of results was also cumbersome and took considerable time, because long alphanumeric printed lists had to be evaluated. These limitations quite often had the consequence that engineers hesitated to change an existing analysis model in its basic definitions but rather adjust requirements. Sometimes this approach was regarded as analyzing an existing design to meet requirements.

The performance of computers and the functionality of engineering software originally did not allow for the analysis of a structural design model on the fly for decisions on structural alternative designs to be made. Structural analysis consists of the steps

* idealization and modeling of a structural design
* setting up an appropriate model for computational analysis
* executing the analysis and
* interpretation and evaluation of results.

In this process it should be supported to easily change a computational model according to a design change and analyze the consequences in structural behavior and possibly perform the necessary steps iteratively until requirements are met.

In consequence a tool would be needed that allows the user to

* base a computational model for structural analysis directly on structural design definitions and changes,
* analyze the model instantaneously and
* evaluate computational results relevant for a structural design alternative.

Feasible design alternatives in a structural design include for example additional or removed structural elements, alternative support conditions, different or adjusted material properties or cross sectional changes for beams. Such alternatives must easily and interactively be changeable. Resulting changes is structural behavior must be visually illustrated and visually comparable to the structural behavior of previous design decisions.

With such a tool students could focus on “play through” various design alternatives, evaluate alternative design decisions and compare different solutions in its consequences on structural behavior instead of focusing on the mechanics of converting structural model definitions into a computational model.

The software solution proposed and included aims to close the gap between simple and understandable structural analysis examples and complex and closed professional software solutions. It aims to

* provide general solutions procedures for complex computational analysis in Structural Engineering, that
* allows students complete, unrestricted, detailed insights into the functionality and execution of solution algorithms with access to any intermediate results and that
* conceptionally support standard problem classes like First Order Systems (heat transfer), Second Order Systems (strength and deformation analysis) as well in 2D as in 3D with stationary/instationary and static/dynamic analysis.

1. **Transparency, Openness and Traceability**

The goals outlined above require that any individual steps in solution algorithms must be absolutely ***transparent*** to the user, i.e. the utilization of software or software libraries and tools that are only available in machine readable, executable formats is not acceptable. Such software does not support or even allow for any insights into internal solution approaches and strategies nor is it possible to monitor progress and intermediate results.

For engineers there exists a natural boundary between system software that is completely independent of application software and application software that is specific to a specific application domain like structural engineering. These boundaries, however, have shifted over the years considerably. While originally engineers had to include general functionalities into their solutions, it is now supported by computer science and expected by users of modern software solutions to provide powerful and standardized functionality for many general tasks. These include specifically tasks like file access, data handling and querying, network communication, graphics and geometry and user interfaces. As components of application software they are, however, not anymore a transparent part of application solutions much like operation systems in previous solutions. Such components are sometimes referred to as *Middleware*.

Transparency is sometimes also hindered or actively prevented by legal restrictions regarding copyright, right of use and exploitation rights. Transparency thus requires software to be free of such legal restrictions. Engineering software solutions for the purposes outlined above must consequently be available in ***open*** formats.

**Transparency** and **Openness** are necessary prerequisites for **Traceability** of all individual steps in a solution process, i.e. the user of a software solution must be able to follow and trace each separate step in a solution process including intermediate states and results.

For these purposes, a - not indisputable - approach was chosen for the implementation of solution algorithms in Structural Engineering. Generally in academic education, application software is required to be available in open and freely accessible formats - so called “Open Source” software. This requirement originally included even software for operating systems of computers. Consequently, in education and research the use of operating systems like UNIX or LINUX was favored.

Application programming in academics was mostly based on programming languages and integrated development environments (IDE) like C or Java for the same purposes.

Professional applications in construction industry however were mostly based upon proprietary operating systems like MS-Windows and programming languages like C++ which were both not open and copyright protected.

This discrepancy lead to frequent complaints by professional employers in construction industry that academic education was “detached from practical necessities and realities”.

The discrepancy was sometimes even aggravated because the development of proprietary software developed - sometimes considerably - more dynamically than the development of open systems. This development often changed perceptions. Today, in academia as well as in industry the use of proprietary Operating Systems is widely accepted. Even in application software development the use of proprietary IDEs (Integrated Development Environments) is more and more accepted. This has a lot to do - in the opinion of the author - with the growing importance of standardization and *Middleware* software.

Thus, for the purposes of this development a relatively new and modern IDE was chosen with C# as computer language and MS-Visual Studio as IDE which are both provided by Microsoft and which are both restricted and copyright protected. C# is a modern development by Microsoft that was first published in the year 2001 after Microsoft was restricted from licensing Java. As a new, modern development **C#** is

* consequently based on the concepts of **Object-Orientation** with
* **.NET Framework** as Middleware for File Access, Data Management (**L**anguage **IN**tegrated **Q**uery - LINQ), Network Communication, Geometry and Graphics, with
* a modern **U**ser **I**nterface (**UI**) development **W**indows **P**resentation **F**oundation (**WPF**) that fully blends into the MS-Windows environment with a far-reaching separation of design from functionality by the use of an interpretative E**X**tensible **A**pplication **M**arkup **L**anguage (**XAML**) and which
* is supported by a modern professional IDE under the name MS-**Visual Studio**  
  that is free to use with unrestricted access for educational purposes.

Project development in this environment can be made easily accessible in source code for students.

With these provisions, requirements for transparent and open software developments in engineering that are traceable by students in all its steps and results with any intermediate stages.

1. **Concept Study for an Integrative Structural Design**

Current practical applications in Structural Engineering are characterized by the fact that they are carried out on the basis of an increasingly complex and extensive set of digital data. The development of so-called “Building Information Modeling - **BIM**” has even led to a strong ascent in this general tendency.

Not many years ago, computing power availability and storage capacity were severely limiting the use of digital data models. Even then, setting up a design model, deriving a computational model and performing the analysis was a tedious and time consuming process that could take up to days and weeks for large projects.

In the year 2012 the authors *F. Gerold, K. Beucke, F. Seible* have published an article in the „Journal of Computing in Civil Engineering“ under the title „Integrative Structural Design“. In this article the idea of an “Integra**tive** Structural Design” was formulated as opposed to an “Integra**ted** Structural design”. *Integrated* was understood as a largely automated structural design based upon a fully consistent set of data whereas *Integrative* was formulated as a process where an engineer is fully in control of design decisions based upon a flexible set of tools allowing for interactively testing and trying different design alternatives in real time. Interactive User Interfaces allow for easy and transparent model modifications and computing power and storage capacity are powerful enough to perform all tasks in this process even for large models in real time.

The article demonstrates this process for an earthquake analysis of a complex frame structure. A dynamic analysis on the basis of recorded acceleration data with 30,000 time steps is carried out in seconds of time and the resulting design data can be visually displayed and evaluated immediately. Specifically in case of earthquake analysis this can prove very valuable as previously such analyses were very time consuming and were difficult to judge in their results and consequences. Intuitive design decisions like member strengthening are not always helpful and sometimes even lead to negative consequences under these loading conditions.

The adoption of a fully “Integrative Design” in the sense described above is probably still a long way to go as it requires not only different tools but also a different state of mind and education for structural engineers. However, it is hoped that this application software can serve as a concept study for the viability of the approach and for changes in education of engineers.

Fundamentally, the functionality of all analyses is separated into

* read, edit and save design model data
* display model data in interactive tabular as well as visual formats
* analyze the model data
* display the results of analyses in interactive tabular and visual formats.

Different model alternatives and design variations can be displayed and compared in parallel windows. Resulting design variables can be queried and displayed in interactive tabular or visual formats.

Conventionally, the definition of model data is most conveniently prepared in a simple text file of applicable input data. A number of simple models is available in an associated input-directory of the application. A simple text editor is included for the presentation and editing of model data.

After reading input data of a model, the complete model data definitions can be displayed in various tables where individual table data can be interactively edited and amended. This will lead to consistent changes of internal model data for subsequent model analyses. Any resulting state of a model can then be saved into a new input file.

Model analysis can either be initiated explicitly by the user or implicitly in the software when needed, e.g. when results of an analysis are to be displayed. Implicit analysis is not supported for instationary/dynamic analysis which must be initiated explicitly by the user.

Graphic display of data resulting from an analysis can be explored interactively. Resulting deformations can be displayed and “Clicking” individual structural members and nodes will result in displaying what is called a “pop-up window” with a textual representation of resulting design variables. “Right Clicking” will close any pop-up window.

**User Manual**

**Definition of data for a new Model:**

Basis for any engineering analysis is the definition of a **model** defined by the corresponding **model data** that determine **model behavior**.

In Finite Element analysis model data consists of **Nodes** determining model geometry, **Elements** determining model topology, external **Influences** (loading) on a model and **boundary conditions**, e.g. fixed temperature, deformation constraints and supports.

The definition of a complete set of data for a specific model is conventionally prepared in a separate text file as this is often the most efficient and simple way as compared to user dialogues.

Application areas are selected for Structural Analysis, Heat Transfer Analysis and Elasticity Analysis. Structural Analysis is of central importance for structural engineering but is often treated as a special case in engineering analysis with special conventions and solution approaches. In this approach it is treated on the same basis as and consistent with any other engineering analysis. In the context of this application, it is presented on the same basis for 1D and 2D model idealizations with 2 (trusses) as well as 3 (beams, frames) degrees-of-freedom per node or element. Hinges at the ends of elements and at nodes define reductions of degrees-of-freedom.

The most basic engineering analysis is Heat Transfer aAnalysis that is presented as a simple example for the general approach identical for all other analyses. Implementations are presented for 1D and 2D analysis with a single degree-of-freedom. Structural Analysis is presented as a specific case of strength analysis derived from general elasticity theory. In practical applications, both are commonly treated together. The general case of Elasticity Analysis is often subject of higher education at a later stage of engineering education as theory as well as interpretation of results is more demanding. Resulting variables of elasticity analysis are strains and stresses that are not immediately useful for practical solutions as practical solutions are mostly based on deformations and internal force definitions.

Engineering analyses of physical properties is commonly carried out for external influences that are independent of time variation. This is denoted as static analysis or in heat transfer as stationary analysis. In case, external influences are dependent on time variation like time dependent heat variation, wind forces or ground accelerations, time stepping solution procedures will be needed for instationary heat transfer (1st order differential equation) or dynamic structural analysis (2nd order differential equation).

Some simple examples are provided in subdirectory “input” for each application area with subdivisions for instationary and dynamic analysis. These are

**input HeatTransfer Instationary**

**StructuralAnalysis Dynamics**

**Elasticity**

All input files are characterized by file ending “.inp”. Empty input file examples with commented templates are provided in files “1Template.inp”.

**Presenting and Editing existing and Adding new model definitions:**

Various different methods and tools are available for displaying, editing and deleting existing model definitions interactively and new definitions may be added when needed. Thus, enabling and supporting users for **Interactive Model Exploration**s as outlined above.

The complete set of data defining a specific model can be displayed in alphanumeric data tables (**DataGrids**) for a **tabular model representation**. Any row in a data table can be selected, edited or deleted preserving consistency with internal model definitions by not only changing the visual presentation but also by propagating changes to the internal model data. Deleting definitions is achieved by selecting a complete row in a data table and pressing the Delete-button. Editing definitions is achieved by selecting individual cell items in a row and editing its contents. For example, nodal coordinates could be changed thus changing the geometry of a model or elements could be deleted changing the topology of a model. Likewise, loading and support conditions can be changed interactively. The new state of a model can subsequently be inspected by displaying or visualizing newly updated model definitions in another window.

Additional, new model definitions (Nodes, Elements, Material, Loading, Support Conditions) can be created by “double clicking” the associated data table (DataGrid). This will initiate a corresponding user dialog for entering the associated data required for a specific new data definition.

Often it is easier and more illustrative to inspect model definitions in visual representations. For these purposes, model definitions can be displayed in addition or parallel to tabular formats in visual, graphical forms. Both options (**show/visualize model data**) are available from the main menu.

Visual representations of corresponding model data are displayed in a separate window and additional information on model definitions can be shown as additional graphical or textual information via specific “pop-up windows” that are initiated by “Clicking” a specific item in a model, e.g. a Node or an Element. “Right-clicking” will close a pop-up window.

Any changes of model data definitions made during run-time of the application will only be transient, i.e. they are discarded when the application is terminated. Permanent changes to model data must either be made by editing a corresponding input-file and saving it under a given name or by saving a current state of model definitions to a new input file. Both have to be initiated explicitly by a user from the menu (**read/edit/save model data**).

The resulting new model behavior can be recomputed based upon the changes made and the current state of a model is analyzed (**Static Analysis/Dynamic Analysis**). Corresponding results can be displayed in tabular and/or visual formats.

**Inspecting results of model analysis:**

Another main idea and intention of this implementation are provisions for an interactive exploration of results of an analysis.

For this purpose, the essential results of an analysis can be displayed in separate data tables or they can be illustrated in visual form.

Primary results of an analysis are the *primal variables*, e.g. nodal temperatures and nodal deformations. The corresponding reactions at boundary conditions are computed and stored as dual variables. For an inspection of results of an analysis however, additional information is needed that can be derived from the primal and dual variables, e.g. heat flow, internal forces. All of this information can be displayed in data tables and/or visually in separate windows.

The presentation of results in data tables allows for a quick overview over relative values of results and for quickly identifying maximum values. The visual representation is helpful in localizing results and distribution of results in a model. The visual, graphical representation is furthermore supported by “pop-up windows” that are initiated by “left clicking” graphical and text elements. “Pop-up windows” will display additional detailed information for the elements selected. This will include derived information that needs to be computed like internal forces in beams and trusses. “Right clicking” will close the corresponding “pop-up”.

**Program Download via Git:**

[**https://github.com/KarlBeucke/FE-Analysis-App**](https://github.com/KarlBeucke/FE-Berechnungen)

contains an executable application and an associated setup-file for installation under MS-Windows. The location of the download should be in the standard directory *Documents/* with a subdirectory *Documents/FE Analysis*. Additional subdirectories are provided for some simple examples for each application area under *Documents/FE Analysis*/input and for this documentation *Documents/FE Analysis*/*documentation*.

[**https://github.com/KarlBeucke/FE-Analysis**](https://github.com/KarlBeucke/FE-Berechnungen)

contains a complete project under MS-Visual Studio with all program files in source code including a library for Finite Element Analysis *(FE Library*). The Finite Element library contains general standard functionality that is independent of any specific application like matrix algebra, equation solver, time step solver, general model information like geometric Nodes and abstract model information with the complete class hierarchy.

The solution file (*FE Analysis.sln*) can be started directly in MS-Visual Studio.

The project can also be started in *Debug Mode* where “*break points*” and “*watch points*” can be defined thus allowing users to follow application execution step-by-step including monitoring intermediate results in a solution process.

**Program Start:**

The main windows for the application simply contains three different application areas

* Structural Analysis
* Heat Transfer Analysis
* Elasticity Analysis

The general solution procedure is identical for each application area with

* Model data read, edit and save
* Model data show and visualize
* Model analysis
* show and visualize results of the analysis

Methods of analysis available include **static and stationary analysis** and depending on the application area in addition **dynamic and/or instationary analysis** with the **eigensolution** of the physical problem, i.e. the eigenvalues and eigenvectors of the system.

This will require additional functionality for

* instationary/dynamic Model Data show  
  ------------------------------------
* Eigensolution analyze
* Eigensolution Results show
* Eigensolution Results visualize  
  --------------------------------------
* instationary Analysis/dynamic Structural Analysis  
  --------------------------------------
* instationary/dynamic Analysis Results show
* time dependent model states visualize
* Nodal Time History visualize

**Model data for instationary/dynamic analysis** are again **read from a text file** which can be edited and be saved under a user defined name.

As described above, **model data can be interactively deleted, edited and new definitions be added**. Rows in data tables can be deleted, individual cell items can be edited and new definitions can be added by double clicking an associated data table. Again it is important to note that model data are consistently adjusted in the backgound.

Results of model analysis can be displayed in separate windows and the **resulting new state of a complete model at a specific time step** can be displayed.

Results of time dependent model analysis can also be displayed as **nodal time histories** over the complete duration of an analysis.

**Definition of Model Data and Model Analysis**

Model data for Finite Element Analysis consist primarily of **Nodes**, defining the geometry of a model and the number of degrees-of-freedom for nodal deformation. Different types of nodal connections consist of **Elements**, defining the topology and behavior of a model. Predefined behavior at boundaries of a model are defined by **Boundary Conditions** specifying predefined behavior like support conditions of a structure or heat flow. **External Influences** on a model like forces or heat sources are often referred to as **Loads** acting upon a model.

A system matrix is allocated by traversing all nodes with the associated degrees-of-freedom. Element matrices are evaluated on the basis of model information by traversing all element definitions assembling all element matrices into the system matrix. External influences are assembled into a right-hand-side load vector and predefined boundary conditions are assembled in the vector of system unknowns. Thus, a l**inear system of equations** is set up and solved for the values of the **system unknowns** at the nodes as primal variables and for the **system reactions** at predefined boundary conditions as dual variables.

The resulting **state of a computational model** (e.g. internal forces, heat flow) can be derived on the basis of the system unknowns and reactions for each element of the model.

**Time-dependent analyses** of a model (dynamic, instationary) are required for external influences on models that are varying over time like earthquakes, wind or heat radiation. Different **time-stepping solvers** are available for these purposes. This solution approach is referred to as **time domain analysis**. Alternatively, the **eigensolutions** of a system can be computed (eigenvalues and eigenvectors) and the system response be evaluated via superposition of eigensolutions. This is referred to as **frequency domain analysis**.

**Defining in and Reading Model Data from a text file**

By default, model data are defined in a text file with the extension .*inp* (*input*) and read from an input-directory in the directory of the associated application. The storage location can also be selected specifically by the user.

Any arbitrary text editor can be used for definition of data in the text file. A very simple editor is provided with this application in order to be independent of any additional installation.

Individual **Keywords** define the type of model definitions to be entered. Keywords are **followed by an arbitrary number of text lines** each defining the data needed for a corresponding new model definition thus defining an input area. An empty line will terminate each input area, i.e. new definition of data associated with the preceding keyword.

Individual **items** in each input-line are **separated by Tab-keys**.

Model data definitions can also be defined and extended by “double clicking” tables in an associated tabular representation (DataGrid).

**Input of Application specific Model Definitions**

A complete model is identified under a specific **Permanent Unique Identifier** (Name) that is independent of run-time. All individual model definitions are also characterized by permanent unique identifiers, e.g. n1, e1, etc..

Access to any model definition is solely organized via the unique identifiers, meaning that references to model definitions can be established independent from time of instantiation of a definition and even before it is defined. References to model data are exclusively resolved only immediately before a new analysis is started. This allows for interactive changes of model definitions and consistent uses of changed definitions in a new analysis.

Only at first time of access to a specific definition, necessary relations are resolved and access established to internal storage.

**List of available keywords:**

Keyword ***Model Name***is followed by a single line containing a unique name for the model to be established

Keyword ***Spatial Dimension***is followed by a single line containing 2 integer values separated by a Tab for the **spatial dimension** of the model (1D, 2D, 3D) and for the **number of degrees-of-freedom** of nodes in a model. In Structural Analysis the number of degrees-of-freedom of a specific node can be reduced thus accounting for hinges.

Keyword ***Nodes***is followed by an arbitrary number of lines each containing 2 to 4 items for a unique name for the node followed its nodal coordinates. A single item in a line indicates that the number of degrees-of-freedom of all following nodes is changed accordingly.

In its most simple way, a 2D frame with 4 nodes and a hinge at node3 is defined by:

*Model Name*

*Frame*

*Spatial Dimension* resp. with nodal hinge  *Spatial Dimension*

*2 3 2 2*

*Nodes*

*n1 0 0  
n2 0 4  
2*

*n3 3 4*

*3  
n4 3 0*

The model name is defined as “*Frame*”. The spatial dimension is 2D with 3 degrees-of-freedom for x,y displacements and rotation. Node *n3* has only 2 degrees-of-freedom indicating a hinge.

**Generating Nodes in a Model**

Regularly distributed Nodes can be generated. The following functionality is available for these purposes:

Keyword ***Nodes Group***is followed by a single line with with a text initializer for all Nodes in that group. Following this line is an arbitrary number of lines each containing coordinates for the corresponding node. Unique identifiers can be generated with a maximum of 100 per direction. Nodes n0000 with coordinates (0;3) and n0001 with coordinates (0;2) can for example be generated by

*Nodes Group*

*n*

*0 3 // n0000 with coordinates (0;3)*

*0 2 // n0001 with coordinates (0;2)*

Keyword ***Equidistant Node Mesh***has three options for 1D, 2D and 3D each followed by a single line with with a text initializer for all Nodes in that mesh, the starting coordinates in the mesh and the number of repetitions. Unique identifiers will be generated with a maximum of 100 per direction, Nxxzzyy

*Equidistant Node Mesh*

*N 0. 2. 3*

generating 3 nodes in 1D at a distance of x=2:

N00 (0), N01 (2), N02 (4)

*A 0. 1. 3 1. 1. 3*

*generating a mesh of 3\*3 nodes in 2D starting at coordinates (0;1) at a distance of 1. in x and y:*

*A0000 (0;1) A0100 (1;1) A0200 (2;1)*

*A0001 (0;2) A0101 (1;2) A0201(2;2)*

*A0002(0;3) A0102 (1;3) A0202 (2;3)*

*B 0. 1. 3 1. 1. 3 2. 1. 3*

*generating a mesh of 3\*3\*3 nodes in 3D starting at coordinates (0;1;2) at a distance of 1. in x, y and z:*

*B000000 (0;1;2) B010000 (1;1;2) B020000 (2;1;2) constituting an x,y plane at z=2*

*B000100 (0;2;2) B010100 (1;2;2) B020100 (2;2;2)*

*B000200 (0;3;2) B010200 (1;3;2) B020200 (2;3;2)*

*B000001 (0;1;3) B010001 (1;1;3) B020001 (2;1;3) constituting an x,y plane at z=3*

*B000101 (0;2;3) B010101 (1;2;3) B020101 (2;2;3)*

*B000201 (0;3;3) B010201 (1;3;3) B020201 (2;3;3)*

*B000002 (0;1;4) B010002 (1;1;4) B020002 (2;1;4) constituting an x,y plane at z=4*

*B000102 (0;2;4) B010102 (1;2;4) B020102 (2;2;4)*

*B000202 (0;3;4) B010202 (1;3;4) B020202 (2;3;4)*

Keyword ***Variable Node Mesh***is followed by an arbitrary sequence of 2 lines with the 1st line containing an arbitrary number of distances between nodes followed by a 2nd line with a text initializer for all Nodes followed by the starting coordinates which can be either 1, 2 or 3 depending on the dimensionality of the nodes (1D, 2D, 3D). Multiple meshes can be generated.

*Variable Node Mesh*

**1D** *0. 1. 3. 6. generating 4 nodes in 1D starting at coordinates (0) and   
 X 0. coordinates 0, 1, 3, 6*

*X00 (0) X01 (1) X02 (3) and X03 (6)*

**2D** *0. 1. 3. 6. generating 4\*4 nodes in 2D starting at coordinates (0;0) and   
 Y 0. 0. coordinates 0, 1, 3 and 6:*

*Y0000 (0;0) Y0100 (1;0) Y0200 (3;0) and Y0300 (6;0),*

*Y0001 (0;1) Y0101 (1;1) Y0201 (3;1) and Y0301 (6;1),*

*Y0002 (0;3) Y0102 (1;3) Y0202 (3;3) and Y0302 (6;3),*

*Y0003 (0;6) Y0103 (1;6) Y0203 (3;6) and Y0303 (6;6)*

**3D** *0. 1. 3. 6. generating 4\*4\*4 nodes in 3D starting at coordinates (0;0;0) and   
 Z 0. 0. 0. coordinates 0, 1, 3 and 6:*

*Z000000 (0;0;0) Z010000 (1;0;0) Z020000 (3;0;0) and Z030000 (6;0;0),*

*Z000100 (0;1;0) Z010100 (1;1;0) Z020100 (3;1;0) and Z030100 (6;1;0),*

*Z000200 (0;3;0) Z010200 (1;3;0) Z020200 (3;3;0) and Z030200 (6;3;0),*

*Z000300 (0;6;0) Z010300 (1;6;0) Z020300 (3;6;0) and Z030300 (6;6;0)*

*Z000001 (0;0;1) Z010001 (1;0;1) Z020001 (3;0;1) and Z030001 (6;0;1),*

*Z000101 (0;1;1) Z010101 (1;1;1) Z020101 (3;1;1) and Z030101 (6;1;1),*

*Z000201 (0;3;1) Z010201 (1;3;1) Z020201 (3;3;1) and Z030201 (6;3;1),*

*Z000301 (0;6;1) Z010301 (1;6;1) Z020301 (3;6;1) and Z030301 (6;6;1)*

*Z000002 (0;0;3) Z010002 (1;0;3) Z020002 (3;0;3) and Z030002 (6;0;3),*

*Z000102 (0;1;3) Z010102 (1;1;3) Z020102 (3;1;3) and Z030102 (6;1;3),*

*Z000202 (0;3;3) Z010202 (1;3;3) Z020202 (3;3;3) and Z030202 (6;3;3),*

*Z000302 (0;6;3) Z010302 (1;6;3) Z020302 (3;6;3) and Z030302 (6;6;3)*

*Z000003 (0;0;6) Z010003 (1;0;6) Z020003 (3;0;6) and Z030003 (6;0;6),*

*Z000103 (0;1;6) Z010103 (1;1;6) Z020103 (3;1;6) and Z030103 (6;1;6),*

*Z000203 (0;3;6) Z010203 (1;3;6) Z020203 (3;3;6) and Z030203 (6;3;6),*

*Z000303 (0;6;6) Z010303 (1;6;6) Z020303 (3;6;6) and Z030303 (6;6;6)*

**Input for Structural Analysis**

The implementation of elements for Structural Analysis is basically rstricted to 2-dimensional truss and beam elements with or without hinges, i.e. the space dimensionality is 2 and the number of degrees-of-freedom is 2 or 3. Hinges in the structure are defined by either restricting nodal degrees-of-freedom to 2 or by selecting an element with a hinge at an end.  
Keyword

* ***Beam***

is followed by an arbitrary number of lines each containing text identifiers for the element name, 2 identifiers for start and end node, identifier for the element element cross section and for the element material, e.g.

*Beam*

*Col00 F0 G0 c0 m0*

* ***CrossSection***

is followed by an arbitrary number of lines each containing a text identifier for the cross section name, the cross section area and optionally the moment of inertia, e.g.

*CrossSection*

*c0 0.18 0.0054*

* ***Material***

is followed by an arbitrary number of lines each containing a text identifier for the element material name, for Young’s modulus and optionally 1 to 5 values for Poisson’s ratio, specific mass, and spring stiffnesses in x,y and r required for spring elements, e.g.

*Material*

*m0 2.1e7 0.22 0.175*

* ***NodeLoad***

is followed by an arbitrary number of lines each containing a text identifier for a nodal load, the corresponding node and 1 to 3 values for load value in x,y and z, e.g.

*NodeLoad*

*NodeLoad1 U3-3 1000 0 0*

* ***Support***

is followed by an arbitrary number of lines each containing a text identifier for a nodal support, the corresponding node, a text value with “xyr” for a fixed support in x,y and/or rotation and optionally 3 values for predefined displacements/rotation in x, y and r, e.g.

*Support*

*Support1 F0 xy*

These are the most fundamental functionalities required for beam theory. Furthermore supported are functionalities with **keywords** for

* ***Truss***

followed by an arbitrary number of lines each containing a text identifiers for the name of the truss element, the 2 corresponding nodes, the truss cross section and the truss material, e.g.

*Truss*

*e00 n0 n1 c1 iso*

* ***BeamHinged***

followed by an arbitrary number of lines each containing a text identifiers for the name of the hinged beam element, the 2 corresponding nodes, the beam cross section, the beam material and an integer value for a hinge at the start (1) or the end (2) node of the beam, e.g.

*BeamHinged*

*e2 n1 n2 c0 m0 2*

*e3 n2 33 c0 m0 1*

* ***SpringElement***

followed by an arbitrary number of lines each containing text identifiers for the name of the spring element, the corresponding node and the spring material values, e.g.

*SpringElement*

*rotationalSpring n1 m1*

The material identifier must be defined under the material keyword with an identifier for the spring support and additional values for spring stiffnesses in x,y and/or phi, e.g.

*Material*

*m1 rotationalSpring 0 0 100*further Keywords available for loading include

***PointLoad***

followed by an arbitrary number of lines each containing text identifiers for the PointLoad, the corresponding element, load values in x and y and load application point in percentage of element length (here: 50%, i.e. in middle of element) e.g.

*PointLoad*

*P1 Bm10 0 -500 0,5*

* ***LineLoad***

followed by an arbitrary number of lines each containing text identifiers for the LineLoad, the corresponding element and load values at start and end of the element in x and y

*LineLoad*

*LineLoad1 Bm10 0 -200 0 -200*

**Input Data for dynamic Structural Analysis**

the following **Keywords** are available for dynamic Structural Analysis:

* ***Eigensolutions***

is followed by a line with an identifier for the Eigensolution and the number of eigensolutions to be considered, e.g.

*Eigensolutions*

*2DOFEigen 2*

* ***TimeIntegration***

is followed by a line with an identifier for the time integration, maximum time of analysis, duration of time steps, an integer value for the method to be used (1: Newmark, 2: Wilson Theta und 3: Alfa) and the integration parameters for the specific method selected, e.g.

*TimeIntegration*

*SixDOFGroundExcitation 125 0,4 1 0,25 0,5*

* ***TimeDependentNodeLoads***

followed by a line with a text identifier for the load, identifier for an associated node (or “*ground*” for ground excitation) and load direction, i.e. number of degree-of-freedom, e.g.

*tL0 n2 0*

alternatively, instead of specifying a node, *“ground*” may be specified for ground excitation

*tL1 ground 0*

subsequent lines will define time dependent loading either by

* reading input data from file (file selection dialog)  
  *File*
* defined by a piecewise linear sequence of pairs of values for <time; load>  
  *0;0 0.8;1 1.6;0 3.2;-1 4.8;0 5.6;1 6.4;0*
* input defined by amplitude, frequency and phase angle for harmonic excitation  
  *1 0.03 0*
* ***InitialConditions***

is followed by an arbitrary number of lines each containing an identifier for the associated node followed by 2 values for the initial deformation and velocity at each degree-of-freedom (here: 1 degree-of-freedom), e.g.

*InitialConditions*

*k0 1 0*

*k1 0,9 0*

* ***Damping***

followed by a line with a uniform modal damping ratio for all Eigenstates or a sequence of values for each Eigenstate considered

*Damping*

*0.02*

**Input Data for Heat Transfer Analysis**

The implementation of elements for heat transfer analysis is focused on elements with 2, 3 or 4 nodes in 2D and an element with 8 nodes in 3D. The number of nodal degrees-of-freedom is 1.

* ***Elements2D2Nodes***

is followed by an arbitrary number of lines each defining a unique element identifier, 2 identifiers for the associated nodes and 1 identifier for the element material, e.g.

*Elements2D2Nodes*

*e0 n00 n01 iso*

* ***Elements2D3Nodes***

is followed by an arbitrary number of lines each defining a unique element identifier, 3 identifiers for the associated nodes and 1 identifier for the element material, e.g.

*Elements2D3Nodes*

*e0 n00 n01 n02 iso*

* ***Elements2D4Nodes***

is followed by an arbitrary number of lines each defining a unique element identifier, 3 identifiers for the associated nodes and 1 identifier for the element material, e.g.

*Elements2D4Nodes*

*e0 n00 n01 n02 n03 iso*

* ***Elements3D8Nodes***

is followed by an arbitrary number of lines each defining a unique element identifier, 8 identifiers for the associated nodes and 1 identifier for the element material, e.g.

*Elements3D8Nodes*

*e0 n00 n01 n02 n03 n04 n05 n06 n07 iso*

* ***Material***

is followed by an arbitrary number of lines each defining a unique material identifier, 1 value for the element conductivity and optionally an additional value for (material density \* conductivity). In case of 3D analysis 3 different values may be defined for material conductivity in x-,y-,z-direction.

*Material*

*iso 5 1*

* ***NodeLoads***

is followed by an arbitrary number of lines each defining a unique identifier for the node load, 1 identifier for the associated node and 1 value for the heat load at that node, e.g.

*NodeLoads*

*nl0 n05 5*

* ***LineLoads***

is followed by an arbitrary number of lines each defining a unique identifier for the line load, 2 identifiers for the associated nodes and 2 values for the linearly distributed heat loads at the 2 nodes, e.g.

*LineLoads*

*ll0 n05 n06 5 10*

* ***ElementLoads3***

is followed by an arbitrary number of lines each defining a unique identifier for the element load, 1 identifier for the associated element and 3 values for the heat loads at the 3 element nodes, e.g.

*ElementLoads3*

*el0 e0 30 30 30*

* ***ElementLoads4***

is followed by an arbitrary number of lines each defining a unique identifier for the element load, 1 identifier for the associated element and 4 values for the heat loads at the 4 element nodes, e.g.

*ElementLoads3*

*el0 e0 30 30 30 30*

* ***BoundaryConditions***

is followed by an arbitrary number of lines each defining a unique identifier for the boundary condition, 1 identifier for the associated node and 1 values for the predefined heat condition at the node, e.g.

*BoundaryConditions*

*bc0 n00 10*

**Input Data for instationary Heat Transfer Analysis**

the following **Keywords** are available for instationary Heat Transfer Analysis:

* ***Eigensolutions***

is followed by a line with an identifier for the Eigensolution and the number of eigensolutions to be considered, e.g.

*Eigensolutions*

*chmineyEigen 2*

* ***TimeIntegration***

is followed by a line with an identifier for the time integration, maximum time of analysis, duration of time steps and the integration parameter “alfa” for 1st order time step analysis, e.g.

*TimeIntegration*

*chimney 43200 120 1*

* ***Initial Temperatures***

followed by a line with a nodal identifier (incl. *all*) and temperatur value or „*stationary solution*“, e.g.

*Initial Temperatures*

*stationary solution*

*all 30*

* ***TimeDependent BoundaryConditions***

followed by an arbitrary number of lines with an identifer for the boundary condition and an identifier for the associated node and

* *file* for temperature distribution to be read from file or
* 1 value constant temperature or
* 3 values for harmonic excitation with amplitude, frequency and phase angle or
* a sequence of value pairs <time; temperature> for piecewise linear temperature distribution

*TimeDependent BoundaryConditions*

*tdbc0 n06 file   
or   
tdbc1 n06 0;0 15;20 20;30*

* ***TimeDependent NodeLoads***

followed by an arbitrary number of lines with an identifier for the node load, an identifier for the associated node and temperature distribution defined as above

* ***TimeDependent ElementLoads***

followed by an arbitrary number of lines with an identifier for the element load and temperature distribution as in “TimeDependent BoundaryConditions”.

**Input Data for Elasticity Analysis**

The implementation of elements for elasticity analysis is restricted to examples for 2D elements with 3 nodes and 3D elements with 8 nodes. The number of nodal degrees-of-freedom is 3.

* ***Element2D3***

followed by an arbitrary number of lines with an identifier for the element, 3 identifiers for the associated nodes, 1 identifier for the element cross section values and 1 for the element material, e.g.

*Element2D3*

*ELower00 n00 n10 n11 thick planeStress*

* ***Element3D8***

followed by an arbitrary number of lines with an identifier for the element, 8 identifiers for the associated nodes and 1 identifier for the element material, e.g.

*Element3D8*

*E000000 n000000 n000100 n010100 n010000 n000001 n000101 n010101 n010001 planeStress*

* ***3D8ElementMesh***

followed by an arbitrary number of lines with an identifier for the element, 3 identifiers for the associated nodes, 1 identifier for the element cross section values and 1 for the element material.

Generating elements is generally based upon generating a node mesh, e.g.

*Variable Node Mesh*

*0 1 3 7 15*

*K 0 0 0*

here: initial for element identifiers, initial for node identifiers, 4 elements in x,y,z resp. and material identifier

*3D8ElementMesh*

*E N 4 iso*

* ***CrossSection***

followed by an arbitrary number of lines with an identifier for the element cross section and 1 value for the element thickness

*CrossSection*

*thick 1*

* ***Material***

followed by an arbitrary number of lines with an identifier for the element material, 1 value for the modulus of elasticity (Young’s modulus) and 1 value for Poisson’s ratio

*Material*

*planeStress 3e7 0*

* ***NodeLoads***

followed by an arbitrary number of lines with an identifier for the element material, 1 value for the modulus of elasticity (Young’s modulus) and 1 value for Poisson’s ratio

*NodeLoads*

*P n62 0 -1e5*

* ***LineLoads***

followed by an arbitrary number of lines with an identifier for the line load, identifier for the start node of the line load, 2 load values at the start node, identifier of end node and 2 load values at the end node.

*LineLoads*

*L1 n62 0 -10000 n42 0 -10000*

* ***BoundaryCondition Nodes***

followed by an arbitrary number of lines with an identifier for the boundary condition, node identifier, definition of fixed nodal condition (e.g. “xy”) and optionally predefined deformations in x,y and z.

*BoundaryCondition Nodes*

*bc00 n00 xy*

Boundary Conditions in 3D space are usually defined for complete boundary faces and are generally also generated

* ***BoundaryCondition Faces***

followed by an arbitrary number of lines with an initial for identifiers for the boundary conditions, initial for face identifiers, initial for nodal identifiers, number of nodes generated and identifier for direction of fixed support, e.g.

*BoundaryCondition Faces*

*F X0 N 5 x*

*F Y0 N 5 y*

* ***BoundaryCondition Boussinesq***

followed by a single line with coordinates in each direction and an arbitrary number of lines with an initial for identifiers for the boundary conditions, face identifiers, initial for nodal identifiers and identifier for direction of fixed support, e.g.

*BoundaryCondition Boussinesq*

*0 1 3 7 15*

*B XMax N x*

*B YMax N y*

*B ZMax N z*

**Sample input data, executable Program, Source code and documentation**

Directory “FE-Analysis-App” contains the application “FE Analysis.application” with an associated “setup.exe” for installation under MS-Windows. Furthermore, there exists a subdirectory “input” with example data input for each application area. Finally, there is a subdirectory ” documentation containing this documentation and some more helpful documents.

Directory “FE Analysis Source code” contains the complete source code for this application including a “FE Library”.

**Examples for Structural Analysis input data**

Directory “input/StructuralAnalysis” contains

* 1Template (contains templates for the use of each keyword in Structural Analysis)

and some simple examples for analysis of a

* continuous beam
* truss
* building
* hinged frame
* frame
* frame with spring support
* chimney and a
* two bay frame

Subdirectory “input/StructuralAnalysis/Dynamics” contains some examples for dynamic analysis under earthquake excitation or piecewise-linear excitation in intervals

* classroom example for a simple system with 6 degrees-of-freedom
* truss
* chimney

Subdirectory “input/StructuralAnalysis/Dynamics/ExcitationFiles” contains file input for excitation acceleration input of a real earthquake (BM68elc.cc) and of piecewise linear intervals data.

**Examples for Heat Transfer Analysis input data**

Directory “input/HeatTransfer” contains

* 1Template (with templates for the use of each keyword in Heat Transfer Analysis)

and some simple examples for analysis of a

* classroom example
* wall corner
* chimney

Subdirectory “input/HeatTransfer/instationary” contains some examples for instationary heat transfer analysis under different forms of excitation or time-varying initial conditions

* classroom example for a simple system with 6 degrees-of-freedom
* truss
* chimney

Subdirectory “input/HeatTransfer/instationary/ExcitationFiles” contains file input for heat input of different heat input curves.

**Examples for Elasticity Analysis input data**

Directory “input/Elasticity” contains

* 1Template (with templates for the use of each keyword in Elasticity Analysis)

and some simple examples for analysis of a

* Cantilever
* elastic Halfspace