Multidiszciplináris Tudományok, 14. kötet, 4. szám (2024) pp. 116–124. **https://doi.org/10.35925/j.multi.2024.4.10**

THERMOMECHANICAL ANALYSIS OF FUNCTIONALLY GRADED COMPONENTS USING ABAQUS

Dávid Gönczi
senior lecturer, Institute of Applied Mechanics, University of Miskolc
3515 Miskolc, Miskolc-Egyetemváros, e-mail: david.gonczi@uni-miskolc.hu

Abstract
The main objective of this paper is to investigate the thermomechanical analysis of functionally graded structural components using Abaqus complete environment. We present the capabilities offered by the pre- and postprocessors of Abaqus finite element software for solving problems of bodies composed of heterogeneous materials, and how these can be utilized in custom numerical methods. We investigate, how we can utilize the built-in Python kernel of Abaqus complete environment and how we can use the results to train neural network or for solving optimization problems.

Keywords: *FGM, Abaqus, thermomechanics, optimization*

1. Introduction

As technology advances at an ever-increasing pace, the demand for new materials with special properties is also growing. In many fields, it is observed that engineers are researching the applications of these new, advanced materials. Pure metals are rarely used in engineering practice due to the limitations of their material behaviour. In many cases, solving a problem requires a material that is simultaneously hard, heat-resistant, or ductile. To address this issue, metals are combined with other metals or non-metal components to improve their material properties. One method for producing materials with enhanced properties is to combine them in a solid state, known as composite materials. These advanced, inhomogeneous materials are made from one or more solid-state substances with different mechanical and chemical properties. Composites offer excellent properties that differ from those of the individual components, and in most cases, they also have a lower mass. The use of these materials is often limited by a phenomenon called delamination. This process is particularly problematic in high-temperature environments when the base materials have different coefficients of linear thermal expansion.

To solve this problem, Japanese researchers developed the concept of Functionally Graded Materials in the mid-1980s during a hypersonic spacecraft project. Functionally Graded Materials (FGMs) are advanced materials whose composition and structure gradually change, resulting in corresponding changes in their properties. In these materials, the sharp interfaces between the constituent materials are eliminated. Instead of a sharp interface, where failure might initiate, a graded (gradual) interface is formed, providing a smooth transition from one material to another.

Numerous studies have dealt with the mechanics of functionally graded materials (FGMs) from various aspects. A variety of papers have proposed analytical, semi-analytical, and numerical methods to solve thermomechanical problems in spheres, cylinders, beams, and disks (e.g. (Pen and Li, 2009; Gönczi, 2017; Kim and Noda, 2002; Nayak et al., 2020; Kiss, 2020; Afsal et al., 2023; Khatir et al., 2021)). These methods are restricted to special geometries, loading or specific material composition.

Several books provide solutions to linear elastic problems in non-homogeneous bodies, such as those found in (Hetnarski and Eslami, 2010; Shen, 2009; Noda et al., 2000). The description of physical phenomena that depend on time and space usually leads to systems of partial differential equations, for which exact, closed-form solutions cannot be determined in most cases. With the advancement of computing, numerical methods have come to the forefront in solving engineering problems, with one of the most widespread methods today being the Finite Element Method (FEM), which is a numerical technique where the desired fields are approximated over a finite number of subdomains (elements) using approximation functions. By joining these subdomains, the parameters describing these fields are then determined through an algebraic equation or inequality system derived from some error principle or variational principle. Finite element analysis (FEA) greatly simplifies complex product design processes. When properly integrated into the design workflow, it accelerates problem-solving and optimization tasks (Gönczi, 2021) while reducing costs. Additionally, thanks to standardized data formats and integrated modules, FEA can be easily incorporated into the design process, enhancing efficiency and streamlining development.

2. The preprocessor of Abaqus CAE

Abaqus CAE (Complete Abaqus Environment) is one of the leading general-purpose finite element software tools. It is widely used in the industry due to its ability to analyze a broad range of engineering problems. Over the years, several modules have been developed, with the most important being (Abaqus 6.13):

- Abaqus/Standard and Abaqus/Explicit: A solver based on implicit integration schemes and a module for handling highly nonlinear dynamic and thermomechanical problems.
- Abaqus CFD: A fluid dynamics solver for solving engineering tasks.
- Abaqus ElectroMagnetic: A module for solving electromagnetic problems.

The main steps of using the software are illustrated in Figure 1.

Figure 1. *Modelling in Abaqus CAE.*

In the early versions of Abaqus, only the processor module was implemented, which was written in Fortran. In 1999, Abaqus CAE was introduced, expanding the software with pre- and post-processing modules. These additions allowed users to work in a graphical interface, providing flexibility in model creation. The pre- and post-processing modules were developed using the open-source Python language.

In summary, Abaqus CAE consists of components written in different programming languages, the processor is responsible for the actual numerical simulation, written in a low-level language (Fortran).

The pre- and postprocessors are designed for user convenience and efficiency, these components are written in a higher-level language (Python).

The processor is the essential part of the software, developed first and is the first to receive new procedures. Often, new features are only later incorporated into the preprocessor, meaning it initially has fewer functionalities compared to the processor. The communication between physical modules often occurs through file exchanges. Figure 3 illustrates the interaction between modules and their associated key files.

The main files generated and used during simulation include:

- .inp: The core input file for defining the modelling task; without this, no simulation can run, as it's the processor's main input.
- .f or .for: Subroutines required for simulations, marked in the input file as optional inputs. Certain modelling options are only accessible through these.
- .cae: Pre-processor file type, essential for setting up tasks in the pre-processor and version-dependent. It offers more flexibility in areas like geometry compared to the input file.
- .jnl: A Python-based log file generated by the preprocessor, recording user activity step by step.
- .odb: Output file containing results, which is used by the postprocessor.

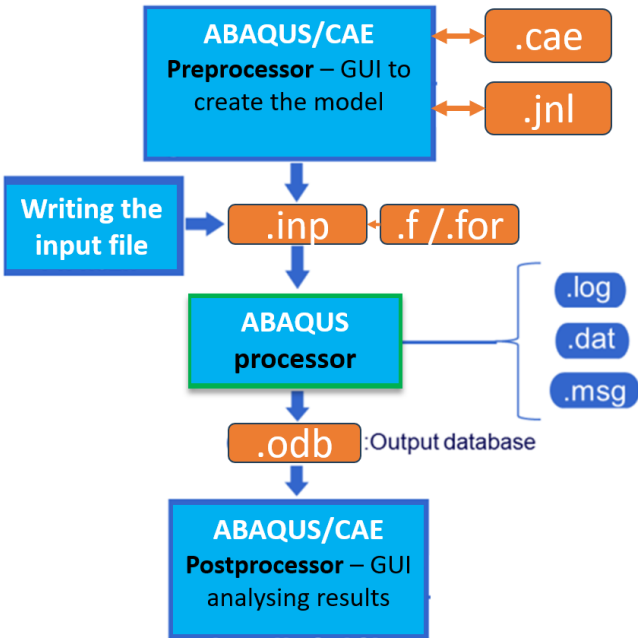

Figure 2. *The architecture of Abaqus CAE.*

Some of the files generated during simulations store information about the simulation's progress. The input file is human-readable and easily editable. In the early versions of the software, these files were manually created. Even in newer versions, the input file remains the primary driver of the simulation, with the preprocessor generating it at the end of model setup and launching the simulation. Since the input file offers more options than the preprocessor, manual editing is often unavoidable. Additionally, subroutine files written in Fortran can be attached, allowing access to a wide range of functions, such as user-defined material models (UMAT) that can be utilized for functionally graded materials or specifying initial stress states. For these features to work, appropriate compilers must be installed.

To automate model creation, simulation execution, and data recording, we can utilize the powerful features provided by the preprocessor. Python scripts can control the simulation's execution and make decisions based on the results stored in the output file. Here we can create AI agents to control the simulations.

Since the software's components communicate via files, they can also be controlled, written, and read by custom code. The resulting data can be used for optimization tasks or even for training neural networks.

There are several ways to program the Abaqus finite element software. If we want to modify the equations or methods, we will need to write subroutines. We can also write code to generate input files, especially if you want to add new functionality to the CAE preprocessor. Additionally, you can create scripts to flexibly set up and control simulations, using Python-based code for these tasks.

The Abaqus CAE system extends the Python package with more than 500 classes (or objects) and numerous methods that operate between them. These are grouped into three main categories: session, mdb, and odb, as illustrated in the left part of Figure 3, which highlights its object-oriented approach. Containers, in this context, refer to collections of similar objects.

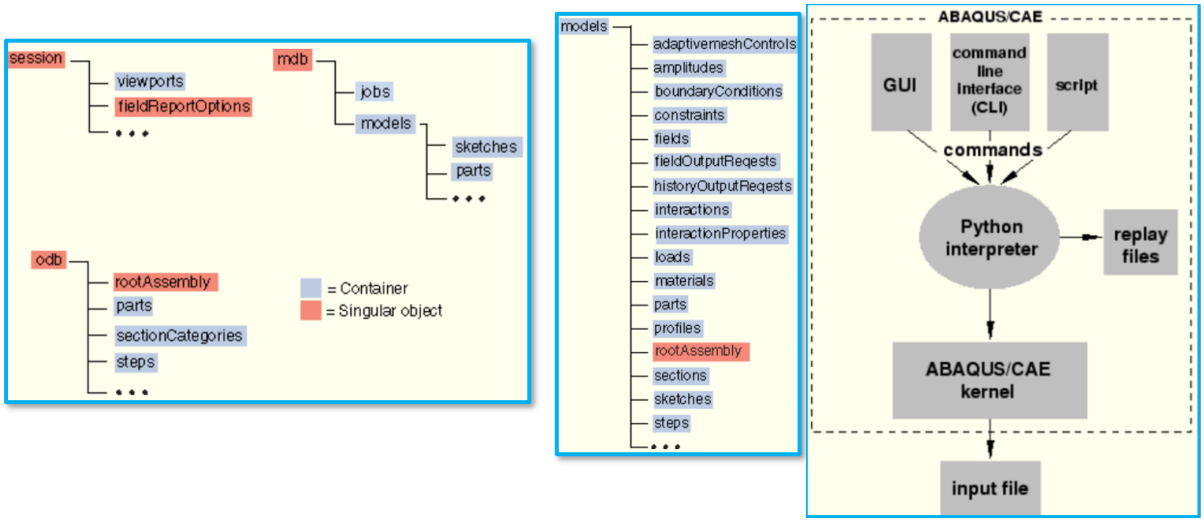

Figure 3. *The hierarchy of classes and objects of Abaqus and the structure of the preprocessor (Abaqus 6.13).*

These classes, containers, and objects can be effectively used for scripting. This way we can write scripts that create models with given parameters. This allows us to write AI agents to run simulations with given set of parameters based on previous results and other programmed logic. Objects can be created using constructors. An example of this is mdb.models['Model-1'].Part(dimensionality= TWO_D_PLANAR, name='curved_beam', type=DEFORMABLE_BODY). Objects have attributes, and their values can be modified using the setValues() setter method. Some objects do not have constructors and are created as part of other objects. Python scripts can be run from both the GUI and the command line (without the GUI). Additionally, backward compatibility is maintained in this case, unlike when using the original CAE files. The built-in Python kernel of Abaqus CAE is illustrated in the right part of Figure 3.

3. Numerical example

Let us consider a curved beam in a cylindrical coordinate system ($Or\varphi z$). The sketch of the problem can be seen in Fig. 4. At first let us present the analytical solution of the problem. The displacement vector

u of the Euler-Bernoulli curved beam in the case of in-plane deformation can be represented as (Ecsedi and Dluhi, 2006), (Ecsedi and Gönczi, 2022)

$$\mathbf{u} = u\mathbf{e}_r + v\mathbf{e}_\varphi + w\mathbf{e}_z \ , u = U(\varphi),\ v = r\phi(\varphi) + \frac{dU}{d\varphi},\ w = 0, \tag{1}$$

where $\phi(\varphi)$ is the rotation of the cross section and the beams deforms in its principal ($r\varphi$) plane. For the stress resultant - displacement relations we introduce the following cross-sectional properties:

$$AE_0 = \int_A EdA\,,\ r_c = \frac{\int_A rEdA}{EA_0}, R = \frac{AE_0}{\int_A r^{-1}EdA},\ e = r_c - R. \tag{2}$$

where A is the area of the cross section, E is the modulus of elasticity. The thermal loading is characterized by two quantities (using α, t linear coefficient of thermal expansion and temperature field)

$$\int_A E\alpha t dA = E_0 A h_n\,,\ \int_A rE\alpha t dA = E_0 A h_m. \tag{3}$$

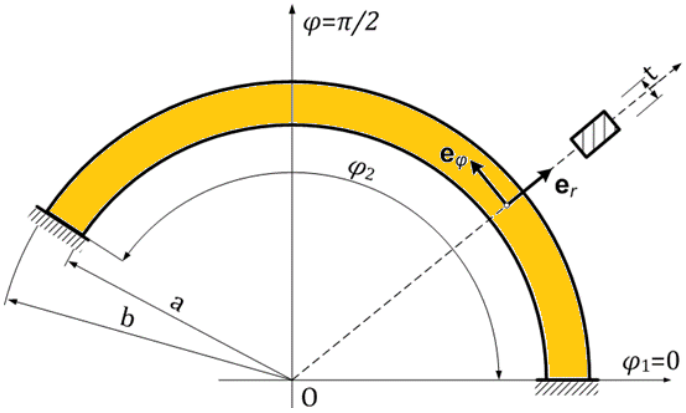

Figure 4. *The sketch of the curved beam.*

The stress resultants are defined according to paper (Ecsedi and Dluhi, 2006) as

$$N(\varphi) = \int_A \sigma_\varphi dA\,,\ M(\varphi) = \int_A r\sigma_\varphi dA\,,\ S(\varphi) = \int_A \tau_{r\varphi} dA, \tag{4}$$

where $\tau_{r\varphi}$ is the shear stress and σ_φ is the tangential normal stress. Using the equilibrium equations of the beam and adding the previously presented equations and notations the following system of equations can be derived:

$$W(\varphi) = \frac{N_0 R r_c}{AE_0 e} cos\,\varphi - \frac{S_0 R r_c}{AE_0 e} sin\,\varphi + \frac{M_0 R}{AE_0 e} + \frac{R r_c h_n}{e} - \frac{R h_m}{e}, \tag{5}$$

$$\frac{d\phi}{d\varphi} = -\frac{N_0 R}{AE_0 e} cos\,\varphi + \frac{S_0 R}{AE_0 e} sin\,\varphi + \frac{M_0}{AE_0 e} - \frac{R h_n}{e} + \frac{h_m}{e}, \tag{6}$$

$$\frac{d^2U(\varphi)}{d\varphi^2} + U(\varphi) = \frac{N_0 R r_c}{AE_0 e} cos\,\varphi + \frac{S_0 R r_c}{AE_0 e} sin\,\varphi + \frac{M_0 R}{AE_0 e} + \frac{R}{e}(r_c h_n - h_m). \tag{7}$$

In the previous equations N_0, S_0, M_0 were reaction force components. From these equations the displacement components can be calculated:

$$\phi(\varphi) = -\frac{N_0 R}{AE_0 e} sin\,\varphi + \frac{S_0 R}{AE_0 e}(1 - cos\,\varphi) + \frac{M_0}{AE_0 e}\varphi + \frac{h_m - Rh_n}{e}\varphi, \phi(\varphi_1) = 0. \quad (8)$$

$$U(\varphi) = C_1\,cos\,\varphi + C_2\,sin\,\varphi + \frac{R}{e}(r_c h_n - h_m) - \frac{M_0 R}{AE_0 e} + \frac{N_0 R r_c}{2AE_0 e}\varphi\,sin\,\varphi + \frac{S_0 R r_c}{2AE_0 e}\varphi\,cos\,\varphi. \quad (9)$$

$$V(\varphi) = C_2\,cos\,\varphi - C_1\,sin\,\varphi + \frac{N_0 R r_c}{2AE_0 e}(sin\,\varphi + \varphi\,cos\,\varphi) + \frac{S_0 R r_c}{2AE_0 e}(cos\,\varphi - sin\,\varphi). \quad (10)$$

The unknown constants C_1, C_2 and the unknown reactions N_0, S_0, M_0 are obtained from the boundary conditions.

After that we can create the model in Abaqus. We used scripts and parameters to automate the simulation. Due to the complexity of the material (FGM), we needed a material class to compute the values of the material properties. The simplest approach is to define layers in the functionally graded body, in which the material parameters are constants. In our case every element row has its own calculated constant set of values provided by the material class. We used the built-in classes of Abaqus CAE to create the model using parameters, which include the geometry, boundary conditions, meshing and loading: e.g. *Class Runsimulation*(*Material: material*, *internal_radius*, *external_radius*, *fi*, *thickness*, *temperature_field*, *number_of_layers*, *boundary_1*, *boundary_2*). When the simulation is ready, we can use the classes and objects of the postprocessor in the odb class to get the results. These steps can speed up the calculations because we can use the processor without starting the GUI of the preprocessor or postprocessor (especially when a lot of shorter calculations are needed). For the calculations the following data were used:

$$a = 0.085\text{m}, b = 0.09\text{m}, h = 0.2\text{mm}, E_0 = 205\text{GPa}, T_1 = 100°\text{C}, T_2 = 80°\text{C},$$

$$t = T_2 + \frac{T_1 - T_2}{b - a}(r - a), \alpha_0 = 3.9 \cdot 10^{-6}°\text{C}^{-1}, \alpha(r) = \alpha_0 \left(\frac{r}{a}\right)^2, E(r) = E_0 \left(\frac{r}{a}\right)^2, \varphi_2 = 135°.$$

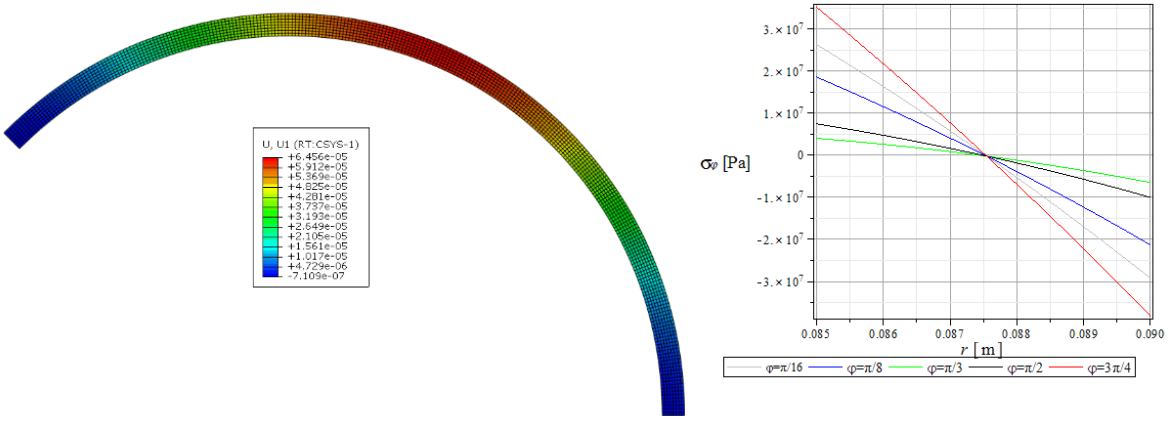

Figure 5. *The mesh and the results of the curved beam.*

The results were in good agreement, the maximum relative error for the dominant hoop or tangential stresses is less, than 2%. We could easily refine the mesh with control parameters to execute the mesh convergence test. The mesh presented in Fig. 5 was a balanced mesh which produced good accuracy. Finer meshes lead to a very small accuracy increase for the stress distribution (less than 2%). If we want to investigate the effect of the material distribution, we can do it easily by modifying the arguments of the material class.

4. Discussion about the applications of scripts

The next question is how to use the previously created Python script to enhance the simulation of functionally graded structural components. When a series of calculations is required, we can program an AI agent, which controls the parameter inputs for the simulations. It ensures that the behaviour of the system remains within reasonable range, which means for example – if certain parameters reach a threshold that results in excessively high stresses, the agent resets the incrementation of that parameter. It also plays an important role in handling errors during simulation and restarting the process if the system shuts down. This approach allows for the generation of large datasets to train neural networks, which means that we can use the previously presented method to calculate stresses and displacements as an input for the network.

Neural networks can significantly reduce lengthy computational tasks. For larger models, especially with complex geometries, it's beneficial to employ modelling sub-techniques to accelerate calculations, as this can lead to a significant increase in speed when producing large amounts of data.

In many cases, we can also use optimization procedures for the design of engineering structures. One example is evolutionary algorithms, which allow us to search for globally good solutions based on various criteria (although we may not necessarily find the absolute best solution). In this context, we have a variable array that stores the parameters to be optimized, along with settings such as the number of generations, population size, elitism (how many of the best individuals are directly carried over to the next population), and tournament size (how many individuals compete against each other). The main components of genetic algorithms include:

- Data conversion to chromosomes: Often, variables are converted into a binary sequence. This involves designating discrete values within the examined range based on the chromosome length for our investigations.
- Conversion of chromosomes to data sets: The variable array is retrieved from the chromosome.
- Creation of the initial population: A set of chromosomes representing the initial population is generated randomly.
- Running simulations and determining target values: Selected variable combinations are compared based on results.
- Tournament selection: Using a tournament function, selected individuals compete against each other (e.g., randomly).
- Chromosome mutation: Random changes are introduced to selected chromosomes using a specified function.
- Creating a new population: The population is sorted based on simulation results. Some individuals are carried over through elitism. The remaining individuals compete, and from the two tournament winners (and their chromosomes), new offspring are generated, which may mutate with a certain probability or inherit directly from the parents' chromosomes.

- The control function of the genetic algorithm: It runs calculations across populations, creates new ones based on results, and manages the simulation process.

For optimization tasks, the system can be analyzed from multiple perspectives. In the case of functionally graded materials, one of the most straightforward approaches is to examine the material composition. Additionally, we can analyze the geometry or even optimize based on cost. There are a lot of algorithms for optimization besides genetic algorithms. The gradient based method uses information about the gradient of the objective function to iteratively improve the solution. They are highly efficient for problems where the objective function is smooth and differentiable, but it can get trapped in local minima for non-convex problems. Another method is the simulated annealing, which is inspired by the annealing process in metallurgy, this method allows random exploration of the design space with occasional acceptance of worse solutions to avoid local minima. As the algorithm progresses, the probability of accepting worse solutions decreases. The particle swarm optimization is a population-based optimization method inspired by the social behaviour of birds and fish. Each "particle" adjusts its position in the search space based on its own experience and the experience of neighbouring particles. It can be applied in various structural optimization problems, including material selection, shape optimization, and topology optimization. It is simple to implement, suitable for large-scale and highly non-convex problems, but it can converge prematurely or slowly if not well-tuned. Differential evolution is a population-based optimization technique that relies on the differences between randomly selected individuals to drive the search. It mutates and recombines solutions and selects the best ones for the next iteration and is suitable for optimizing complex, non-differentiable, and non-linear functions. It can be efficient for global optimization and robust against local extremum but can be computationally expensive. Level set methods represent the design boundary implicitly as a level set of a higher-dimensional function. The optimization process evolves the boundary to minimize the objective function. These are useful in shape and topology optimization. These handle complex boundary evolutions efficiently and are well-suited for large-scale structural optimization. They are computationally expensive and require fine-tuning. In surrogate-based optimization, a surrogate model (e.g., kriging, radial basis functions, neural networks) approximates the expensive-to-evaluate objective function. The surrogate model is iteratively refined during the optimization process. It is preferable to use when evaluating the objective function is computationally expensive (e.g., FE analysis of complex problems). It reduces computational cost by limiting the number of expensive function evaluations, but it requires careful modelling.

5. Summary

A method was presented to run simulations using Python scripts in Abaqus CAE. We investigated the options to use the built-in Python kernel of the complete Abaqus environment. We solved a curved beam problem using an exact solution based on the Euler-Bernoulli beam theory. Then we checked our finite element script using the previously presented analytical solution. The results were in good agreement and we could easily modify the model or further improve the accuracy of the simulation by adjusting the parameters of the script. Then we investigated the methods that can use these scripts. We considered the utilization of neural networks or the application of different optimization algorithms.

References

[1] Pen, X., & Li, X. (2009). Thermoelastic analysis of functionally graded annulus with arbitrary gradient. *Applied Mathematics and Mechanics (English Edition), 30*(10), 1211–1220. **https://doi.org/10.1007/s10483-009-1001-7**

[2] Gönczi, D. (2017). Thermoelastic analysis of thick-walled functionally graded spherical pressure vessels with temperature-dependent material properties. *Journal of Computational and Applied Mechanics, 12*(2), 109–125. **https://doi.org/10.32973/jcam.2017.008**

[3] Kim, K. S., & Noda, N. (2002). Green's function approach to unsteady thermal stresses in an infinite hollow cylinder of functionally graded material. *Acta. Mech., 156*, 61–145. **https://doi.org/10.1007/BF01176753**

[4] Nayak, P., Bhowmick, P., & Saha, K. N. (2020). Elasto-plastic analysis of thermo-mechanically loaded functionally graded disks by an iterative variational method. *Engineering Science and Technology, an International Journal, 23*(1), 42–64. **https://doi.org/10.1016/j.jestch.2019.04.007**

[5] Kiss, L. P. (2020). Nonlinear stability analysis of FGM shallow arches under an arbitrary concentrated radial force. *International Journal of Mechanics and Materials in Design, 16*(1), 91–108. **https://doi.org/10.1007/s10999-019-09460-2**

[6] Afsal, K. P., Swaminathan, K., Indu N., & Sachin, H. (2023). A novel EFG meshless-ANN approach for static analysis of FGM plates based on the higher-order theory. *Mechanics of Advanced Materials and Structures, 31*(25), 6501–6517. **https://doi.org/10.1080/15376494.2023.2231459**

[7] Khatir, S., Tiachacht, S., Le Thanh, C., Ghandourah, E., Mirjalili, S., & Wahab, M. A. (2021). An improved Artificial Neural Network using Arithmetic Optimization Algorithm for damage assessment in FGM composite plates. *Composite Structures, 273*, 114287. **https://doi.org/10.1016/j.compstruct.2021.114287**

[8] Hetnarski, R. B., & Eslami, M. R. (2010). *Thermal Stresses – Advanced Theory and Applications.* Springer, New York, USA. **https://doi.org/10.1007/978-3-030-10436-**8

[9] Shen, H.-S. (2009). *Functionally Graded Materials: Nonlinear Analysis of Plates and Shells.* CRC Press, London, UK. **https://doi.org/10.1201/9781420092578**

[10] Noda, N., Hetnarski, R. B., & Tanigawa, Y. (2000). *Thermal Stresses.* Lastran Corporation, Rochester, New York, USA. **https://doi.org/10.1115/1.1349549**

[11] Gönczi, D. (2021). Topológiai optimalizálási feladatok alapvető sajátosságai Abaqus végeselemes programrendszerben. *Multidiszciplináris Tudományok, 11*(4), 177–187. **https://doi.org/10.35925/j.multi.2021.4.22** (in Hungarian)

[12] Abaqus 6.13 online documentation. Dassault Systems. 2015.

[13] Ecsedi, I., & Gönczi, D. (2022). Thermal stresses in radially non–homogeneous curved beams. *Annals of the Faculty of Engineering Hunedoara, 20*(4), 107–114.

[14] Ecsedi, I., & Dluli, K. (2006). A linear model for the static and dynamic analysis of nonhomogeneous curved beams. *Applied Mathematical Modelling, 29*(1-2), 1211–123. **https://doi.org/10.1016/j.apm.2005.03.006**