Example of essential boundary conditions

Performing an FEM simulation is like teamwork, where team players are factors such as geometry, material properties, load, boundary conditions, network, solver in a broader sense. Effective contributions from all team members make the team very successful (valid and desired simulation result). Of course, the user is the captain of the team, with whose prudence and engineering skills the team succeeds. Okay, now that you know you're an important decision maker as captain, it's good to know in depth about your team player, which will make it easier for you to work when used with the right technical sense. So, let's welcome that the player – boundary conditions Yes, in this article, you will read about the boundary conditions in the final element of analysis (FEA), their types and applications in real practice. Why do we need them? Finite element analysis involves solving partial differential equations applied over analytical domains for variables (offset, stress, etc.) that are interesting. 7 general steps in any FEM simulation These control equations are applied to the finite elements discretized above the domain and are difficult to solve for the problem as such. Consider the example of a static, linear problem of FE analysis – Bending of the cantilever beam. If we approximate this beam with a linear element and linearize the control equations, we will have elements for the well-known formulation [k] {u} = {F}, where {F}, {u} are force and displacement vectors, and [k] is the strength matrix. Based on the approximation of one element for easier illustration, the above equation will look as if q and θ represent the offsets and rotational shifts of the two nodes of the element that we assumed. In a cantilever beam, one node (corresponding to the fixed end) is fully constrained, which means no displacement and no rotation. This is indeed a boundary condition. As we clearly know, q1 and θ1 are zero, we can exclude the corresponding rows and columns from the above formulation of the array vector. Ultimately, the 4X4 matrix is reduced to a 2X2 matrix, resulting in fewer equations to be solved without compromising accuracy. Sounds like a great idea, doesn't it? Cool. Now that you've just noticed the importance of a boundary state in a 2-node element, imagine how powerful and numerically efficient the boundary application would be for problems with thousands of nodes or degrees of freedom. With the correct use of the engineering sense, the correct set of boundary conditions can be obtained. With experience, I have practically witnessed the importance of border conditions in computationally larger problems. If you're a beginner, then it's actually good practice to spend some understanding of the problem in your hand before actually jumping into CAE you intend to use. This ensures that you take advantage of boundary conditions and solve your problem smartly with smaller resources without losing accuracy. Going back to the borders, there are mainly two categories of border conditions, namely basic and natural border conditions. The following table is aimed at the difference between them. Basic boundary conditions Natural boundary conditions Determining the value of a variable at the boundary Specify the value of the derivation of the variable at the boundary They are explicitly set for resolution These conditions are automatically, naturally met without external conditions They are met in the exact sense They are met up to the order of polynomial degree Some examples would be displacement in stress analysis Bending moment or she she sheath forces in stress analysis Temperature in thermal analysis Adiabatic boundary in heat conduction analysis After going through this table you can now find out the type of boundary state that we used for our console beam illustration? And if you read or looked at standard FEM textbooks or manuals, you would come across conditions such as Dirichlet boundary conditions and Neumann border conditions. As a beginner, it is safe to keep this thumb rule in mind that in most cases, Dirichlet boundary conditions belong to the Basic and Neumann boundary conditions of the Natural Category. This claim is not always true, but it applies to most simple examples. If you have gone through one of my previous articles (Strong and weak wording in fea), during the weak formulation of the problem, natural boundary conditions (Neumann) are applied automatically. These boundary conditions are applied before the development of algebraic equations for unknown coefficients. But the scenario is different when it comes to basic boundary conditions. The shape functions we choose do not automatically meet these conditions. Therefore, basic boundary conditions (Dirichlet) should be used before the global solution of FE equations. The topic of basic and natural boundary conditions is difficult for a beginner to understand when reading the standard FEM textbook directly. Simple! ~ Renga Book on Python Scripting for ABAQUS: I wrote a book to help you write Python scripts for ABAQUS in just 10 days. Why would you buy my book? Please use the link below to purchase the book. Crash Course on Python Scripting for ABAQUS: Learn how to write python scripts for ABAQUS in 10 days Hermes distinguishes between basic and natural boundary conditions. The first type eliminates degrees of freedom from the domain boundary (the solution is while the other does not. Examples of basic boundary conditions are Dirichlet conditions for H1 problems and perfect conductor conditions in space H(curl). Examples of natural boundary conditions are Neumann or Newton (Robin) conditions for H1-problems and impedance conditions in space H(curl). Only basic conditions are explicitly treated in Hermes, while natural ones are defined using surface integrals in a weak formulation. Let's start by showing the default ways to define basic boundary conditions. We have already met the default base boundary condition in the previous example 03-poisson: // Initialize the base boundary conditions. DefaultEssentialBCConst<double> bc_essential(Hermes::vector<std::string>(Bottom, Inner, Outer, Left), FIXED_BDY_TEMP); He assigned a constant FIXED_BDY_TEMP to all borderline edges with the Bottom, Inner, Outer, or Left tags. After you create one or more basic boundary conditions, they are passed to the EssentialBC container class: EssentialBCs<double> bcs(&bc_essential); The purpose of this container is to collect all the basic boundary conditions to be passed into space, as we will see shortly. The EssentialBCs class constructor can accept Hermes::vector of the basic boundary conditions. The next default base boundary condition reads boundary values from a given function. This is useful, for example, in benchmarks with a known exact solution. Typical use is as follows: // Set the exact solution. CustomExactSolution accurate (&mesh, EXACT_SOL_P); Initialize boundary conditions. DefaultEssentialBCNonConst<double> bc_essential (Bdy, & EssentialBC<double> bcs(&bc_essential); This technique is used in the P01/07-general tutorial example and in several benchmarks that are part of hermes storage examples. Custom base conditions can be created by subclassing the abstract EssentialBoundaryCondition class. This class can represent any basic boundary conditions that depend on space and time. Each child of this class must redefine pure virtual functions get_value_type() and value(). This will be explained in more detail in the following example 04-bc-dirichlet. $\begingroup$ What is the difference between basic boundary conditions and natural boundary conditions? $\endgroup$ OSTI.GOV Technical Report: Basic boundary conditions with equal C1 endpoints in curved domains The implementation of basic boundary conditions in the analysis of C1 finite elements requires proper processing of both boundary conditions on the second-order solution differential and the curvature of the domain boundary. The method for storing basic boundary conditions using straight features (where features are not deformed to approach a curved domain) is </double> </double> </double> </std:string> </double>It is proven that multiplying the array equation by the local rotation matrix at each boundary node is not an optimal transformation. There is a uniquely optimal transformation that does not take the form of a similarity transformation due to the non-orthogonality of the transformation to curved coordinates. Authors: N.M Ferraro, S.C. Jardin, X. Luo Publication Date: Fri Feb 19 00:00:00 EST 2010 Research Org.: Princeton Plasma Physics Lab. (PPPL), Princeton, NJ (United States) Sponsorship Org.: USDOE Office of Science (SC) OSTI Identifier: 973199 Message Number: PPPL-4497TRN: US1001906 DOE Contract Number: DE-ACO2-09CH11466 Source Type: Technical Report Country Publication: United States Language: English Subject: 70 PLASMA PHYSICS AND FUSION TECHNOLOGY; LIMIT CONDITIONS; IMPLEMENTATION; ROTATION; TRANSFORMATION; The method of the final element. Differential differential equations N.M. Ferraro, S.C. Jardin, X. Luo. Basic boundary conditions with equal c1 finite elements in curved domains. United States: N. p., 2010. Web. doi:10.2172/973199. N.M. Ferraro, S.C. Jardin, X. Luo. Basic boundary conditions with equal c1 finite elements in curved domains. United States. doi:10.2172/973199. N.M. Ferraro, S.C. Jardin, X. Luo. Fri. Basic boundary conditions with equal c1 finite elements in curved domains. United States. doi:10.2172/973199. . @article{osti_973199, title = {Essential Boundary Conditions with Straight C1 Finite Elements in Curved Domains}, author = {N.M. Ferraro, S.C. Jardin, X. Luo}, abstractNote = {Implementation of basic boundary conditions in the analysis of C1 finite elements requires proper processing of both boundary conditions on the second-order solution differential and the curvature of the domain boundary. The method for storing basic boundary conditions using straight features (where elements are not deformed to approach a curved domain) is described. It is proven that multiplying the array equation by the local rotation matrix at each boundary node is not an optimal transformation. A uniquely optimal transformation is found that does not take the form of a similarity transformation because of the non-orthogonality of the transformation to curved coordinates.}, doi = {10.2172/973199}, journal = {},number = volume = , place = {United States}, year = {Fri 19 00:00:00 EST 2010}, month = {Fri Feb 19 00:00:00 EST 2010} } Similar entries in OSTI.GOV collections: