1D Model Problem: Verification and Validation:

Over the past two lectures, we have discussed computer implementation of a Bubnov-Galerkin finite element approximation of our 1D model problem. Once one has a finite element code, however, one must ensure it acts as intended. This is the purpose of validation and verification. According to the American Society of Mechanical Engineers (ASME):

Verification: The process of determining that a computational model accurately represents the underlying mathematical model and its solution.

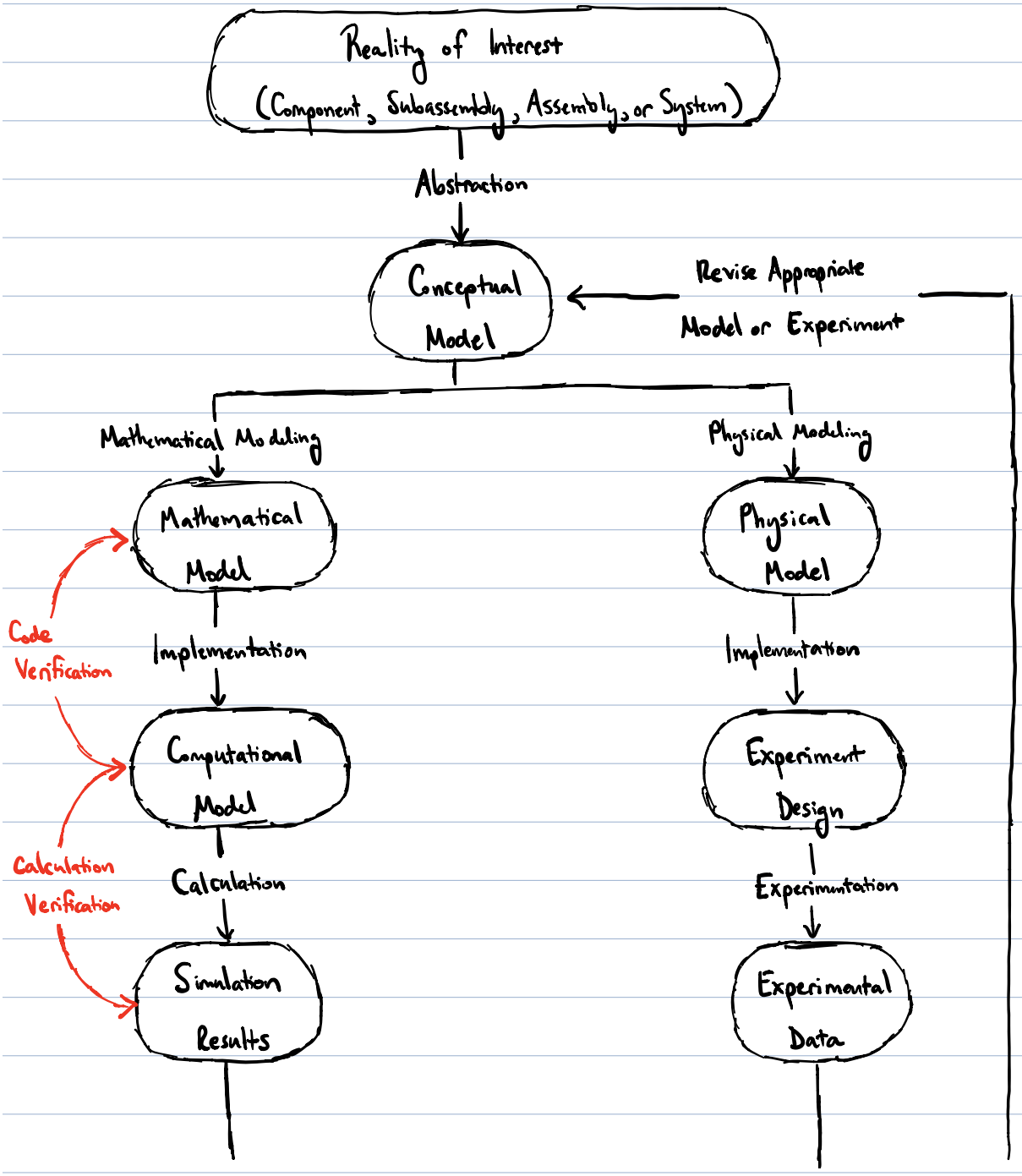
"Are the equations being solved correctly?"

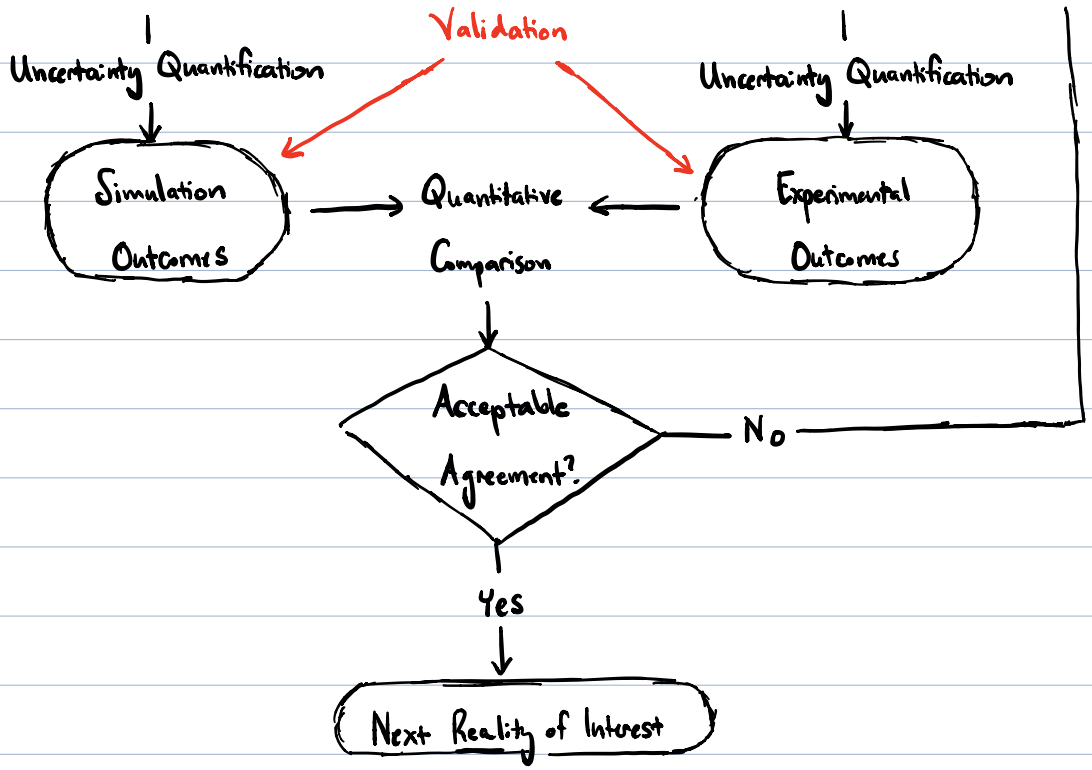
Validation: The process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model.

"Are the right equations being solved?"

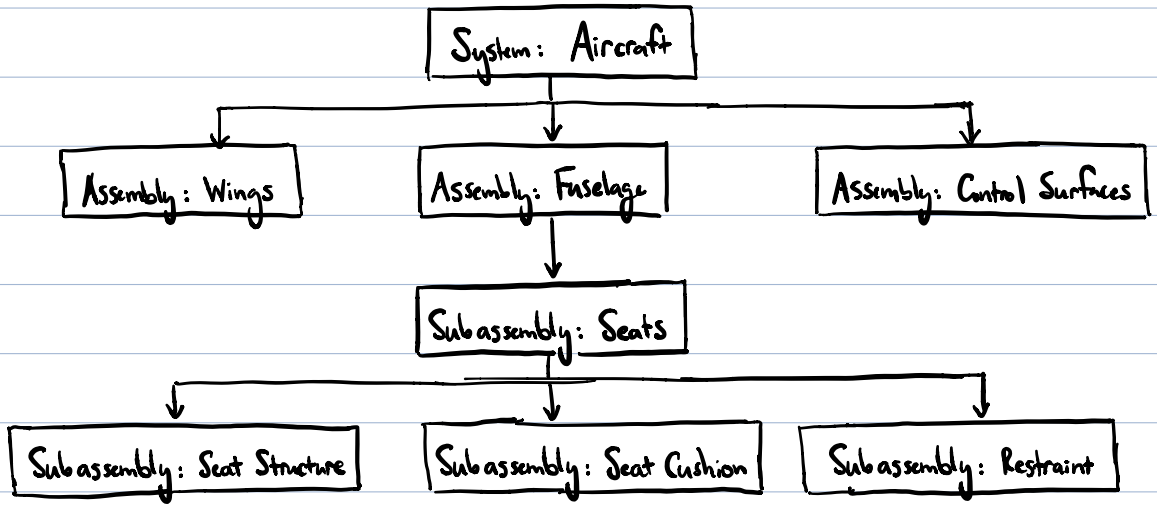
The ASME further subdivides verification into two major components: code verification - seeking to remove programming and logic errors in the computer program - and calculation verification - seeking to estimate the numerical errors due to discretization.

Verification and validation are perhaps best explained with a visual aid illustrating the verification and validation process, taken gratuitously from ASME 10-2006: Guide for Verification and Validation in Computational Solid Mechanics:





In the above flow chart, the reality of interest represents the physical system for which data is being obtained. In the context of solid mechanics, it could be a system, assembly, subassembly, or component. For instance, for an aircraft:



Typically, the verification and validation process is first executed at the component level (or even the subcomponent level), then at the subassembly level, next at the assembly level, and finally at the system level.

The conceptual model is the collection of assumptions and descriptions of physical processes representing the behavior of the reality of interest from which the mathematical model and validation experiments can be constructed. For instance, when considering the structural deformation of a component, the conceptual model consists of the geometry and material layout of the component as well as loads experienced by the component.

The mathematical model consists of the mathematical equations, initial and boundary conditions, and modeling data needed to describe the conceptual model. In the context of the 1D model problem considered here, the mathematical model is the strong, weak, minimization, or variational form of the problem.

The computational model consists of the numerical approximation of the mathematical model, usually in the form of numerical discretization, solution algorithm, convergence criteria, as well as computer implementation. In the context of the 1D model problem considered here, the computational model is a computer implementation of a finite element approximation of the 1D model problem. Thus it involves physical parameters, discretization parameters, and even quadrature data and the linear solver employed.

The simulation results consist of the predicted fields from the computational model as well as derived quantities of interest. For the 1D model problem considered here, the predicted field is the Galerkin finite element solution, provided, of course, the finite element code is free of bugs. The simulation outcomes account for uncertainties - potential deficiencies due to a lack of knowledge - present in the computational model. These uncertainties are due to the fact that physical parameters must be specified in a computational model, and we often do not know precisely what these physical parameters should be. Note that errors in a computational model are not uncertainties. Rather, errors are recognizable deficiencies due to numerical discretization, solver convergence criteria, or code implementation mistakes.

The physical model, experiment design, experimental data, and experimental outcomes are the experimental analogues of the mathematical model, computational model, simulation results, and simulation outcomes, respectively. Given the focus of this class, we will not discuss these in more depth.

With the above concepts in hand, we can now be more precise as to what we mean by verification and validation. As mentioned previously, verification is the process of determining that a computational model accurately represents the underlying mathematical model and its solution. More precisely, verification is the process of determining the simulation results are mathematically correct. In the context of our 1D model problem, verification involves determining whether or not the discrete solution predicted

by the computational model is an accurate prediction of the solution of the variational form. Code verification, then, is the process of determining whether or not the Galerkin solution predicted by the computational model is actually the Galerkin solution, up to solver convergence criteria, quadrature, and finite precision arithmetic (that is, the computational model is free of code implementation mistakes). Calculation verification, on the other hand, is the process of determining whether or not the Galerkin solution has converged to the solution of the variational form. Note then that code verification should be executed before calculation verification. Typically, code verification is carried out by developers of a finite element code, while calculation verification is carried out by users of a finite element code.

Recall validation is alternatively defined as the process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model. Validation is typically carried out by comparing simulation outcomes with experimental outcomes. If there is a bad match, then we are comparing apples and oranges, and either the mathematical model, computational model, physical model, or experiment design must be changed accordingly.

Validation is a very important element of simulation-based engineering, but it is outside the scope of this class. Verification, on the other hand, is vital to the success of any finite element analysis. Thus,

We discuss here a number of code and calculation verification strategies with a specific focus on finite element analysis.

Finite Element Code Verification:

To verify that a finite element code is working as intended, one can conduct a number of verification tests:

Test 1. One can check that the discrete solution predicted by the code is equal to a hand-derived Galerkin finite element solution for the same mesh and polynomial degree.

Test 2. One can check that the discrete solution predicted by the code is equal to a solution predicted by a verified finite element code for the same mesh and polynomial degree.

Test 3. One can check that a quantity of interest predicted by the code converges to a benchmark value under mesh refinement.

Test 4. One can check that the discrete solution predicted by the code converges to known analytical solutions under mesh refinement.

Test 5. One can check that the discrete solution predicted by the code exhibits the expected rate of convergence under mesh refinement.

Test 6. One can check that an analytical solution is recovered when it lives in the finite element approximation space.

In each of the above tests, one should be careful that factors such as an insufficient number of quadrature points, a poor approximation of the domain geometry, loose solver convergence criteria, or even finite precision arithmetic are not responsible for failure of the tests.

To execute the fourth, fifth, and sixth tests above, one must have access to an analytical solution. There are two general ways of deriving analytical solutions for a given problem: (i) the method of exact solutions and (ii) the method of manufactured solutions.

In the method of exact solutions, one specifies material parameters, the loading, the geometry and boundary conditions and tries to analytically solve for the exact solution of the problem at hand. In the context of our 1D model problem, one specifies K, f, L, g_0, and g_L and tries to solve for u. For instance, if:

$$K = 1, \quad f = x, \quad g_0 = 0, \quad g_L = 0, \text{ and } L = 1$$

one can solve the problem:

$$-u_{,xx} = x \qquad \text{for } x \in (0,1)$$

$$u(0) = u(1) = 0$$

for u, attaining:

$$u(x) = \frac{x}{6}(1-x^2)$$

One appeal of the method of exact solutions is that one can choose material parameters, loadings, geometries, and boundary conditions that are physically reasonable and representative of those seen in engineering practice. Unfortunately, while analytically solving for the exact solution is possible for a wide range of material parameters (i.e., K), loadings (i.e., f), geometries (i.e., L), and boundary conditions (i.e., g_0 and g_L) for our 1D model problem, the same cannot be said for vector-valued, nonlinear, multi-field, and multi-dimensional problems. Even for linear scalar multi-dimensional problems, analytically solving for the exact solution is only possible for very simple material parameters, loadings, geometries, and boundary conditions, and even in these cases, the solution is often in terms of an infinite series expansion.

In the method of manufactured solutions, one instead specifies the solution, material parameters, and geometry and determines what the loading and boundary conditions must be. To demonstrate this in the context of our 1D model problem, suppose we choose:

$$u(x) = e^{-x}\sin(\pi x), \quad K = 1, \text{ and } L = 1$$

Then it must hold that:

$$f = -(k u_{,x})_{,x}$$

$$= -\frac{d}{dx}\left(\frac{d}{dx}\left(e^{-x}\sin(\pi x)\right)\right)$$

$$= -\frac{d}{dx}\left(-e^{-x}\sin(\pi x) + \pi e^{-x}\cos(\pi x)\right)$$

$$= (\pi^2 - 1)\, e^{-x}\sin(\pi x) + 2\pi e^{-x}\cos(\pi x)$$

$$g_0 = u(0) = 0$$

$$g_L = u(1) = 0$$

Unlike the method of exact solutions, the method of manufactured solutions can be used to generate analytical solutions for any choice of material parameters and geometry for both linear and nonlinear problems, scalar and vector problems, and one-dimensional and multi-dimensional problems. One can also construct analytical solutions of arbitrary complexity using the method of manufactured solutions. However, the method of manufactured solutions can produce non-physical loadings and boundary conditions if one chooses a non-physical analytical solution. Thus, one should use caution when interpreting the results of a verification test using a manufactured solution.

Of course, in practice, one should not execute just one verification test

to verify that a finite element code is working properly. Instead, one should instead employ an entire suite of tests, including multiple tests in each category of verification test presented earlier. In particular, one should execute the fourth, fifth, and sixth verification tests using:

i. Both polynomial and non-polynomial analytical solutions,

ii. Analytical solutions that depend on every spatial direction for multi-dimensional problems and depend on time for time-dependent problems, and

iii. Analytical solutions without special structure such as spatial or temporal periodicity or symmetries.

One should also test a finite element code using a variety of material parameter, loading, geometry, and boundary condition specifications to ensure the code does not only work for a subset of cases of engineering interest. However, one should be careful to choose material parameters, loadings, geometries, and boundary conditions that result in well-posed problems. For instance, one should not choose a negative viscosity for a fluid flow problem or a negative Young's modulus for a structures problem. These choices can yield a singular stiffness matrix for an otherwise stable and accurate finite element method.

Note that a polynomial degree k analytical solution is recovered by a finite element approximation of degree greater than or equal to k. Thus such an analytical solution cannot be used to confirm convergence rates for such an approximation. This is why one should conduct verification tests using both polynomial and non-polynomial analytical solutions.

It should be emphasized once more that numerical quadrature, solver convergence criteria, and finite precision arithmetic can significantly impact the results of a verification test. For instance, we never can realize actual convergence of a Galerkin finite element solution to the exact solution due to finite precision arithmetic. Instead, solution error will stall around machine precision once the mesh is sufficiently fine. As another example, a finite element code will not return the exact solution even if the solution lives in the finite element approximation space if integrals are not evaluated exactly. Geometry approximation can also greatly impact the results of a verification test. As we will learn later in this class, we often approximate the geometry by a polygonal domain in a finite element method. For most problems of engineering interest, the Galerkin solution still converges to the exact solution provided the polygonal domain converges to the true geometry under mesh refinement. However, this is not always true. For instance, if the solution for a simply-supported Kirchhoff-Love circular plate is approximated by a sequence of finite element solutions on successively refined polygonal

domains, the sequence converges to a function that is not the exact solution. This is referred to as Babuška's paradox.

Up to this point, we have discussed verification tests that gauge whether a finite element code in its totality is working as intended. Unfortunately, these tests do not reveal what part of a finite element code is not behaving as intended. To glean such insight, one needs to employ unit tests that test individual units of source code - sets of one or more computer program modules together with associated control data. Appropriate unit tests for a finite element code could include:

- ★ Unit tests that check whether or not certain polynomial functions are integrated exactly using the implemented quadrature rules in the finite element code.

- ★ Unit tests that check whether or not shape function values and derivatives are equal to known values at specific locations in the parent element.

- ★ Unit tests that check whether or not the element formation routine returns known element stiffness matrix and force vector entries for a particular element configuration and given material parameters and loading.

★ Unit tests that check whether or not the element assembly routine updates the correct global stiffness matrix entries and the correct force vector entries.

Unit tests should be conducted any time the finite element code is updated to ensure the code changes did not break basic subroutine functionality. If a unit test fails, one knows that a subroutine involved in the unit test has a bug.

Finite Element Calculation Verification:

To check whether a verified finite element code yields an accurate solution to a given problem, one often checks if one or more quantities of interest are grid independent. That is, one often checks if the monitored quantities of interest, which could include a component of the displacement field or stress tensor at particular spatial locations in structural finite element analysis, do not change under mesh refinement. However, the discrete solution may be still changing under mesh refinement even if the monitored quantities of interest do not change. Worse yet, even if the discrete solution does not change under mesh refinement, this does not guarantee that the discrete solution has already converged to the exact solution. It is possible that additional mesh resolution is required to capture highly localized solution features such as stress concentrations. Since we do not know what the exact solution is a priori for most applications, we do not

even know to look out for such solution features.

A much better approach to finite element calculation verification is to use so-called a posteriori error estimation. Namely, for many applications of interest, we can compute upper and lower bounds for certain norms of the solution error using the discrete solution without actually knowing what the exact solution is. These bounds can then be used to estimate what a particular norm of the solution error actually is. It is often possible to compute element-wise error estimates using these bounds as well, and these estimates can in turn be employed to inform local adaptation of the finite element mesh. In fact, so-called goal-oriented error estimation can be used to guide local mesh refinement to improve finite element estimates of desired quantities of interest. Unfortunately, a proper introduction to the topics of a posteriori error estimation and adaptive mesh refinement requires the use of powerful mathematical tools from functional analysis, so a further discussion on these topics is beyond the scope of this class. Instead, the interested student should consult the text "A Posteriori Error Estimation in Finite Element Analysis" by Ainsworth and Oden.