

BENCHMARKS

J. T. Oden

TICAM, The University of Texas

The notion of benchmarking a code refers to comparisons of results of the code to some standard “benchmark”, which may be an analytical solution or, as many argue, a single physical experiment. Various uses of the notion of benchmarking are found in the literature: for a brief summary, see Roache [1]. Among these are:

1. *Code-to-Code Comparisons of Code Performance.* Here execution speeds or numerical results, for example, of different codes solving one or more benchmark problems are compared on the same computer or on different computers running the same code.

2. *Code-to- Code Comparisons Versus Verified Code.* Here one assumes that a fully-verified code is in hand, and one compares results of different codes with those produced by the master code on a suite of benchmark problems.

3. *Comparison of Code Predictions to Results of Physical Experiments.* Here results of one or more codes are compared with results obtained by carefully executed physical experiments.

4. *Comparison of Code Results to Analytical or Highly Accurate Solutions.* Here a variety of features of basic algorithms in a code are tested by comparison of computed results with known analytic solutions or with “overkill” solutions obtained with very fine meshes using verified codes.

All of these approaches suffer from defects, some more serious than others.

Use 1 (Code-to-Code for Performance) has nothing to do with verification. Whether a code is slow or fast has little to do with whether it is solving the model correctly. To be sure, performance comparisons are useful and provide information on how fast Code *A* solves a problem compared to Code *B*, but both may be hopelessly inaccurate, solving incorrectly a given problem very quickly. Of course, if Code *A* is bug-free and Code *B* isn't, a comparison may reveal that at least one of them presents a problem. The use of Code-to-Code comparisons for judging accuracy is, in general, futile, unless one code is absolutely and thoroughly verified (an impossibility). (Incidentally, the only way to salvage some value in a code-to-code comparison for a verification view-point might be to tie the accuracy of the calculation to the effort needed to attain it: i.e. to determine the CPU time a test code would use to attain a solution of a specified accuracy. Even this is not strictly a verification test.)

Use 2 (Code-to-Verified-Code Comparisons) is not entirely useless, but is an exercise fraught with pitfalls. If the “verified” code is not as accurate as the tested code, how can useful conclusions be made by simply comparing results? Which is correct? It is easy to manufacture cases in which the base code, the verified code, is inferior in many

respects to the code or codes being tested. Such comparisons may be meaningless, without reference to other information, such as analytical solutions, but that is Use 4, not Use 2.

Use 3 (Comparison with Experiments), while a common benchmark scenario, is the worst of all. If discrepancies between computed and observed responses are observed, what is the source of error? Is the model bad or is the code inadequate or does the code have bugs? There is no systematic way to distinguish between modeling error and approximation error by a simple comparison of computed predictions of a single experiment (which itself is subject to gross error). One must first “verify the code”. How? Benchmarks? This is exactly the focus of this discussion. Thus, the argument supporting Use 3 is circular.

Use 4 (Comparison with Accurate Solutions) is somewhat better, but also imperfect. Exactly what should be compared? Here we come to the central question of benchmarking: What exactly are the comparisons with results of benchmark problems supposed to test? Going to the core of the issue: what do we want to know about a code to determine if it is “solving the model correctly”? The issue is often much more complicated than this: if we are given results of a “black box” commercial code where the precise model used to predict response is unknown (as it is propriety information, belonging to the code developer) what information can a suite of benchmark calculations provide?

We argue that : 1) benchmarking is primarily a verification exercise, not one of validation. Comparisons of computed predictions with experiments is the essence of validation—thus the subject code to be tested against experiments must have been already subject to verification test using, among other tools, benchmarking tests; 2) one must clearly set out tests designed to establish how well the subject code can solve the model used to depict natural phenomena or the behavior of engineering systems; 3) on the other hand, benchmarking need not necessarily be tied exclusively to the model used to develop the code, as the user has right to know where and when the model becomes invalid or inaccurate; 4) benchmarking, or the selection of a benchmark suite, is a delicate and complex proposition: tests on performance of a code on distorted meshes, on problems with nice smooth solutions, or on specific quantities of interest in a calculation must be made with a clear understanding of the capabilities of the model on which the code is based (we give examples below).

Some principles we recommend for designing benchmark problems are:

I. **Failure** - Benchmark problems should be designed to establish the limits of the code under consideration. Limits means failure: when does the code fail to convey or produce results in agreement with a suite of carefully-selected analytical solutions? Design the benchmarks so that bad results, not always glorious agreement, are obtained, by altering time steps, load steps, mesh size, approximation order, material mis-match, boundary-layers, point or line-loads, etc. Surely, good agreement on simple problems is necessary, but not sufficient to determine the real limitations of interest to the user.

II. **Convergence** - If a code produces pretty results but doesn't converge or doesn't converge at acceptable rates, then its use as a simulation tool is suspect at best. Also, the real meaning of the limit to which the code converges to is important. Benchmark problems should be designed to determine via numerical experiments the rates of convergence of a model as the mesh size, the order of the approximation, time step, load step, etc. are varied, convergence being measured in a space of meaningful norms: the energy norm for the model, L^1 and L^2 norms (or L^p -norms if appropriate) on the solution and its gradient, pointwise (L^∞) norms, etc. Whenever possible these should be compared with a priori estimates if they are available for the problem at hand. Convergence rates, in general, depend upon the regularity of the solution. A typical, a priori estimate is of the form,

$$\|error\|_s \leq C \frac{h^{\min(r-s, p+1-s)}}{p^{r-s}} \|u\|_r,$$

where C is a constant, h is a mesh size, p the local polynomial order of the approximation, and r a measure of *regularity* of the solution u . Here we are thinking of error measures of the type,

$$\|error\|_s = \left\{ \int_{\text{domain of problem}} \left[\left| \frac{\partial^s error}{\partial x^s} \right|^p \dots \right] d(\text{volume}) \right\}^{\frac{1}{p}}$$

$s \geq 0$, $1 \leq p \leq \infty$, but generally $p=2$, x being a spatial (or temporal) coordinate. One needs to select benchmark problems wherein r , the regularity of the solution, is low, high and, if possible, known precisely (this is why domains with cracks, corners, interfaces, changing boundary and initial conditions are useful in designing benchmarks to determine the rates of convergence observed on a sequence of meshes). Then a sequence of benchmarks can be designed to establish the rates-of-convergence for various regularities. Thus, if a solution u of a problem is known, the error can be calculated for each choice of mesh size and order, and a log - log plot of error versus problem size or h will yield the experimental rates of convergence.

As a corollary to this principle of convergence, we submit that a sequence of solutions (say, a sequence obtained by successive refinements of FE meshes) should converge to some meaningful limit. Unfortunately, limits obtained in such experiments often may have little to do with the model on which the numerical analysis is based. (For example, a sequence of triangular meshes used in the analysis of Kirchhoff plates converges in general, to a solution satisfying completely incorrect boundary conditions.) Thus, convergence studies must be made within the context of a well-defined mathematical theory. The benchmark problems must be well-posed, with correct and meaningful boundary and initial conditions, and when possible, the target solution of the model should be known to some acceptable degree.

Incidentally, the fact that error measures satisfy bounds of the type given above suggests that the idea of ever obtaining “mesh independent solutions” is pure folly. All such numerical solutions are mesh dependent – always (or dependent on other discretization parameters). Better terminology would be: “a sufficiently fine mesh is used to guarantee that the error is less than (say) one percent in the (say) energy norm.”

III. Local Behavior, Diffusion, Propagation - Users interested in simulating physical phenomena are concerned with how well the code in hand can depict so-called local phenomena. What are stresses at a point? How does the code handle convection of a signal propagating through a domain? What is the real versus numerical diffusion? Do signals propagate at predicted speeds? Are bifurcations, localizations mesh dependent? Or do the predicted critical parameters agree with those depicted in appropriate analytical solutions? These issues must be considered in designing benchmark problems.

IV. Model Limitations - Given a code (often a black box), design benchmark problems to fool or verify or, at least, determine the limitations of the model on which the code is based. This is not always easy. As a general rule, the design of effective benchmark problems (as a verification tool) should be based on models which supercede in complexity and sophistication those on which the code is based. Limitations of the subject code can be determined in a broader context. Then the limits of the model on which the code is based can also be roughly evaluated.

There are several example benchmark problems that illustrate this point. Consider the benchmark problem illustrated in Fig. 1 [2]: a linearly elastic, cantilever beam-like structure, fixed at one end, free at the other, is supported (constrained) at point B and subjected to a point load P at A . What are the reactive force R_B at B and the vertical displacements u_A and u_C at points A and C ? (Whether the problem is two- or three-dimensional is unimportant.) The answers are surprising. If the model used is two- or three-dimensional linear (or non-linear) the elasticity, the answers are:

$$\begin{aligned} R_B &= 0 \\ u_A &= \pm\infty \\ u_C &= \text{a finite number depending on the material properties, } P, \\ &\text{and the geometrical parameters.} \end{aligned}$$

Why? Because the constraint at B is applied to a single point, a set of measure zero. Any finite element approximation of this problem would produce a value of R_B of $R_B^h \neq 0$, but R_B^h should converge to zero as the mesh is refined. Any code based on two- or three-dimensional elasticity that would yield a sequence of solutions converging to $R_B \neq 0$ is wrong: it fails the test for verification. If $R_B \neq 0$ is attained, the model has been changed by some artificial fudging. The model cannot be verified when such criminal acts are committed. Also, $u_A = \pm\infty$, because a point source produces infinite energy in elasticity theory. This (u_A is infinite) is the correct solution for this model. The action of P is diffused over the distance from A to C : u_C is finite and computable. On the other hand, if Bernoulli-Euler beam theory is used as a basis for the model, then

$R_b =$ a finite number, depending on the material and geometric properties and on P ; u_A and $u_C =$ finite, dependent on similar parameters. Why? Because the energy now involves squares

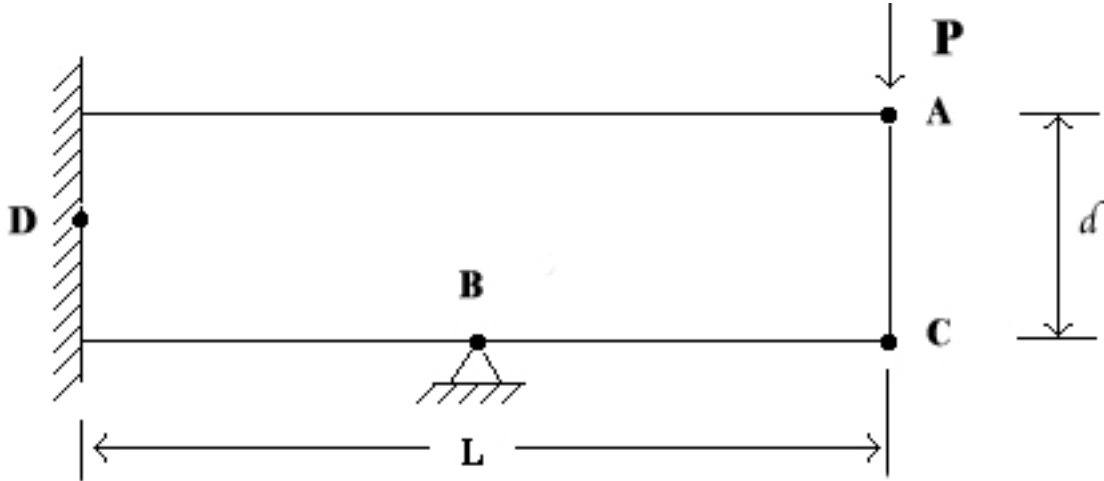


Figure 1: Benchmark problem.

of second derivatives of the displacement, so point loads and point supports are admissible as they provide finite energy.

The conclusion: the results of benchmark problems must be viewed and evaluated within the context of what is predictable by the theory upon which the underlying model is based. Once again, the goal of verification is to determine if the model is solved correctly. Of course, if we were to ask: “what is the stress at point D ,” these two models (elasticity and beam theory), may yield dramatically different results (depending on d/L , for instance).

There are countless examples of this type. The classical “pinched cylinder” problem of a circular cylinder under radial line loading P is another one. What is the displacement under P ? If a Kirchhoff shell theory is used, the displacement is finite; if a Reissner-Mindlin theory is used, it is infinite. Thus, model verification depends upon what can and should be deliverable by the model. Such examples suggest that when the base model used in a code is unknown, the benchmarks should be based on models presumed to be of the highest sophistication and breadth appropriate for the class of problems under consideration.

References

1. Roache, P., *Verification and Validation in Computational Science and Engineering*, Hermosa Publications, Albuquerque, 1998.
2. Babuska, I. and Oden, J.T., "Benchmark Computation: What is its Purpose and Meaning?," *IACM Bulletin*, Vol. 7, No. 4, October-December, 1992. "Benchmark Computations: Further Comments," *IACM Bulletin*, Vol. 10, No. 1, January-March, 1995.