MANUFACTURED SOLUTIONS WITH DISCONTINUITIES*

C. NATHAN WOODS AND RYAN P. STARKEY†

Abstract. The verification that computer codes correctly solve their model equations is critical to the growth and success of numerical simulation. The method of manufactured solutions (MMS) is the best method currently available for this kind of verification for differential equations, but it is limited, except in the case algebraic and differential equations with smooth solutions. An integrative method of manufactured solutions (IMMS) can extend the applicability of MMS to both discontinuous solutions and integral equations. The Busemann Advanced Concepts Lab has developed a tool for the easy implementation of IMMS to integral balance laws, and example applications to both the linear heat equation as well as the full Euler equations are presented.

Key words. verification, validation, manufactured solutions, integrative manufactured solutions, MMS, IMMS, Python, SciPy, Sympy, MASA

AMS subject classifications. 65M15, 65G99, 65R20

1. Introduction.

The use of numerical simulations and mathematical models to predict the behavior of real-world systems continues to grow in almost every current field of study, beginning with the sciences and engineering, but now reaching into economics, finance, and the social sciences as well. While these models and simulations have been instrumental in advances in many fields, they have also led to their own share of disasters. As the demands of the day become increasingly complex, it is becoming increasingly critical that researchers know precisely the limits and limitations of both their models and the computer programs or codes that they use to solve them. The limitations of numerical simulation arise naturally out of the physical and numerical approximations that are used to solve them, and so it is important to both understand and quantify the errors introduced by these approximations in order to use simulations effectively in the design, prediction, and analysis of real-world systems.

The classification scheme of Roache[5] is particularly useful for better understanding the errors inherent in a numerical simulation:

1. Errors that are a result of modeling approximations, such as incompressibility, continuity, and so on.
2. Errors that are ordered by some measure of the problem discretization
3. Errors that are the result of some other non-physical approximation (e.g. far-field boundary conditions)
4. Errors in programming, mistakes, or bugs
5. Errors that result from the representation of numbers on a computer (round-off error)

The study and quantification of modeling errors for a given application is generally referred to as model "validation". The study and quantification of the remaining numerical and programming errors is known as "verification". One common way to describe the difference between the two activities is, "Verification means solving the equations right, and validation means solving the right equations"[5]. Verification is further subdivided into two parts, code or software verification, and solution verification. Code verification shows that the code or software does solve the mathematical model correctly within some domain of inputs, while solution verification estimates the

*This work was supported by SOMEONE TBD

† Department of Aerospace Engineering Sciences, University of Colorado at Boulder, 429 UCB Boulder, CO 80309-0429, USA (charles.n.woods@colorado.edu).

expected error for a solution to a specific problem. This paper is primarily concerned with code verification.

Despite the overwhelming need for reliable verification of numerical simulations and scientific codes, the majority of such codes are verified principally by comparison with other unverified codes or by comparison with experimental data, both of which can fail to reveal important errors and behaviors. With the development of the powerful, general, and thorough method of manufactured solutions (MMS) at the end of the twentieth century, and the integrative method of manufactured solutions (IMMS) described herein, it is now feasible to convincingly verify any scientific code, but this sort of verification has still not become widespread. If this situation is to change, then thorough verification will have to become both simple and readily available to code developers. The Busemann Advanced Concepts Laboratory at the University of Colorado at Boulder has developed tools that make the implementation of IMMS both simple and an effective verification tool for users in industry, government, and academia alike.

2. Background. The most rigorous approach to verification commonly taken by developers today is to assemble a suite of exact solutions against which the code is compared[2]. If these tests do not show evidence to the contrary, then the code is assumed to be correct. Unfortunately, this process has no clear ending point, and therefore many such test suites are neither thorough nor complete, typically using simple exact solutions that do not fully exercise the governing equations, nor the boundary and initial conditions, nor the various alternative code paths (e.g. flux limiters) that might be encountered in real-world applications of the code.

Recent developments in code verification have favored an alternative approach to code verification which uses the more precise "code order verification" defined by Knupp[2], "The process by which one verifies the theoretical order-of-accuracy of the algorithm employed by the code to solve its governing equations." This is done by measuring the rate at which the solution returned by the code approach an exact solution as the computational grid is refined, and comparing the results to the theoretical rate that would be expected. This kind of testing is remarkably sensitive[5], but it relies on the ready availability of exact solutions that fully exercise the code as described above, so it is best paired with the method of manufactured solutions (MMS).

The basic idea of MMS is simple. Rather than finding a solution that fits an equation, one chooses a solution and derives the equation. A general description of the problem of mathematical modeling is to find solutions to the possibly vector-valued equation

$$f\left(u\left(x\right)\right) = 0 \tag{2.1}$$

This is very difficult to do, for general forms of f. A much simpler problem is to simply choose a useful form for $u\left(x\right)$, and derive a new equation

$$g\left(u\left(x\right)\right) = 0 \tag{2.2}$$

where $g = f + S$. If the code being verified is capable of handling the additional user-defined source terms S', then running the code with those source terms will yield the original, exact solution. Most importantly, the choice of u is almost completely arbitrary, and so it can be chosen to fully exercise all important aspects of the code. Knupp[2] gives the following guidelines for constructing useful manufactured solutions $u\left(x\right)$:

1. Manufactured solutions should be sufficiently smooth on the problem domain so that the theoretical order-of-accuracy can be matched by the observed order-of-accuracy obtained from the test.
2. The solution should be general enough that it exercises every term in the governing equation.
3. The solution should have a sufficient number of nontrivial derivatives.
4. Solution derivatives should be bounded by a small constant.
5. The manufactured solution should not prevent the code from running successfully to completion during testing. (Robustness is not part of verification)
6. Manufactured solutions should be composed of simple analytic functions.
7. The solution should be constructed in such a manner that any operators in the governing equations make sense.
8. Solutions should not grow in exponentially in time.

The demonstration of theoretical order of accuracy for such a manufactured solution constitutes code order verification as described above.

Consider an example. The Poisson equation is given by

$$f = \nabla \cdot (\nabla (u(x))) = S(u(x)). \tag{2.3}$$

Suppose a manufactured solution is chosen as $u(x) = x^2y^2$. Upon substitution of this solution into the Poisson equation with $S(u(x)) = 0$, the equation yields $f(u(x)) = S(u(x)) = 2y^2 + 2x^2$. The new source term $S + S'$ is then supplied into the Poisson solver, and the results are checked against $u(x)$.

The Center for Predictive Engineering and Computational Sciences (PECOS) at the University of Texas at Austin has developed and maintains the Manufactured Analytical Solution Abstraction (MASA) library[3], which provides a generalized interface for manufactured solution and source term evaluation. Like most MMS work to date, MASA focuses on application to the solution of partial differential equations.

The method of manufactured solutions works exceptionally well for differential equations, because the differential source terms can be computed easily and analytically, often through the use of computer algebra systems (CAS). Unfortunately, many differential equations of interest do not admit continuous solutions for certain physically admissible initial and boundary conditions, and this has been a problem for MMS for some time[3][6][2]. Currently, these situations are handled either in a two-step process which verifies the code against smooth manufactured solutions and then against discontinuous exact solutions, or else a manufactured solution is chosen that is not discontinuous but merely steep enough to appear as a discontinuity when discretized. However, for equations that admit such weak, or discontinuous solutions, a better option is available.

The principal difficulty with applying MMS to discontinuous solutions is that such solutions are not differentiable everywhere, so they cannot satisfy the differential equations for which MMS was developed. These are solutions only in the weak sense, which is to say that they satisfy the integral equations that result from integrating the original differential equations over some suitable computational volume. In reality, many numerical codes that solve differential equations (finite volume, finite element) actually work on the integral equations anyway. This provides a clear path to reliable code verification for discontinuous codes, using the integral–rather than the differential–form of the equations, and this method has no difficulty whatsoever with discontinuous solutions.

3. Manufacturing Integral Solutions. Consider a general form of integral equation that would be obtained by integrating a differential balance law over some volume:

$$dF\left(u\left(x\right)\right) = S\left(u\left(x\right)\right) \Rightarrow \int_{d\mathcal{V}} F\left(u\left(x\right)\right) = \int_{\mathcal{V}} S\left(u\left(x\right)\right) \tag{3.1}$$

In such an equation, the change in some quantity $u\left(x\right)$ contained within a volume $\mathcal{V}$ is given by the amount created/destroyed by sources/sinks S throughout the volume, combined with the flux F through the boundary of the volume $d\mathcal{V}$. For the case where $S = 0$, the total amount of $u\left(x\right)$ is conserved, and such equations are called conservation laws.

This form of integral balance law is extremely common, and is the basis behind finite volume and finite element codes for partial differential equations. For a code such as this, the differential form of MMS is not strictly applicable, and an integral form is more suitable for use in verification. This is especially true for equations which may admit discontinuous solutions, since the manufacture of such solutions for integral equations is both simple and straight-forward.

Given some suitable choice for $u\left(x\right) = f\left(x\right)$, one can compute the following:

$$\int_{d\mathcal{V}} F\left(f\left(x\right)\right) = \int_{\mathcal{V}} \left(S\left(f\left(x\right)\right) + S'\left(f\left(x\right)\right)\right) \tag{3.2}$$

This yields a new source term S' that can be added to an existing program that solves Eq. 3.1, and that will yield the exact solution $f\left(x\right)$. Exact solutions that will thoroughly exercise any terms or conditions are therefore available for any program for which the source terms can be specified by the user. This constitutes the integrative method of manufactured solutions (IMMS).

There are two ways to go about implementing the IMMS method. One can attempt to compute the source term analytically, which yields an exact, functional value for S', but it is frequently very difficult to carry out the necessary integration. Alternatively, one can leverage numerical integration tools, which have no such difficulties, at the cost of introducing numerical error. Fortunately, errors in a typical numerical simulation are many orders of magnitude greater than those encountered in numerical integration, which makes this an acceptable tool for many problems. This paper is concerned primarily with the numerical approach.

Once $\int_{\mathcal{V}} S'\left(f\left(x\right)\right)$ is found, one can compute S' by numerically differentiating $\int_{\mathcal{V}} S'$. The differentiated result may then be supplied to the code directly. This has the disadvantage of introducing yet more numerical error, but it is very simple to implement for many codes. Second, one can use an existing computational mesh to define $\mathcal{V}$ and compute $\int_{\mathcal{V}} S'$ directly for each cell defined by the mesh. This same mesh would be used by the code being verified. Either choice would be valid.

To summarize, the question of manufacturing solutions to integral equations consists of two steps:

1. Choose a suitable solution $f\left(x\right)$
2. Evaluate the necessary integrals in order to compute $\int_{\mathcal{V}} S'\left(f\left(x\right)\right)$
3. Either differentiate $\int_{\mathcal{V}} S'\left(f\left(x\right)\right)$ numerically to find S' or use the integrated form directly, as a source term in the code being tested, to force the return of the manufactured solution.

4. Implementation and Testing. The Busemann Advanced Concepts Lab has developed prototype software that implements integral manufactured solutions for integrated balance laws as in Eq. 3.1. `BACL-IMMS` is written primarily in the Python programming language, and leverages both SymPy (a computer algebra system) and SciPy (a numerical scientific library) in order to ease the process of computing manufactured source terms for any user-defined equation for any solution form.

`BACL-manufactured` consists of four main tools in addition to the two software libraries that work together to greatly simplify the computation of manufactured source terms for general equations. These tools are:

1. An n-dimensional integration tool `nquad` that recursively applies QUADPACK integration routines from netlib.org[1] to evaluate multidimensional integrals
2. A specialized integration module that greatly simplifies the use of `nquad`, parallel processing, and integration of vector quantities
3. A base equation class that provides functions for the simple integration of flux integrals, volume integrals, and complete balance integrals
4. Various subclasses that implement various specific equation sets

Items 1-3 are fairly straightforward, but it is worth noting that `nquad` is planned for inclusion in the next release of SciPy (version 0.13). Item 4 is the critical feature that makes `BACL-IMMS` expandable. In order to implement a new equation set, one simply needs to create a subclass of the base equation that converts a SymPy representation of the manufactured solution into the various flux and source terms in Eq. 3.1. Because of the flexible syntax Python provides, everything else can be handled easily without much intervention on the part of the user.

At this time, modules have been written for both the heat equation and the Euler equations. Provided with the limits of integration, a symbolic form of the manufactured solution, and a functional description of any discontinuities that may be present, the software will compute and return $\int_{\mathcal{V}} S'$.

IMMS has many potential advantages in the world of software verification. Unlike traditional MMS, it can easily handle discontinuous solutions, and it is a more faithful representation of the underlying mathematical model for codes that use the finite-volume and finite-element models. However, it must be demonstrated that IMMS is accurate enough to be used in code verification, and that requires its own testing. The obvious first test for a program of this type is to use it on known, exact solutions, for which S' is identically zero. The second test is to compare S' with published functional, non-zero forms.

4.1. The Heat Equation. The simple heat equation with constant material properties is given by

$$0 = \frac{\partial}{\partial t}\rho c_P u - k\nabla^2 u \tag{4.1}$$

$$\Rightarrow 0 = \int_{d\mathcal{V}} (\rho c_P u\, dxdydz - k\frac{\partial u}{\partial x}\, dtdydz - k\frac{\partial u}{\partial y}\, dtdxdz - k\frac{\partial u}{\partial z}\, dtdxdy) \tag{4.2}$$

A few exact solutions to this equation are given by:

1. $u = A\xi + B$
2. $u = A\left(\xi^2 + 2at\right) + B$
3. $u = A\left(\xi^3 + 6at\xi\right) + B$
4. $u = A\left(\xi^4 + 12at\xi^2 + 12a^2t^2\right) + B$
5. $u = A\exp\left(a\mu^2 t + \mu\xi\right) + B$

6. $u = A\exp\left(a\mu^2 t - \mu\xi\right) + B$
7. $u = A\exp\left(-a\mu^2 t\right)\cos\left(\mu\xi + B\right) + C$
8. $u = A\exp\left(-\mu\xi\right)\cos\left(\mu\xi - 2a\mu^2 t + B\right) + C$

[4], where A,B,C, a, and μ are arbitrary constants, and n is a positive integer.

The spatial coordinate ξ is taken as

$$\xi = \cos(\phi)\cos(\theta)\,x - \cos(\phi)\sin(\theta)\,y + \sin(\theta)\,z \tag{4.3}$$

subject to the constraint $\phi = \theta$.

The value of S' can be computed for varying values of ϕ in order to measure the accuracy of a particular IMMS implementation, and results for `BACL-IMMS` are shown in Fig. 4.1

The documentation for the MASA library discussed earlier includes both recommended solution forms and analytic source terms for many different equation sets. For the heat equation, they give:[3]

$$u = \cos(A_x x + A_t t)\cos(B_y y + B_t t)\cos(C_z z C_t t)\cos(D_t t) \tag{4.4}$$

with the source term S' given by

$$\begin{aligned} S' = \rho c_P \big(&\sin(A_x x + A_t t)\cos(B_y y + B_t t)\cos(C_z z + C_t t)\cos(D_t t)\,A_t \\ &+ \cos(A_x x + A_t t)\sin(B_y y + B_t t)\cos(C_z z + C_t t)\cos(D_t t)\,B_t \\ &+ \cos(A_x x + A_t t)\cos(B_y y + B_t t)\sin(C_z z + C_t t)\cos(D_t t)\,C_t \\ &+ \cos(A_x x + A_t t)\cos(B_y y + B_t t)\cos(C_z z + C_t t)\sin(D_t t)\,D_t\big) \\ &+ k\left(A_x^2 + B_y^2 + C_z^2\right)\cos(A_x x + A_t t)\cos(B_y y + B_t t)\cos(C_z z + C_t t)\cos(D_t t) \end{aligned} \tag{4.5}$$

I'm not super happy about using this one-off example, or with this section in general, but the code takes WAY too long to run. This one example took hours, so any kind of parameter sweep will take weeks, even on the office computer!

As stated earlier, the IMMS method does not directly yield S', but rather $\int_{\mathcal{V}} S'$. Therefore, the result of `BACL-IMMS` must be differentiated in order to compare directly with the results from MASA. Using a fifth-order, centered-difference formula to compute derivatives, agreement is found to within 10^{-10} for the specific case given by

$$A_x, A_t, B_y, B_t, C_z, C_t, D_t, \rho, C_P, k = 1.0, 1.0, 0.5, -0.25, 0.7, 0.0, 0.1, 1.0, 1.0, 1.0 \tag{4.6}$$

In the case of this simple, linear equation, it is clear that IMMS maintains more than the level of accuracy needed for the verification of most numerical codes.

4.2. The Euler Equations. The Euler equations of inviscid fluid dynamics are a much more interesting equation set for several reasons. First, they are a system of equations. Second, they are non-linear. Third, they specifically allow for discontinuous solutions in the form of shocks and slip lines.

The Euler equations are given by

$$\begin{aligned} 0 =& \frac{\partial E}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} + \frac{\partial H}{\partial z} \\ & \Downarrow \\ 0 =& \int_{d\mathcal{V}} (E\, dxdydz + F\, dtdydz + G\, dtdxdz + H\, dtdxdy) \end{aligned} \tag{4.7}$$

where

$$E = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho e \end{bmatrix} \; F = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \rho ue + up \end{bmatrix} \; G = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ \rho ve + vp \end{bmatrix} \; H = \begin{bmatrix} \rho w \\ \rho uw \\ \rho vw \\ 3\rho w^2 + p \\ \rho we + wp \end{bmatrix} \tag{4.8}$$

The problem defined by the initial data

$$W = \begin{cases} W_L; x < 0 \\ W_R; else \end{cases} \tag{4.9}$$

where $W = [p, \rho, u, v, w]$ and transmissive boundaries on all sides is named a Riemann problem. This problem has an exact solution which consists of various waves radiating from the initial discontinuity, possibly including shocks. For the specific initial data[7]

$$\begin{aligned} W_L &= [460.894 \quad 5.99924 \quad 19.5975 \quad 0.0 \quad 0.0] \\ W_R &= [46.0950 \quad 5.99242 \quad -6.19633 \quad 0.0 \quad 0.0] \end{aligned} \tag{4.10}$$

the solution consists of two shocks, bracketing a slip line, and is easily represented as a piecewise-constant function. When this solution is integrated with the significant figures shown above, errors are again on the order of the last significant figure, well within acceptable ranges.

5. Conclusion. As the role of numerical simulations continues to increase in all aspects of life, it becomes more and more critical that those simulations be completely trustworthy. Rigorous code verification using the method of manufactured solutions will continue to grow in importance as precise knowledge of the limits and limitations of computer codes become more and more important. The integrative method of manufactured solutions expands the capabilities of the original method, allowing the rigorous verification of both codes that solve integral equations and, by extension, codes that produce discontinuous solutions.

Since the analytic evaluation of integrals is often difficult, numeric integration techniques are recommended for the implementation of IMMS. The quality of verification achieved by this method is obviously dependent on the integration routines used, but it has been demonstrated that highly accurate options are readily available. Additionally, `BACL-IMMS` provides a convenient tool for implementing IMMS in the particular case of integral balance and conservation laws. It is hoped that these tools, together with the MASA library discussed above, will greatly simplify the process of rigorous verification for the developers of numerical simulation codes.

Although integral manufactured solutions are not as simple as their differential cousins, they nonetheless show great promise. Using high-quality integration tools, it is possible to place very tight bounds on the numerical error in computational science,

thus removing much of the guess-work that comes with writing and debugging code. This will greatly improve the quality of numerical results and the speed with which reliable codes can be generated, and it will be an invaluable tool in the quantification and estimation of uncertainty inherent in these numerical codes. This, combined with the increased importance of software validation, will lead to a reliable, mature, and trustworthy evolution of computational science.

REFERENCES

[1] AT&T Bell Laboratories, University of Tennessee, and Oak Ridge National Laboratory, *www.netlib.org*.

[2] P. Knupp and K. Salari, *Verification of Computer Codes in Computational Science and Engineering*, Chapman and Hall/CRC, 2002.

[3] Nicholas Malaya, Kemelli C. Estacio-Hiroms, Roy H. Stogner, Karl W. Schulz, Paul T. Bauman, and Graham F. Carey, *MASA : a library for verification using manufactured and analytical solutions*, Engineering with Computers, (2012).

[4] Andrei D. Polyanin, *Linear Heat Equation*.

[5] Patrick J. Roache, *Verification and Validation in Computational Science and Engineering*, Hermosa Publishers, 1998.

[6] ———, *Code Verification by the Method of Manufactured Solutions*, Journal of Fluids Engineering, 124 (2002), p. 4.

[7] Euletorio Toro, *Riemann Solvers and Numerical Methods for Fluid Dynamics*, Springer, 2009.