



AIAA S-141-2024

Standard

Standard for Code Verification in Computational Fluid Dynamics

AIAA standards are copyrighted by the American Institute of Aeronautics and Astronautics (AIAA), 12700 Sunrise Valley Drive, Reston, VA 20191-5807 USA. All rights reserved.

AIAA grants you a license as follows: The right to download an electronic file of this AIAA standard for storage on one computer for purposes of viewing, and/or printing one copy of the AIAA standard for individual use. Neither the electronic file nor the hard copy print may be reproduced in any way. In addition, the electronic file may not be distributed elsewhere over computer networks or otherwise. The hard copy print may only be distributed to other employees for their internal use within your organization.

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without prior written permission of the publisher.

Standard

Standard for Code Verification in Computational Fluid Dynamics

Sponsored by

American Institute of Aeronautics and Astronautics

Approved

11 December 2024

Abstract

This Standard explains the role of code verification in ensuring the reliability of predictive computations. It provides the relevant definitions, methods, and theory. It also provides practical guidance and worked examples of code verification. The Standard specifies the requirements for carrying out code verification and for checking whether existing code verification covers the use of a code for a particular application. It is intended for code developers and users, their managers, and those who make decisions using simulation results, and it identifies the relevant roles and responsibilities of these groups. It is the first AIAA Standard on this topic.

Published by
American Institute of Aeronautics and Astronautics
12700 Sunrise Valley Drive, Reston, VA 20191
Copyright © 2024 American Institute of Aeronautics and Astronautics.
All rights reserved.

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without prior written permission of the publisher.

Printed in the United States of America.

ISBN 978-1-62410-746-7

TABLE OF CONTENTS

Section	Page
FOREWORD	3
EXECUTIVE SUMMARY	5
1 INTRODUCTION AND OVERVIEW	6
1.1 WHAT IS CODE VERIFICATION AND WHY IS IT IMPORTANT?	6
1.2 SCOPE AND OBJECTIVES OF THIS STANDARD	9
1.3 OVERVIEW OF CODE VERIFICATION PROCEDURES	10
1.4 RESPONSIBILITY FOR CODE VERIFICATION	11
2 FUNDAMENTAL CONCEPTS IN CODE VERIFICATION	13
2.1 DEFINITIONS FOR DISCRETIZATIONS, NUMERICAL SCHEMES, AND NUMERICAL SOLUTIONS ..	13
2.2 ORDER OF ACCURACY OF DISCRETE SOLUTIONS	16
2.3 CALCULATING THE ERROR FOR CODE VERIFICATION	18
3 RECOMMENDED CODE VERIFICATION PROCEDURE	20
3.1 ESTABLISHING A CODE VERIFICATION PLAN AND ACCEPTANCE CRITERIA	20
3.2 TYPES AND SOURCES OF REFERENCE SOLUTIONS	22
3.3 THE USE OF SYSTEMATIC MESH AND TIME-STEP REFINEMENT	22
3.3.1 <i>Systematic Mesh Refinement</i>	22
3.3.2 <i>Time-Step Refinement</i>	23
3.3.3 <i>Elimination and Bounding of Other Sources of Error</i>	24
3.3.4 <i>Computational Models with Inherent Grid-Size Dependence</i>	25
3.4 MULTIPHYSICS AND COMPLEX COMPUTATIONAL MODELS	26
3.5 EXERCISING CAUTION WITH MODEL VARIANTS AND SPECIFIC IMPLEMENTATIONS	26
3.6 DOCUMENTATION AND ARCHIVING OF RESULTS	27
4 REQUIREMENTS, COMPLIANCE, AND TAILORING	28
4.1 DEFINITIONS FOR REQUIREMENTS, COMPLIANCE, AND TAILORING	28
4.2 REQUIREMENTS	30
4.3 COMPLIANCE	31
4.4 TAILORING	32
4.4.1 <i>Restrictions on Tailoring of This Standard</i>	32
4.4.2 <i>Example of Tailoring of This Standard by Modification of Requirement 2</i>	33
4.4.3 <i>Documenting Deviations from the Requirements of This Standard</i>	34
4.4.4 <i>Claiming Compliance When Tailoring Is Used for This Standard</i>	34
4.4.5 <i>Closing Remarks</i>	35
5 SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS	36
APPENDIX 1 EXAMPLES OF CODE VERIFICATION	38
APPENDIX 1.1 CODE VERIFICATION USING A TRADITIONAL ANALYTICAL SOLUTION	38
Appendix 1.1.1 <i>Additional Derivation Details of Appendix 1.1</i>	40
APPENDIX 1.2 CODE VERIFICATION USING THE METHOD OF MANUFACTURED SOLUTIONS	46
APPENDIX 1.3 CODE VERIFICATION USING A BENCHMARK COMPUTATIONAL SOLUTION	49
APPENDIX 1.4 OTHER CODE VERIFICATION EXAMPLES FROM THE LITERATURE	53
APPENDIX 2 RECOMMENDATIONS FOR CODE VERIFICATION PRACTICE	54
APPENDIX 3 THE METHOD OF MANUFACTURED SOLUTIONS	57

NOMENCLATURE AND SYMBOLS 59
 Subscripts 60
ACKNOWLEDGMENTS 61
REFERENCES.....62

Foreword

This Standard is intended for: (1) code developers and their managers; (2) researchers, analysts, and application engineers who use CFD codes for investigation, design, or analysis; and (3) managers who have the ultimate responsibility of making decisions based on the reliability of computational simulation results.

The Standard covers the two ways in which code verification arises in CFD practice, namely: (1) the process of performing code verification; and (2) the process of checking whether the code verification that has already been performed is adequate for the use of a CFD code for a particular application.

The AIAA Committee on Standards for CFD is the author of this Standard, AIAA Standard S-141-2024. At the time this Standard was completed and approved, the Committee had the following members:

1. Urmila Ghia, Chairperson
2. Sami Bayyuk
3. John Benek
4. Kenneth Bryden
5. Robert Bush
6. Terrence Conlisk
7. Sami Habchi
8. Ali Hadid
9. Robert Harris
10. Charles Hirsch
11. Mark Jurkovich
12. Hyung Lee
13. Edward Lynch
14. Mory Mani
15. Unmeel Mehta
16. William Oberkampf
17. Francisco Palacios
18. Dominique Pelletier
19. Joseph Powers
20. Christopher Roy
21. Christopher Rumsey
22. John Schaefer
23. Scott Shaw
24. Tom Shih

A consensus body of the above members approved this Standard in March 2024. The individual members may have voted for approval or disapproval of this Standard, or may have abstained.

The AIAA Committee on Standards for CFD is publishing this “Standard for Code Verification in CFD,” AIAA Standard S-141-2024, as a sequel to the “Recommended Practice for Code Verification in CFD,” AIAA Recommended Practice R-141-2021, that the Committee published in November 2021. This “Standard for Code Verification in CFD” supersedes and replaces the earlier “Recommended Practice for Code Verification in CFD.”

Relative to the Recommended Practice, this Standard contains updates and changes that reflect the evolution of the document from a Recommended Practice to a Standard, and contains clarifications and improvements throughout the document. The most important differences between this Standard and the Recommended Practice it replaces are the following:

1. The Standard addresses both the process of code verification and the process of checking whether code verification covers a particular application. The Recommended Practice addresses only the process of code verification.
2. The Standard specifies the requirements for the code verification process and the requirements for checking code verification coverage. The Standard also specifies the compliance provisions for these requirements and the allowed tailoring of the requirements. The Recommended Practice does not contain any of these elements.

Executive Summary

This Standard specifies the requirements for: (1) carrying out code verification for Computational Fluid Dynamics (CFD) codes; and (2) checking whether code verification that has already been carried out covers the use of the code for a particular application. This Standard also specifies the compliance provisions for the requirements and the allowed tailoring of the requirements.

The requirements of this Standard rest on a rigorous procedure for code verification. The Standard outlines this procedure and the minimum criteria that must be satisfied in the procedure to assert that code verification has been successfully carried out.

This Standard defines code verification and explains its role and importance within the overall framework of code and solution verification, model validation, and uncertainty quantification (VVUQ) in CFD. The document emphasizes the need for code verification as a necessary first step in the VVUQ process. It also identifies who should carry out code verification and who should assume responsibility for ensuring that appropriate code verification has been carried out before relying on the computational results produced by a CFD code.

This Standard provides and explains the main definitions relating to: (1) discretization of the mathematical model equations used in CFD codes; (2) discrete solutions and the methods for obtaining them; (3) errors in discrete solutions; (4) systematic mesh and time-step refinement; (5) formal order of accuracy; and (6) observed order of accuracy. The code verification process is then explained in terms of comparing the observed order of accuracy of the numerical solution with the formal order of accuracy of the discretization scheme used in the code. This document also outlines how round-off errors and iterative convergence errors can be accounted for in code verification and how they can be quantified and controlled.

This Standard also explains how a code verification procedure should be targeted for specific code features or combinations of features. This targeting is accomplished by designing the code verification procedure to cover all the terms of the mathematical model equations, methods, inputs, and parameters that are exercised in the code features or combination of features. The criteria to be used for assessing the outcome of the code verification process should also be chosen *a priori*. Establishing the observed order of accuracy requires determination of the error in the computational solution and this, in turn, requires the corresponding exact solution or a highly accurate numerical solution to be known. The following three types of reference solutions may be used for code verification: (1) traditional exact analytical solutions; (2) manufactured exact solutions; and (3) benchmark computational solutions. The use of each of these types of reference solutions is explained and demonstrated with an example. Sufficient detail is given for each of these examples to enable the process to be replicated for different test cases with different codes. This Standard also briefly discusses the main weaknesses and limitations of code verification, especially its lack of applicability to models with a built-in dependence on the mesh spacing, such as wall-modeled large eddy simulation (LES), and the difficulty of performing it for coupled or multiphysics codes and for model equations with high complexity.

The overall conclusion and recommendation of this Standard is that any claim of correctness of a code or a computational solution for a particular application should rest on adequate code verification, whether done by developers, software vendors, analysts, or other end-users. Furthermore, without code verification, reliance on computational results in making decisions incurs unquantifiable risks.

The body of this Standard is kept intentionally brief. Additional details are provided in appendices, and references to external literature are given for additional exposition and details.

1 Introduction and Overview

1.1 What Is Code Verification and Why Is It Important?

Computational Fluid Dynamics (CFD) has become an important and increasingly used tool in the design, analysis, and optimization of aerospace and other engineered systems and products. As the reliance on CFD grows, and as this reliance extends to assessments for certification, for safety and risk analysis, and for other high-consequence decision making, CFD code developers, modelers, and analysts are increasingly required to provide evidence for and to quantify the accuracy and reliability of the simulation results they generate and use [1, 2, 3].

A computational solution is typically sought, obtained, and used when the correct results (that is, the “real-world,” physically realizable results) are unknown. When a computational solution is used in this predictive mode, the most fundamental practical question that arises is the following: “how closely does the discrete computational solution approximate the corresponding actual physical reality?” In this predictive mode, determining the accuracy and reliability of a computational solution can normally be accomplished only by indirect or inferential means. These means are typically divided into verification [3, 4, 5, 6, 7], validation [3, 4, 5, 6, 7], and uncertainty quantification [7, 8, 9]. Verification is further divided into two separate branches: (1) code verification; and (2) solution (or calculation) verification [3, 6, 10]. Each of these activities has its own methods and techniques that have undergone significant study and development over the past couple of decades [11]. The remainder of this section defines and explains verification, validation, and uncertainty quantification, especially as they apply to CFD. In these definitions and explanations, use is made of various terms from CFD and numerical methods (such as *domain*, *computational algorithm*, *discretization*, and *discretization error*) that are defined or explained in later sections of this Standard.

Before defining verification and validation, it is helpful to recall the mathematical model equations that are implemented in CFD codes. These are typically the full potential, Euler, or Navier-Stokes equations in their integral, differential, or integro-differential forms, together with auxiliary models, such as models for the effects of turbulence and for constitutive relations. Additional equations may also be included to model the transport of species or particles, chemical kinetics, radiative heat transfer, and other physical phenomena or processes occurring along with the fluid flow. In this Standard, these equations and models are collectively referred to as the mathematical model equations or simply the model equations.

Code verification is the process of establishing the correctness of the computational algorithms and their implementation in a CFD code [3, 4, 5, 6]. Correctness here means that the computational algorithms and their implementation produce the correct computational solution to the model equations solved in the code. Correctness is attained when the computational algorithms are free from defects or shortcomings and when the implementation of these algorithms is free from coding errors. Correctness applies to all aspects of the computational algorithms and their implementation, including the following: (1) reading and assignment of the geometry, meshing, and other input data; (2) formulation of the discretized equations from the model equations; (3) programming of the discretized equations used in the interior of the domain; (4) programming of the discretized equations used on the boundaries of the domain; (5) programming of the coupling between the interior and boundary discretizations; (6) application of the boundary conditions; (7) iterative or direct solution of the systems of discrete equations; (8) the formulation and programming of the coupling of multiple model equations; (9) data exchanges and couplings between partitioned solutions in parallel processing; and (10) the generation of the final solution results in the desired form. Correctness here has no bearing on and makes no statement about the completeness or suitability of the model equations for a particular application or on how well these equations represent the physical reality they are intended to model.

Code verification is carried out for an individual feature or combination of features of a code, not for all features and all combinations of features at the same time. Accordingly, claims about code verification should be made for the features or combinations of features that have undergone code verification, not for the code as a whole, unless every feature and combination of features in that code has undergone code verification.

Code verification is most effectively carried out by quantifying the calculation error, which is the difference between the numerical solution generated by the CFD code and the corresponding exact solution to the model equations solved in the code. In generating numerical solutions for code verification, all sources of calculation error other than the discretization error should be minimized to ensure that the discretization error is the predominant contributor to the calculation error. This can be accomplished by converging the solution residuals to arithmetic round-off precision. The reasons for this are explained in Section 2.

Code verification is thus primarily a testing and quality assurance activity focused on the discretizations and solution algorithms adopted in a CFD code and on the programming of these discretizations and algorithms in the code. In the terminology adopted in this Standard, “code verification” could be expanded as “code correctness verification.” Code verification is specific to the CFD code on which it is performed and is an activity that usually has a defined end and outcome. The outcome of a completed and successful code verification program is an attestation to the correctness of the implementation in the CFD code of the portions of the code covered in the verification.

In this Standard, carrying out or performing code verification means that code verification (1) has been done properly, and (2) has successfully established that the tested code is correct and free of errors that affect the solution. It does not include the situation in which code verification was not done properly or when the code verification had a negative outcome.

Solution or calculation verification begins with the premise that code verification has been successfully carried out and seeks to estimate the errors in a computational solution when no exact solutions are available. These errors may arise from various sources, including the following: (1) the use of finite mesh spacings and time step sizes; (2) boundary condition implementations; (3) iterative tolerance (iterative convergence errors); (4) the storage of floating-point numbers with finite precision (round-off errors); and (5) other artifacts of the computational process, such as those arising from domain decomposition and parallel processing [3, 4, 5, 6].

Solution verification is thus an error estimation activity focused on the suitability of the choices the user made with the code for a given simulation, including choices for the mesh resolution, the solution parameters, and the iterative convergence criteria. In the terminology adopted in this Standard, “solution verification” could be expanded as “verification of the sufficiency of the solution accuracy for the intended application.” Solution verification applies to a computation performed for a particular test case with a particular combination of verified features in a code when no exact solutions are available. The outcome of a solution verification exercise is an estimate of the errors in the numerical solution produced for the test case relative to the exact solution of the model equations and an assessment of the sufficiency of the accuracy for the intended purpose.

It is helpful to observe that the term verification is used differently in code verification and solution verification. In code verification, the term refers to verifying correctness. In solution verification, the term refers to evaluating the accuracy and checking its sufficiency for the intended purpose of a simulation.

Validation is the process of quantifying the accuracy and completeness of the model equations that are discretized and solved in the CFD code relative to the corresponding experimental observations [4, 5]. Validation is normally carried out by comparing verified computational solutions with the corresponding experimental results, and such a comparison must take into account the accuracy of both the computational solution and the experimental measurements. The validity envelope of a

model can be expanded and corroborated from validations done with different experiments and different verified CFD codes in which that model is implemented. Validation is typically an ongoing activity that has no defined end, and every additional validation experiment provides a new data point in the validation domain.

The distinction between verification and validation is often expressed as follows: verification is concerned with solving the chosen model equations *right*, while validation is concerned with choosing and solving the *right* model equations. Thus, verification does not attempt to determine whether the model equations provide an accurate representation of reality, and validation does not attempt to determine whether a code generates the correct numerical solution to the model equations implemented in the code.

Although the precise definitions of verification and validation vary across the literature and even between formal standards issued by different organizations, the definitions given above can be viewed as consensus definitions that are widely accepted and used in the CFD community and more generally in the scientific computing community.

Uncertainty Quantification (UQ) determines the uncertainty in a CFD solution from uncertainties in the modeling and in the input variables, including uncertainties in the following: (1) the modeling formulation and modeling approximations for the problem of interest; (2) the specifications of the domain and boundary geometry, including manufacturing tolerances; (3) the specifications of the boundary conditions; (4) the material properties (including the fluid properties), or the constitutive relations; (5) the discrete solution process; and (6) the extrapolation of the model equations beyond their range of validation [4, 7, 11]. Uncertainty here means lack of complete knowledge or full precision in a model or in the value of a parameter or input variable.

Verification (which includes code verification and solution verification), validation, and uncertainty quantification (VVUQ) together aim to establish the reliability of predictive computations for an application and to estimate the error bounds of these computations for that application. The importance of code verification follows from its role as a prerequisite to solution verification, validation, and UQ.

Returning to the question of “how closely does the discrete computational solution approximate the corresponding actual physical reality?” it can be seen that the division between verification, validation, and uncertainty quantification reflects an assumption that the answer to the question can be decomposed into three parts. These are: (1) code and solution verification answers the sub-question of whether the chosen model equations are discretized correctly (code verification) and whether the discrete equations are solved with sufficient accuracy for the intended application (solution verification); (2) validation answers the sub-question of whether the chosen model equations faithfully represent the real physical behavior and phenomena being simulated; and (3) uncertainty quantification calculates the estimated uncertainty in a computational solution under specified conditions. Thus, verification and validation are both needed (together) to provide confidence in the accuracy of a computational solution relative to the corresponding physical reality. In contrast, UQ is needed to include and combine the effects of all the relevant uncertainties in the solution to ensure sufficient reliability in the solution for the target application.

This Standard is focused on code verification. For this reason, solution verification, validation, and UQ are not discussed further here. Despite the narrow focus of this Standard, it should be borne in mind that code verification is only the initial step in establishing the quantified accuracy and reliability of a computational solution. The accuracy, reliability, and fitness for purpose of the computational solution for the application of interest cannot be established without the remaining steps, namely, solution verification, validation, and UQ.

1.2 Scope and Objectives of This Standard

This Standard is written for developers and users of CFD codes, their managers, and those who make decisions based on results from CFD simulations. This Standard provides the relevant definitions, theoretical background, and computational procedures. It also provides generally accepted procedures for code verification. The document emphasizes practical aspects of the code verification process. The main body of the document is intentionally brief, with details and derivations relegated to appendices, and with references provided for additional or more detailed coverage of the subject.

The scope and objectives of this Standard are as follows:

- Explain what code verification is and explain its role in deciding whether a computational code is fit for a particular purpose.
- Explain and increase awareness of the importance and need for code verification, especially when the CFD code will be used in mission-critical or high-consequence decision making.
- Identify who is responsible for ensuring that code verification has been carried out and who should carry out the actual work of code verification.
- Review the theoretical background and the discrete solution criteria that are relevant for code verification, including the definitions of consistency, stability, convergence, and order of accuracy.
- Explain how to plan a code verification program, starting from the selection of a suitable reference solution, all the way to compiling, interpreting, and documenting the results.
- Provide guidance on how to determine the observed order of accuracy.
- Demonstrate how to carry out code verification for each of the three acceptable categories of reference solution; namely, (1) traditional exact analytical solutions; (2) exact manufactured solutions; and (3) benchmark computational solutions.
- Provide references to literature for more detailed coverage of code verification.

The concepts and practice of code verification have matured to a level that allows the formulation of a systematic process and an engineering standard for code verification. Any systematic process and any standard for code verification should nevertheless still allow for improvisation for special cases and for handling various potential complications.

The terminology and the presentation in this Standard are aimed at finite difference, finite element, finite volume, and spectral element methods. The general concepts presented here are applicable for other, less common discretizations, like the Lattice Boltzmann, smoothed particle hydrodynamics, and particle-in-cell methods, but their specific verification techniques are not explicitly covered.

This Standard addresses code verification for flow solvers. Code verification for pre- and post-processing codes, tools, and scripts (which will typically have their own interpolation, approximation, discretization, and other errors) requires verification techniques that may differ from the techniques used for flow solvers, and these techniques are not specifically addressed in this Standard. It should nevertheless be stated that code verification for a complete computational modeling workflow requires independent code verification for all pre- and post-processing components and any external libraries that are used with a flow solver.

This Standard is focused on CFD codes. However, it also covers code verification for general multiphysics codes, and most of the concepts and methods presented in the Standard are equally

applicable to codes for other categories of computational mechanics. Examples include codes for solid mechanics, electromagnetics, and radiative transport.

The Standard focuses on code verification, including its theory, methods, and practices. It does not cover the use of code verification in solution verification, or uncertainty quantification, or in the assessment of suitability of a code for an application.

This Standard is organized as follows: Section 1 defines code verification, explains its importance, and outlines its role within the overall VVUQ process for CFD. It also identifies the individuals who should carry out code verification and those who should ensure that it has been carried out before using computational results. It also outlines the objectives, contents, and scope of this Standard. It also provides a brief overview of the code verification process. Section 2 provides the theoretical background and the basic definitions and concepts related to code verification. Section 3 provides the recommended code verification process for CFD codes and explains the criteria for successful code verification. Section 4 specifies the requirements of this Standard (for code verification and code verification coverage), the compliance provisions, and the allowed tailoring methods. Section 5 provides a summary of this Standard and its main conclusions and recommendations. Appendix 1 provides examples of code verification using exact analytical solutions, exact manufactured solutions, and benchmark computational solutions, and discusses how to choose such solutions for code verification. Appendix 2 presents recommendations to managers, users, and developers seeking to adopt sound and effective code verification principles and practices. Appendix 3 outlines the method of manufactured solutions and its use in code verification.

1.3 Overview of Code Verification Procedures

As explained in Subsection 1.1, code verification is a process for assessing the correctness of both the discretization of the model equations implemented in a CFD code and the programming of this discrete representation into the complete solution algorithm. Therefore, code verification is a means to establish that a CFD code correctly represents the model equations in discrete form and a means to establish that the code is free of errors that affect the discrete solution. These two forms of correctness are simultaneously and effectively established by comparing the *observed order of accuracy* of the computational solutions generated by the code with the *formal order of accuracy* of the code [3, 6]. There are theoretical reasons underlying the effectiveness of this comparison, and there is also much accumulated experience showing that this comparison is highly effective at revealing subtle algorithmic, coding, and implementation errors that are often missed by comparisons or code verification activities that do not include the comparison between the formal and observed orders of accuracy [3, 6].

It should be emphasized here that the purpose of code verification is not to determine the order of accuracy of code. Rather, the purpose of code verification is to determine whether the code is free from defects that affect the solution, and this is most effectively accomplished by comparing the observed and formal orders of accuracy. The determination of the order of accuracy is, however, a valuable outcome of the code verification process. This is because the order of accuracy is necessary for solution verification, and is needed in many other activities, ranging from *a priori* estimation of suitable mesh spacings, to estimation of computing resources, to designing and setting up simulations and couplings between different codes.

The *observed order of accuracy* is determined by repeating the computational solution on a sequence of meshes with *systematically* decreasing mesh spacing and determining the rate at which the error in the discrete solution decreases with decreasing mesh spacing [3, 6, 10]. For transient calculations, the rate at which the error decreases with decreasing time-step size should also be found to determine the observed order of accuracy in time. The *formal order of accuracy* is the theoretical order of accuracy for the discretization scheme(s) used in the computational solution; it is the rate at which the error in the discrete solution should decrease with *systematically*

decreasing mesh spacing and decreasing time-step size [3, 6, 10]. The requirements for systematic grid refinement are detailed in Subsection 3.3. The error in a discrete solution is the difference (in a suitable norm) between the discrete solution and the corresponding exact analytical solution. Thus, determining the error in a discrete solution requires knowledge of the corresponding exact solution to the model equations.

When the model equations used in a code are relatively simple, hand derivation or other traditional methods of obtaining exact analytical solutions can be used. However, exact analytical solutions to the Euler and Navier-Stokes equations are rare. When such solutions do exist, they are often only for simple flow physics, simple geometry, and limited dimensionality. Hence, such solutions may not serve to verify all parts of a code or the more complex interactions that may occur in the algorithm for more complex flows. The method of manufactured solutions (MMS) provides a means to generate exact solutions for systems of model equations having almost arbitrary complexity [3, 6, 10]. However, the MMS requires the application of source terms and boundary and initial conditions, as further explained in Appendix 3. Applying such terms and conditions requires either access to the source code, which a user typically does not have with proprietary or commercial CFD codes, or the availability of user-defined function facilities that allow the programming of these terms and conditions into the code. The MMS was originally limited to differentiable solutions and simple couplings. However, recent extensions and generalizations to the original MMS have extended its applicability to non-differentiable solutions and solutions with complex multiphysics couplings, as further outlined in Appendix 3. A third option is benchmark computational solutions that have been generated with a verified code and with rigorously established numerical accuracy [3, 7]. Benchmark computational solutions are typically used when traditional analytical solutions are not available and when the MMS cannot be used (either because user-defined functions are not available, or because access to the code source is not possible, or because the code verification is to be carried out strictly without changing the source code). Examples of each of these three categories of code verification solutions are presented in Appendix 1.

In addition to comparing the formal and observed orders of accuracy, code verification would ideally include other tests of correctness. Common tests of correctness check for satisfaction of fundamental physical principles, such as translational invariance, conservation, maintenance of symmetry, and preservation of uniform flow. Such tests, while valuable and even necessary, do not on their own constitute rigorous code verification and do not on their own reveal all errors in a code.

The code verification process becomes more elaborate when a CFD code has a mixed formal order of accuracy, that is, when different terms in the model equation (which is usually expressed as a partial differential equation (PDE) or an integral equation) are discretized with different orders of accuracy. Such situations are discussed in more detail in Sections 2 and 3.

In order for code verification to reveal any discretization errors, it is necessary to ensure that other errors (such as iterative-convergence errors and round-off errors) are sufficiently small, and this can be accomplished by converging the residuals to machine precision.

1.4 Responsibility for Code Verification

Code verification for a CFD code should be initiated, carried out, and documented to an appropriate level by the developers of the code. The developers of a code, whether working with the quality assurance (QA) engineers or alone, and regardless of who takes the lead in code verification, are the single group that should have the greatest role in code verification. This is because developers: (1) are the most qualified to ensure that the code is working as they intended it to work, correctly and with no errors; (2) have full access to the source code; (3) are the most familiar with the implementations and discretizations in the code; and (4) are the most qualified to explain anomalies and to correct any errors discovered during code verification. Ideally, code verification should be made an intrinsic part of the development and QA processes for a code, and should therefore be

included in the development and QA work plans and time schedules, even if it has not been specifically requested. Code verification should also be carried out on an ongoing basis as part of the maintenance and extension of a code throughout its lifecycle. The code verification test cases and results should also be made available for inspection and independent reproduction by users of the code, preferably in separate repositories with their supporting scripts and post-processing tools to enable routine and automated reproduction of code verification results. Developers can also support code verification activities by providing the required functionality for users to specify source terms and boundary conditions for code verification using the MMS. Developers should also publish the formal order of accuracy for the discretizations of the individual terms in the model equations as well as the formal order of accuracy of the implementation of all boundary conditions, as outlined in Section 2.1.

The users of a CFD code should conduct their own code verification for their specific application whenever the available code verification is not sufficient for that application. If the available code verification is sufficient, users are responsible for auditing, checking, analyzing, or even reproducing some of the developers' verification results. The purpose of this is to confirm the adequacy of coverage of these verification results for the users' intended applications and for the specific combination of operating system, hardware, and other tools (such as pre- and post-processing tools and external libraries and scripts) being used in their complete workflow. Users also have responsibility to review relevant software author reports and other literature to ensure that any known errors in the code do not adversely affect their specific application. The developer, supplier, or vendor of a CFD code (whether commercial, open-source, academic, government lab, government agency, or otherwise) may state that code verification has been successfully conducted. However, experience has shown that the levels of detail, comprehensiveness, critical testing, and relevant documentation for code verification are not always adequate. As a result, it is commonly necessary for the end users of a CFD code to perform their own code verification as well, or to ask the supplier for more details on its testing. The two most common deficiencies found in code verification testing are the following: (1) the suite of test problems used in the verification testing do not exercise some components of the code, some terms of the discrete equations, or some auxiliary models that comprise essential elements in the user's specific application; and (2) the verification testing only evaluated the numerical accuracy and not the observed order of accuracy, even though it is well known in the literature [3, 6, 10] that evaluating the observed order of accuracy is stronger and more informative than evaluating the numerical accuracy for a test case.

A decision maker or manager who wishes to use CFD simulation results should specify the purpose and the credibility requirements for these simulations and should provide the resources and time needed for code verification. Those producing the CFD simulation results for the decision maker or manager should take responsibility for quantifying the credibility and accuracy of the results that they produce. Typically, this quantification cannot be accomplished without ensuring that appropriate code verification has been performed (whether by the supplier of the code, by those producing results for the decision maker, or by others). The producers of the CFD simulation results should also endeavor to document the code verification procedures that were relied upon for future reference and auditing purposes.

Developers, testers, users, managers, and decision makers should be aware that the accuracy, reliability, and soundness of computational results from a code that has not undergone adequate code verification are questionable. Accordingly, decisions based on these results are also questionable and entail risks that are difficult to quantify.

Carrying out code verification or even just reviewing the code verification work done by others to ensure its adequacy for an application increases the costs of using CFD. However, this cost can be greatly diminished by good planning and by making code verification an integral part of the development and use of a CFD code. Appendix 2 gives recommendations to managers, users, and developers to incorporate code verification principles and practices into their regular workflows. Wide adoption of these recommendations will allow the community to reap the benefits of code verification efficiently and cost effectively.

2 Fundamental Concepts in Code Verification

This section explains and defines terms related to the discretization that is implemented in a CFD code for the model equations and any included auxiliary models. It also explains and defines terms related to the errors that are introduced by the corresponding discrete approximation and by the numerical solution process. This section also clarifies how these errors relate to and are treated in the code verification process.

2.1 Definitions for Discretizations, Numerical Schemes, and Numerical Solutions

A CFD code solves the discrete analog of the analytical model equations that are represented in it, and typically does so inexactly. Let $f_{calculated}^{discrete}$ denote the actual numerical solution calculated and output by a CFD code for a given test case that is run with given temporal and mesh spacings and given solution control and other parameters. Let $f_{exact}^{discrete}$ denote the corresponding exact solution to the discrete set of equations that are implemented in the code. Let $f_{exact}^{analytical}$ denote the corresponding exact solution to the analytical model equations that are represented in the code, even if this exact solution is not known.

The *error* in a numerical solution, E , generically refers to the difference between a discrete solution and a uniquely defined reference solution. The most common and most useful reference solution is the exact analytical solution to the analytical model equation.

The *discretization error*, $E_{discretization}$, is defined as the difference between the exact solution to the discrete equations and the exact analytical solution to the analytical model equation [3], that is,

$$E_{discretization} = \|f_{exact}^{discrete} - f_{exact}^{analytical}\| \quad (1-a)$$

in which $\|x\|$ denotes an appropriate norm of the quantity x , and all other terms are as defined above. Suitable norms are presented in Section 2.3. The discretization error is a property of the numerical scheme [12], is independent of the code implementation, and defines the truncation error as well as the formal order of accuracy. It may be computed for the dependent variables in the solution and for solution functionals, such as the lift, the total pressure loss across a flow path, or the combustion efficiency.

The *calculation error*, $E_{calculation}$, is defined as the difference between the calculated solution to the discrete equations and the exact analytical solution to the analytical model equation, that is,

$$E_{calculation} = \|f_{calculated}^{discrete} - f_{exact}^{analytical}\| \quad (1-b)$$

in which all terms are as defined above. The calculated solution is the solution that is calculated and output by the code with the temporal and spatial spacings and the solution control and other parameters that were chosen for the computation.

Iterative error arises in steady-state problems when the evolving iterative numerical solution is not sufficiently converged to satisfy the exact solution of the discrete equations [3]. For unsteady CFD problems, iterative error is often associated with sub-iterations used to recover time accuracy for implicit methods.

Round-off error is the error in the numerical solution due to floating-point numbers being represented using a finite number of digits on digital computers [3].

Boundary-condition error arises from the numerical discretization of the physical and numerical boundary conditions required for correct representation of the analytical model.

The sum of the iterative error, the round-off error, and the boundary-condition error will sometimes be referred to as “*other error sources*” in the remainder of this document.

The calculation error is composed of the contributions of the discretization error, the iterative error, the round-off error, and the boundary-condition error. In non-deterministic simulations, the calculation error will also include the statistical sampling error.

Code verification is concerned with the characterization of the calculation error for a specific feature in a code, and especially with how this error varies with discretization spacings. Solution verification (or calculation verification) is concerned primarily with estimating the calculation error for a specific test case with a specific code using the chosen mesh and the chosen solution control settings when no exact solutions are available. Reliable solution verification requires code verification as a prerequisite.

Consistency is a property of a numerical scheme. A numerical scheme (and its included discretization) is consistent with a model equation (such as a PDE or an integro-differential equation) if and only if the numerical scheme (and its included discretization) arbitrarily approaches the model equation as the grid size and the time-step size are reduced to zero [12]. Thus, consistency is typically established by performing a truncation error analysis [12], whether by hand or by symbolic manipulation software. It does not involve determination of the actual errors in a discrete solution, but only an expression for the error.

Stability is a property of a numerical solution. A numerical solution is stable if and only if the difference between the numerical solution and the exact solution to the *discrete equations* (including all contributions to this difference from round-off errors, set-up errors, and other errors) does not grow without bound as the numerical solution is advanced an infinite number of times (in iteration count or in time) towards the final solution [12]. This definition effectively requires the numerical solution process to limit the amplification of any error from any source in applying or solving the numerical scheme. In practice, most numerical solutions are stable only under certain conditions, and a numerical solution should be operated within these conditions to retain stability.

Convergence is a property of a numerical solution. A numerical solution is convergent if and only if the error in the numerical solution tends to zero as the grid and time-step spacing tend to zero [12]. Referring to Equation (1-b), convergence requires that as the mesh spacing on which the discrete solution $f_{calculated}^{discrete}$ is calculated tends to zero, the error $E_{calculated}$ also tends to zero. From a practical point of view, convergence can be regarded as the most valuable property since it is the one that ensures that the numerical results generated from a CFD code can be driven arbitrarily close to the corresponding analytical solution solely by decreasing the grid and time-step spacings.

The convergence characteristics and the stability of a solution are both affected by the iterative error, the round-off error, and the boundary-condition error.

Consistency, stability, and convergence are related [13, 14]. For example, it can be deduced from the definitions given in the preceding three paragraphs that a numerical solution that is not stable cannot be guaranteed to produce a converged numerical solution. Similarly, a numerical scheme that is not consistent cannot be guaranteed *a priori* to yield a converged numerical solution. These conclusions are valid for any numerical scheme and any numerical solution. For linear numerical schemes applied to well-posed initial-value problems, the Lax Equivalence Theorem [14, 15, 16] proves that if a consistent discretization is used, stability is equivalent to convergence (that is, stability implies convergence and convergence implies stability). For nonlinear problems, this equivalence has not been proven, but CFD practice nevertheless generally assumes that consistency and stability are necessary but not sufficient for convergence.

The properties of consistency, stability, and convergence are embodied in Equation (1-c) below, which follows from Equation (1-b) and which also mathematically expresses the Lax Equivalence Theorem for linear problems.

$$\begin{aligned}
 E_{\text{calculation}} &= \|f_{\text{calculated}}^{\text{discrete}} - f_{\text{exact}}^{\text{analytical}}\| \\
 &= \|f_{\text{calculated}}^{\text{discrete}} - f_{\text{exact}}^{\text{discrete}} + f_{\text{exact}}^{\text{discrete}} - f_{\text{exact}}^{\text{analytical}}\| \\
 &\leq \|f_{\text{calculated}}^{\text{discrete}} - f_{\text{exact}}^{\text{discrete}}\| + \|f_{\text{exact}}^{\text{discrete}} - f_{\text{exact}}^{\text{analytical}}\|
 \end{aligned} \tag{1-c}$$

The term $\|f_{\text{calculated}}^{\text{discrete}} - f_{\text{exact}}^{\text{discrete}}\|$ in Equation (1-c) is precisely the sum of the sources labeled “*other error sources*,” and this term implicitly defines the stability condition of the scheme’s implementation. If the scheme is stable, this term decreases at a rate determined by the spectral radius of the amplification matrix for the scheme, as defined, for example, by a Von Neumann analysis. The term $\|f_{\text{exact}}^{\text{discrete}} - f_{\text{exact}}^{\text{analytical}}\|$ in Equation (1-c) is the discretization error, as defined in Equation (1-a). Equation (1-c) can therefore be re-expressed in the form

$$E_{\text{calculation}} \leq \| \text{other error sources} \| + E_{\text{discretization}} \tag{1-d}$$

The behavior of the left-hand side of Equation (1-d) with decreasing mesh and time-step spacings determines the convergence of the numerical solution. The behavior of the first and second terms on the right-hand side of Equation (1-d) with decreasing spacings respectively determine the stability of the numerical solution and the consistency of the numerical scheme. Thus, Equation (1-d) expresses the Lax Equivalence Theorem, namely, that if the scheme is consistent and stable, then it is convergent, and all three error terms decrease with the spacings. In other words, when mesh spacings are reduced towards zero, both the discretization error and the stability error tend to zero, guaranteeing that the calculation error also tends to zero.

The *truncation error*, E_t , is a property of the numerical scheme and is an analytical expression of the difference between the discrete equations (that is, the discrete analog of the model equations represented in the code) and the model equations [12]. The truncation error is typically given in the form of a power series expansion, whether the discrete form employs finite difference, finite volume, finite element, or spectral method discretization. For model equations that are given as PDEs and when a finite difference or finite volume discretization is used, the truncation error is most readily found by a Taylor Series expansion of the individual terms in the discrete form of the model equations [12]. In simplified form, this expansion is typically expressed by

$$E_t = Ch^p + H.O.T. \tag{2}$$

where C is a constant that is given an upper-bound value, p is a constant, h is the local mesh spacing, and the term $H.O.T.$ denotes higher-order terms (that is, terms that have h raised to powers greater than p). It should be noted that the values C and p may vary from point to point in the computational domain, depending on the specific discretizations used throughout the domain. The truncation error determines the consistency of a numerical scheme. For a numerical scheme to be consistent, every term in its truncation error should include multiplication by h^p with $p > 0$ at every discrete point in the computational domain. The term *truncation error* is also commonly used to denote the actual numerical value of the truncation error.

Even though Equation (2) is written for a PDE, analogous or similar forms exist for any type of model equations (such as an integral or integro-differential set of model equations) and for any discretization, including finite element, spectral element, and boundary-integral discretizations. The essential feature of the truncation error is that there is typically a dependence of the truncation error on an exponent of the local mesh spacing, h . For some discretizations, deriving the error explicitly in the form of Equation (2) may not be easy, but it is nevertheless possible. The discretization error

gives a good estimate of the value of the truncation error which, as just explained, is normally expressed in algebraic form rather than as a numerical value.

The result expressed in Equation (2) is for the special case in which the spacings of cells, elements, or points are locally isotropic (that is, are the same in all coordinate directions, or have an aspect ratio of one) and are spatially homogeneous (that is, have a uniform size throughout the domain). However, in most practical applications, these spacings are neither locally isotropic nor spatially homogeneous. In such cases, Equation (2) takes a more general form to account for both local anisotropy and spatial inhomogeneity. Without this generalization, the observed order of accuracy on any mesh with local anisotropy or spatial inhomogeneity will typically fall below the formal order of accuracy derived for the special case. Generalization of Equation (2) to account for local anisotropy is discussed in Section 3.3.1. Generalization of Equation (2) to account for spatial inhomogeneity is not discussed in this document.

Additional details about the definitions given above, the relations between them, and their importance in scientific computing can be found in [12, 13, 15].

2.2 Order of Accuracy of Discrete Solutions

The *formal order of accuracy* is a property of the discretization and numerical scheme selected in the code [12], and more specifically, it is a property of the truncation error, E_t . It is independent of the particular implementation of the selected scheme. The formal order of accuracy is simply the value of p in generalizations of Equation (2) that account for spacings that are spatially inhomogeneous and locally non-isotropic. A scheme that is formally first-order accurate has a value of p equal to 1, while a scheme that is formally second-order accurate has a value of p equal to 2. For schemes with higher orders of accuracy, the truncation errors decrease at a more rapid rate with decreasing h than for schemes with lower orders of accuracy. It should be noted that for any given mesh, the solution error with a higher-order scheme need not be smaller than it is in a lower-order scheme. A higher order of accuracy only implies a faster rate of decrease of the solution error with decreasing grid spacing, not a lower error for every grid spacing.

Numerical schemes often use mixed orders of accuracy, wherein, for example, some of the terms in the discrete representation are discretized with first-order accuracy while others are discretized with second-order accuracy. Numerical schemes may also use different orders of accuracy for the same term in different parts of the domain. For example, a second-order accurate scheme may be used in parts of the computational domain and a first-order accurate scheme may be used in the remainder of the domain. In such cases, the formal order of accuracy of the overall numerical scheme will be the lowest order of accuracy for any term or for any region of the domain [6]. Following the definition of consistency given in Subsection 2.1, a scheme with an order of accuracy of zero or less anywhere in the computational domain is not a consistent scheme.

Determining the formal order of accuracy for an output quantity from a code requires knowledge of all the formal numerical algorithms and formal discretizations that are invoked in computing the quantity. This is also the case for derived field variables and scalar and vector functionals, which require calculations that are separate from the discretization methods and that have their own independent formal orders of accuracy.

For transient problems, depending on the numerical scheme used, the truncation error may be expressed in the form

$$E_t = C_x(\Delta x^p) + C_t(\Delta t^q) + H.O.T. \quad (3)$$

in which Δx and Δt respectively denote the spatial and temporal spacings, C_x and C_t respectively denote the constant coefficients for the spatial and temporal spacings, p and q respectively denote the formal orders of accuracy in space and time, and all other symbols are as defined above. The

form of Equation (3) allows for different formal orders of accuracy in time and space [12]. As with Equation (2), Equation (3) takes on more general forms for meshes in which the spacings are locally anisotropic or spatially inhomogeneous and when non-uniform time-step sizes are used.

It is emphasized that only in the asymptotic convergence region are the *H. O. T.* in Equations (2) and (3) small compared to the leading terms. For numerical schemes with mixed space-time discretizations, the truncation error may also feature terms of the form $\Delta x^r \Delta t^s$ where r and s take on independent values [12, 15]. As explained in Subsection 2.1, the form of Equation (3) is equally valid for discretizations other than the finite difference discretization, and these include the finite volume, finite element, spectral method, and other discretizations.

The *observed order of accuracy* in space is the actual value of p in Equations (2) or (3) that is achieved by the code, while the observed order of accuracy in time is the actual value of q in Equation (3) that is achieved by the code. Determining the observed order of accuracy in space for a given code requires computing the same time-independent solution with that code on a sequence of grids with systematically increasing refinement, determining how the discretization error in the discrete solution changes with grid refinement, and then determining the value of p [3]. The observed order of accuracy depends on the type of mesh being used. Higher orders of accuracy are typically observed with meshes composed of uniform Cartesian cells than with meshes composed of arbitrary polyhedral cells. Accordingly, different observed orders of accuracy are often quoted for the same code for different mesh types. Determining the observed order of accuracy in space and time requires computing the same solution on a sequence of grids with systematically increasing refinement in space and in time and determining how the discretization error changes with this combined refinement [3]. Refinement in space and time is further discussed in Subsection 3.3.

Invoking Equation (2) for the errors E_1 and E_2 that are obtained for time-independent solutions with respectively two different spacings h_1 and h_2 and neglecting the *H. O. T.* yields the ratio

$$\frac{E_1}{E_2} = \left(\frac{h_1}{h_2}\right)^{p_{12}} \quad (4)$$

The value of the observed order of accuracy p_{12} between the spacings h_1 and h_2 can be derived from Equation (4) as

$$p_{12} = \left(\frac{\log(E_1/E_2)}{\log(h_1/h_2)}\right) \quad (5)$$

The evaluation of Equation (5) can be repeated for different pairs of spacings. As the mesh spacing decreases with each new pair, the value of p_{12} should tend to a constant, signifying the attainment of the *asymptotic convergence* regime. The constant value of p_{12} in the asymptotic regime is the observed order of accuracy. Depending on the particulars of the discretization schemes, test case, and mesh type, the number of different refinement levels of the mesh needed to confidently ensure capturing of the asymptotic regime may vary in practice from as little as three or four to as many as 10 or more.

A graphical approach may also be used to find the observed order of accuracy in the asymptotic convergence regime. Taking the logarithm of both sides of Equation (2) and neglecting the *H. O. T.* yields the following relation for time-independent solutions

$$\log(E) = p(\log(h)) + \log C \quad (6)$$

Equation (6) shows that, on a log-log plot, with $\log(h)$ as the abscissa and $\log(E)$ as the ordinate, the convergence curve of a scheme of order p becomes a straight line of slope p as the grid size h is reduced and the asymptotic convergence regime is attained. Determining the observed order of accuracy, therefore, reduces to determining the slope of Equation (6) on a log-log plot.

Several considerations should be borne in mind when determining the observed order of accuracy. Firstly, an exact solution should be available to determine the error E . Secondly, the numerical scheme should be fixed, that is, the same scheme should be used uniformly throughout the domain and throughout the sequence of grids. Thirdly, the observed order of accuracy refers to the order of accuracy of the numerical scheme as the latter is implemented in the code. However, since a code only outputs a numerical solution, it is necessary to ensure that the numerical solution is as close as possible to the exact solution of the discrete equations. This requires that the “*other error sources*” determining the stability of the scheme are minimized or at least are kept well below the discretization error [3] by converging the residuals to machine precision. Section 3 provides further details on and recommendations for the steps that can be taken to ensure this.

The concepts of formal and observed order of accuracy are crucial to code verification, as further explained and demonstrated in Section 3 and Appendix 1. This is because the main activity in the code verification process is comparing the formal and observed orders of accuracy to draw conclusions about the correctness and appropriateness of the implementation of the numerical scheme in the CFD code.

2.3 Calculating the Error for Code Verification

Code verification should be carried out for both the dependent solution variables and any solution functionals or other quantities of interest (QoI) that are to be predicted using the CFD code. The dependent variables may be primary solution field variables, such as the density, velocity, pressure, and total energy, for example, or derived quantities, such as the pressure gradient, or composite quantities, created by combining two or more variables, such as the strain rate. The error for any variable (whether primary, derived, or composite) is found by calculating the difference in a suitable norm between the computed values of that variable and the corresponding exact values of the model equations. For the finite element method, the numerical solution is defined at every point in the entire domain, and the corresponding discretization error is typically computed using an L_1 , L_2 , or L_∞ norm over the domain. For example, for an L_2 norm of the discretization error in the x -component of velocity, u , the error can be expressed in the form

$$\|E_h\|_2 = \|u_h - \tilde{u}\|_2 = \left(\frac{1}{\Omega} \int |u_h - \tilde{u}|^2 d\Omega \right)^{1/2} \quad (7)$$

where u_h denotes the finite element solution, $\tilde{u}$ denotes the exact solution to the model equations, and Ω denotes the domain volume. For a finite volume discretization, this integral is evaluated discretely as

$$\|E_h\|_2 = \|u_h - \tilde{u}\|_2 = \left(\frac{1}{\Omega} \sum_{i=1}^N (|u_i - \tilde{u}_i|^2 \Omega_i) \right)^{1/2} \quad (8)$$

where N denotes the total number of cells, Ω_i denotes the volume of cell i , and u_i and $\tilde{u}_i$ respectively denote the discrete and the exact volume averages over computational cell i . Similar relations for the discrete L_1 and L_∞ norms for the finite volume discretization can be expressed respectively in the forms

$$\|E_h\|_1 = \|u_h - \tilde{u}\|_1 = \frac{1}{\Omega} \sum_{i=1}^N (|u_i - \tilde{u}_i| \Omega_i) \quad (9)$$

and

$$\|E_h\|_\infty = \|u_h - \tilde{u}\|_\infty = \max (|u_i - \tilde{u}_i|) \quad (10)$$

The L_∞ norm poses a more stringent metric than the L_1 and L_2 norms, and this is because the latter two average the error over all the elements, cells, or points in the domain, and may therefore not readily record incorrect error reduction rates at individual points. Indeed, the form of Equation (2)