RELAP5-3D Verification Capability

George Mesina

Oct, 2013

RELAP5-3D Version 4.1.3 can produce verification files for precise comparison of two code runs.

Sequential Verification

Traditional verification initially checks the software both through comparison of coding against algorithms and equations, and through comparison of calculations against analytical solutions, method of manufactured solutions, and even against actual (separate effects) data. After verifying the first version of the software, subsequent versions must also be verified. Redoing the same verification work for each subsequent version is one method. Another method is sequential verification.

Sequential Verification begins by verifying the original version of the software as explained above. On subsequent versions however, only a representative subset of the original tests are run and, if the calculations are the same, the code is regarded as still verified without reexamining the coding against the documented algorithms and equations. This reduces costs both for codes that undergo constant revision and version release, and for codes that grow in complexity due to continual addition of new models and features.

When a new code version is released and its verification tests compared with the previous version, either a difference is found or not. When one is found, it must be examined to determine if accepted changes elsewhere in the program (particularly maintenance, development, and bug fixes) or operating system software upgrades caused it. If so, the differences must be checked and justified. Thereafter, the new values become the basis for future sequential comparison. If not, the difference is caused by an error that must be discovered and solved.

In concept, sequential verification adequately satisfies the requirements of verification of later versions. In practice, the ability to detect differences has been inadequate.

There are two sources of inadequacy. First, if the test subset is insufficient to detect a bug introduced into verified coding, that coding is still considered verified even with the bug. Second, even if calculations differ, they can go undetected unless examined in sufficient detail. This means that all decimal digits of affected values must be examined. In fact if two versions of the code are run on identical input decks and their output files compared character by character, as has been done for over a decade at INL, differences can (and have) gone undetected in unprinted decimal places.

The remainder of this article describes how the new RELAP5-3D verification capability overcomes the second inadequacy of sequential verification via thorough examination of data for differences.

Verification File

RELAP5-3D writes key data on its new verification file for later comparison.

The verification file records the L_1-norm of the most fundamental quantities in RELAP5-3D, namely, the primary variables of the governing equations for TH, heat transfer and neutronics, along with trips, controls, and time step. In addition some error measures and important condition data are recorded. This information can be collected and written to the verification file on any selected timestep. Such a set of information, along with its timestep information, is called a dump to the verification file. The user can control when, how many, and what kind of dumps are made. A dump is always made on the final advancement unless the run ends with a code abort.

Besides dumps, the verification file also contains data that identifies the RELAP5-3D version number, the data and time the executable was built, and the data, time and machine on which it was run. The title of each input case precedes the dumps of that case.

Qualities of the Verification File

There are three important questions about the adequacy of the verification file.

Why use only these key variables? Why not use many or even all the variables in the code?

The verification file needs to be small so that many such files (representing verification tests of different models and code capabilities) can be stored for later comparison without filling a user's disk. If all variables were written, the verification file size would increase with the size of the input model and would be much larger than a printed output or restart file.

Are these key variables sufficient for detecting differences?

Comparing the verification files of two different code versions for the same input file reveals all differences among these key variables. When any other variable that is not recorded on the verification file is miscalculated, it feeds into the calculation of the primary variables on subsequent time steps. This produces differences in the primary variables that are recorded on the verification file.

Are the L_1-norms sufficiently accurate that differences between calculations of two code runs will always be detected?

To guarantee detection of differences between two runs, these quantities are summed and written with 128-bit floating point accuracy. Since RELAP5-3D stores and uses 64-bit floating point values, the quadruple precision sum captures all bits of all data except for instances of extremely disparate relative sizes. For the primary variables of RELAP5-3D, this is rarely an issue.

Further Uses of the Verification File

Other uses for the verification file include code restart, backup testing, and debugging.

The verification file can be used to test code restart capability. If a test case is run to completion and a second run, starting from an intermediate restart point and running to the same endtime, is compared

to it, the calculations must be the same. The verification file can be produced by both runs can be used to make the comparison.

For RELAP5-3D, another useful form of testing with the verification file is backup testing. A RELAP5-3D backup will occur when the code recognizes that a more accurate calculation can be produced if the system had been formed to take certain changes in the flow field into account. These conditions are water packing (and stretching), appearance of a non-condensable in a control volume, and velocity flip-flop. When one of these occurs, the code backs up to the beginning of the time advancement, rebuilds the system with an adjustment for the condition, solves it and moves forward. In these cases, the time step is not cut.

Backup testing forces the code to take an artificial backup by setting one of the backup condition flags. The user can activate any of the three kinds of backup conditions, test on any set of advancements, or test on all advancements. RELAP5-3D safeguards against backing up on a step with a naturally-occurring backup.

Verification files can also be used for debugging. If a code calculation fails, verification dumps can be activated prior to the failure and rerun. To pinpoint the onset of the error, compare the resulting verification file with one created by a prior code version (without the error) running the same input file. Both the first timestep that the error appears and the primary variables that are affected can be identified.

Activation and Naming

The user must place a “199” card into the input deck to activate the debugging capability. The four words of the card specify that it refers to verification, the kind of verification, the start and end of the verification dumps. The start time can be an integer advancement number or a floating point cumulative time. The end time is always an integer and is, respectively, either an advancement number or a number of advancements. If the end number is less than the start number, only the verification dump for the final timestep will be written.

The name of the verification file can be specified with command-line option “-R.” If it is not specified, it takes the same base file name as the restart file with “vrf” as its file extension. If the restart name is defaulted to “restrt” then the verification file uses its default name of “verify.”

Conclusion

This capability is available beginning in version RELAP5-3D version 4.1.3. Further detail is given in the RELAP5-3D manuals for 4.1.3 and beyond.