Measuring Uncertainty in Scientific Computation Using Numerica 21's Test Harness

Brian T. Smith

Numerica 21 Inc.
Angel Fire, NM, USA
carbess@swcp.com

Abstract. The test harness, TH, is a tool developed by Numerica 21 to facilitate the testing and evaluation of scientific software during the development and maintenance phases of such software. This paper describes how the tool can be used to measure uncertainty in scientific computations. It confirms that the actual behavior of the code when subjected to changes, typically small, in the code input data reflects formal analysis of the problem's sensitivity to its input. Although motivated by studying small changes in the input data, the test harness can measure the impact of any changes, including those that go beyond the formal analysis.

Keywords: testing, scientific application code, floating-point computation, data perturbation, computational sensitivity, test harness tool.

1 Introduction

An article on the website of the National Physical Laboratory on a Framework for Uncertainty in Measurement, June 5th, 2007 [1] makes the following statement:

> "A measurement is meaningless without a quantitative statement of its quality in the form of an uncertainty."

This statement is just as true about a scientific computation as it is about a physical measurement. Software is useless unless the uncertainty in the computed results due to changes in its input or instabilities in the way the results are computed are measured or analyzed. To believe computational results, it is essential to demonstrate that the sensitivity of computed results to changes in input data, precision of computation, and other key data for the software is consistent with what is predicted from the characteristics of the problem being solved. Ideally, a measure of the uncertainty of the computed results as a consequence of the uncertainty of data that the results depend upon needs to be obtained.

In many cases, such a measure of uncertainty is difficult to obtain analytically. However, uncertainty in software can still be measured by running the software with perturbed data values to see whether the computed solution changes as expected. With some thought, a measure can often be devised to indicate how the results change with small perturbations in the input and other key data. This paper is about a tool to help provide an assessment of uncertainty in scientific software and computation.

The tool is a general-purpose test harness modified to facilitate the measurement of changes in results due to changes in input data and other values key to the computation.

Presented in this paper is a brief description of the test harness, its design, its features, and how it was modified to facilitate the measurement of uncertainty of a computation with respect to changes in its input. A case study is provided to show how this tool has been used to measure the uncertainty of the solution of a 3-D magnetostatics computation for the vector potential and magnetic flux or field. The solution technique used for this magnetostatics case study is a boundary element package for 3-D magnetostatics problems from a software firm Accurate Solutions In Applied Physics [2].

2 Motivation – Measuring Uncertainty in Software

Software is at the end of the development chain, depending on mathematical models of physical problems that become the basis for the numerical computation. The solutions to these problems depend on the algorithms used and the data used to drive those algorithms.

Software is the final step. As such, we want to determine if the software is behaving as the mathematical and physical models are predicted to behave. The approach proposed here is to provide a tool that allows one to measure the sensitivity of computed results due to changes in input values or critical parameters in the models, algorithms, and software. Such a measurement of sensitivity indicates how uncertain the computational results are with respect to the uncertainty in the values of such critical data.

3 What Is the Test Harness?

The test harness is a tool to evaluate software. In its initial form, it was a change-detection tool that measured differences in results of two programs that were supposed to create the same results. The applications for such a tool are many: to give a few examples,

- the two programs may be actually the same program compiled by two different sets of compiler flags, such as optimization flags;
- the two programs may be the same but run on different machines;
- one program is an enhancement of one other, enhanced to improve performance but compute the same results, enhanced to use different data structures or organized differently, or enhanced to add some new feature but the developer wants to show that the other features remain unaffected by the enhancements.

The key to the test harness in its original form was to measure "significant" differences in results, that is, differences that represent errors and not differences that can be traced to reordering of operations, changing results of stable computations in minor ways. Therefore, it was essential to have the user provide both the criteria for the comparison (say, relative or absolute difference) and a threshold to indicate whether the difference was predicted and thus acceptable, or unpredicted and thus not acceptable, indicating something was wrong. Also, in scientific applications, arrays and other aggregates need to be compared and criteria for them are needed and have in some cases to be specified by the user.

The test harness is designed to support large scientific codes. As such, these codes involve large collections of data and with such programs, the writing, reading, and comparisons of large volumes of data can be costly. Consequently, the test harness allows the user to select which procedures are monitored, which variables are monitored, which parts of arrays are monitored, how often they are monitored or when the monitoring begins or ends. The test harness measures and reports the volume of data it is monitoring so that the user controls how much is monitored on a given run. It also gives execution counts and execution times for each procedure monitored.

Without going into all the details, the test harness can address all these issues, as described in a previous paper [3] and in its user's guide [4]. Enhancements made to the test harness to support uncertainty measurements are described below. The test harness is currently written for scientific codes in Fortran 77/90/95/2003.

4 The Design of the Test Harness

The test harness is a collection of modules containing input/output procedures to read and write the monitored data, a collection of generic INCLUDE files modified by tools to create application-specific INCLUDE files that include application specific source text into the application code, and a collection of procedures to perform data comparisons and report differences in results. Either the program terminates with the first significant difference or reports its results in a tabular form for an entire program execution. In addition, described in the next section, there are a series of tools that read and analyze the application code, determine default places to monitor results, and build the test harness into the application. The application code with the test harness installed into it can be run in one of two modes described below.

The modes for the application code are: `generate` mode and `check` mode. In `generate` mode, the application code runs to completion, creating data from the run to be compared in `check` mode with another version of the application code. In `check` mode, the data written into files in `generate` mode are read at the point where the corresponding data in the second program is computed and a comparison of the results is performed. In `check` mode, there is the option to terminate the execution at the end of the probe where the first unacceptable result (difference or evaluation that indicates a problem) is encountered or to tabulate the difference and continue execution until the application code completes. Upon completion in the latter case, a summary of the unacceptable results is printed.

Four types of probes or monitoring can be specified; three of the types, namely `input`, `output`, and `specific`, probe record data in a file when the test harness is in `generate` mode and read recorded data and perform comparisons with results recorded in `generate` mode. An `input` probe can be placed at any entry point to a procedure; an `output` probe can be placed just before any exit point from a procedure; and a probe of type `specific` can be placed at any place in the execution part of the application code. The probes are different in what they record; this enables them to make certain checks to ensure only corresponding data is being compared. The fourth type of probe is a `perturb` probe which perturbs specified data values in specified ways when the test harness is in `check` mode, allowing the other probes, if present, to read and compare results between the application code with the original data and results with perturbed data.

5 Building the Test Harness into an Application Code

Much like a debugger, monitoring probes must be placed into the application code. The emphasis though with the test harness is to facilitate the comparison and evaluation of results for floating point (although the test harness supports the monitoring of any intrinsic type or derived type object). Besides addressing the added complication of comparing floating point values, test harness must ensure that the data being compared between the `generate` and `check` modes are comparable values; to do this, it has to trace the execution flow by procedure and order the data so that comparable values (values of the same entities) are compared.

Tools have been created to accomplish these tasks and ensure the integrity of the comparisons. Fig. 1 shows the use of the tools to produce a source code file that represents the application code with the test harness build into it. The `analyzer` tool first analyzes the application code, providing a complete specification of all variables in all procedures in the application and performs a simple usage analysis of each variable to determine if it is referenced for its value before it is written into or is always written before it is referenced. Given this analysis and a list of procedures to monitor, the `installer` tool creates a file readable by the `builder` tool that specifies the input and output probes for each entry and exit point for the listed procedures. Also, given the results of the `analyzer` tool and a list of procedures, the probe tool creates a version of the application code with INCLUDE lines inserted into it. The INCLUDE lines will include source text that will be synthesized by the `builder` tool that represents the test harness built into the application code.

At this point, the user is expected to modify both the files created by the `installer` and `probe` tools. The reason is that these files specify default comparisons and thresholds, probably specify more probe variables than are appropriate for the goal of investigating the code, and may specify more probes than are desirable or appropriate (the reference/definition analysis is only approximate and in general includes variables that need not be monitored). In the case

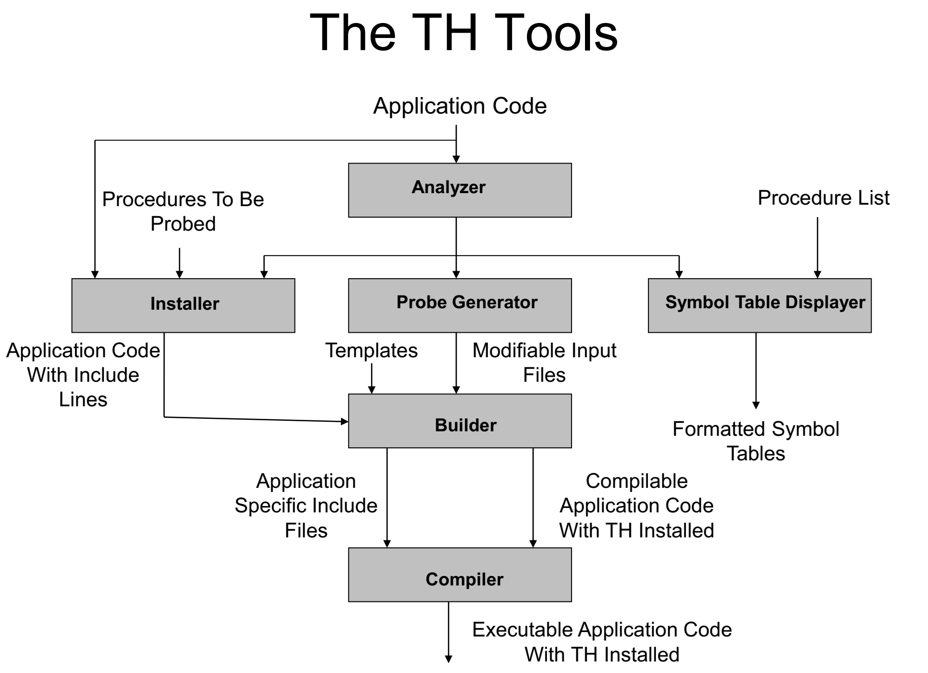

Fig. 1. The test harness tools showing their input and output connections to install the test harness into an application code

of the files created by the `probe` and `installer` tools, these can be modified and reused in subsequent runs without rerunning these tools. In the case of the `installer` file, the only expected modification is the deletion of INCLUDE lines that represent unwanted probes. One of the major modifications to the `builder` monitoring specification files (created by the `probe` tool) is also the deletion of inappropriate monitored variables; changing the default monitoring thresholds to appropriate thresholds for the computation is unavoidable until further tools are provided.

Once the builder input files are modified, the `builder` and `includer` tools with the modified files complete the installation of the test harness into the application code. The monitoring process proceeds by running the application code with the test harness installed and often involves revisiting the choice of thresholds and monitoring. The typical situation is that the `analyzer` and `probe` tools are not rerun while investigating the behavior of the code. The user can change what and how variables are monitored, even what procedures are monitored, without rerunning the `analyzer` and `probe` tools. If variables that are monitored are changed or their comparison criteria or thresholds are changed, the test harness must be rerun in `generate` mode (including measurements of uncertainty or code sensitivity to data).

Fig. 2 shows a typical scenario with the use of the test harness. The top line represents versions of the code that are run in `generate` mode. In this

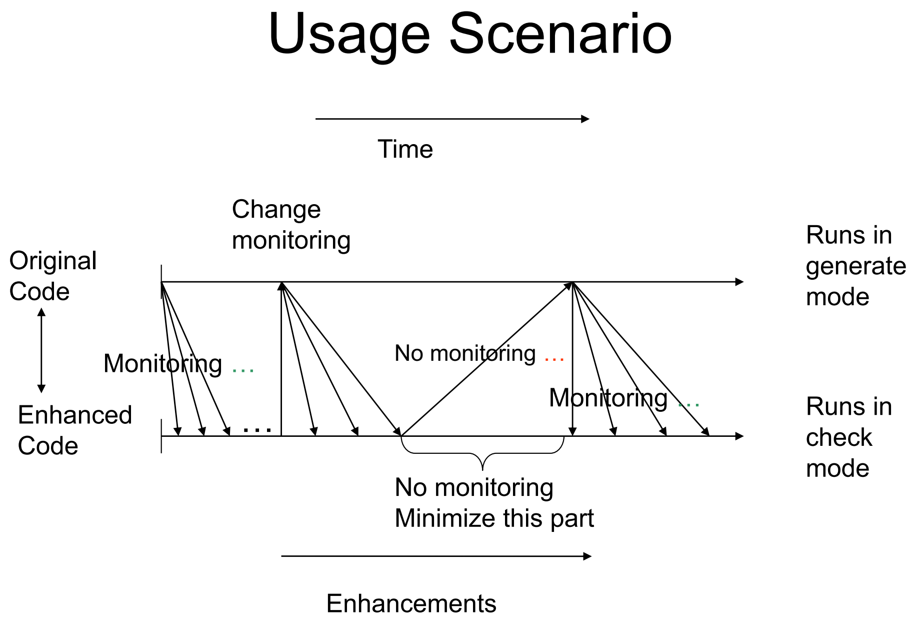

Fig. 2. A scenario for using the test harness to support code development. The top line represents the original version of the code compared with the modified version on the bottom line. The code is enhanced as time progresses to the right and the diagonal lines down represent comparisons made between the original and enhanced versions of the monitored variables. The line with the up arrow indicates the modified version becomes the production version at certain points and the monitoring is changed.

mode, values of variables in procedures specified in the `builder` file are recorded for later comparison. Many runs of the test harness in `check` mode (including perturbing variables) can be run and compared with the monitoring data created in the version run in generate mode.

The bottom line of Fig. 2 represents versions of the application with the test harness built in the application run in `check` mode. Each of these runs may have different criterion for comparison or different perturbations specified in the builder input files but require the `builder` and `installer` tools to be rerun to generate a version to run these different cases. Time progresses towards the right. The slanted lines down from the same "generate" version indicate different runs with typically different builder input files. Also, the application source files with the probes inserted may also be changed as long as the variables that are monitored or evaluated or the order in which they are generated are not changed. At some point, it is desirable to use the code as it has been changed or monitor different items. That process is indicated by the lines with the up arrows in Fig. 2 where the version from the lower line is run is `generate` mode to create anew the monitored data file. The process of monitoring and changing codes continues anew, until the user is satisfied with the results.

The diagonal line with an up arrow indicates a jump in versions where the code that is being monitored is changed in some substantial way and no testing of the changes is made. Hopefully, these kinds of changes do not often occur because they represent situations where code is enhanced and tests are not performed to ensure that the existing code still performs the way it used to before the changes.

For uncertainty measurements, the process is much simpler, essentially depicted only by the downward arrows and the code is never changed. However, the builder input files may be changed to measure the sensitivity of the computed results caused by changes in different variables or combinations of variables. Using the test harness for uncertainty measurements is described in more detail in Sect. 7.

6 Brief Summary of the Test Harness Features

When the execution of the application code with the test harness installed in it is executed, the test harness reads files which specify several options:

- The mode, either `generate` or `check`;
- Whether execution performance for each monitored procedure is to be measured;
- Whether the sizes of the files containing the monitor data for each procedure are recorded and printed at the completion of the application code, when in `generate` mode;
- Whether the application code terminates or continues on the first occurrence of a difference that violates the comparison threshold;
- Whether a summary report of all comparisons performed during the run upon completion of the application code is printed;
- The Fortran logical units for diagnostics and for debugging output;
- The Fortran logical unit for the monitored data;
- The maximum number of monitored procedures;
- The name of the main program;
- The maximum number of routines to be monitored; if it is not provided, the default is 100;
- The default value for the tabulate option. If it is not present, the default value is no tabulation.

The *builder* input files allow the following attributes to be provided; if not provided, a default value is set in all cases:

- The lower bound for array subscripts, when such lower bounds cannot be determined from the source code, for example, when the array is assumed-size. The default is a vector of 1's of the rank of the variable;
- The relative tolerance or threshold used to compare floating point data values. The default is zero, which implies the compared values must be identical. For types other than floating point, the relative tolerance test is not made;

- The norm type, either element or global; the default is element. Element means the comparisons for an array are element by element and for the relative threshold, relative to each element. Global means the relative comparisons are element by element but relative to the norm of the array;
- The absolute tolerance or threshold used to compare arithmetic data values. The default is zero, requiring the compared values to be identical;
- The scope of the comparison, either global for all variables of this name, or local to this procedure. The default is local;
- The section of the array to be compared. The default is the whole array. However, for assumed-size arrays, a section which specifies the final subscript value or range is required because the `analyzer` cannot determine this value;
- Whether to tabulate the comparison results and print the tables at the end of the execution of the application code when in `check` mode. The default is the `tabulate` option specified in the test harness input file;
- The specification of "how" the perturbation is to be performed. The options are a relative or absolute perturbation or by a specific value of the specified variable with the random perturbation at most the size of a specified value, selected from a uniform distribution. There is no default value; the option must be specified;
- The name of a procedure that is to perturb the variable. If provided, the procedure overrides all other specifications of how the variable is to be perturbed. The default is no procedure specified;
- The name of a procedure that is to perform the comparison of data values. The default is no procedure specified.

The output generated when "tabulate=yes" is specified is printed on standard output after the application completes execution. It is a large table with a collection of lines for each probe for each procedure that is monitored. For each procedure, there is a line for each variable. The information printed in the table is:

- The name of the procedure;
- The kind of probe (1 for input probe, 2 for output probe, 3 for a specific probe, and 4 for a perturb probe);
- The probe name;
- The variable name;
- The flag E or N; E indicates the threshold was exceeded; N indicates the threshold was never exceeded;
- The type of comparison, when an array; elemental or global;
- Two sub-tables, one for absolute comparisons and one for relative comparisons. In each sub-table, 2 or 3 columns are provided for the first, maximum, and average differences, indicating the value of the difference. Also provided is the procedure call count for the reported difference and with the linear position in the array, if an array, where the difference occurred.

7 How Uncertainty Evaluation of Software Is Performed with the Test Harness

First, the `builder` input file is modified to specify the variables to be monitored and perturbed. This includes how they are to be perturbed and how the results are to be compared. Then the application code with the test harness in `generate` mode is executed to create a collection of monitored data. Next, the application code is rerun with the test harness in `check mode`, tabulating the results. Just before the test harness closes in the `check` mode run, it prints the results, indicating how the perturbation changed the results.

8 A Case Study – Measuring Uncertainty by Perturbing Data

The case study to demonstrate how to use the test harness to study uncertainty of computed results due to changes in input is a package of double precision codes to solve 3-D magnetostatics problems for the vector potential and magnetic flux using the boundary element method developed by ASAP LLC [2]. The equations solved are the 3-D Laplace equations with boundary conditions specified over the surfaces of 3-D objects. The test problems use spheres, annular cylinders, cubes, and tori.

The boundary element methods solve the Laplace equation by integrating Green's functions over boundary elements to produce a relatively large linear system of equations. The size of the system is dependent on the number of boundary elements. The integrands are singular in many cases and are transformed in several ways to remove the singularities, but unless care is taken, the matrix of the resulting linear system of equations may be near singular; how singular depends on the shape of the object and the aspect ratio of the elements as well as the techniques used to avoid the near singular integrands.

The goal of the study is to measure the uncertainty of the computed boundary solutions for these test objects and to show that the uncertainty analysis could be extended to objects for which the solutions are not known. The perturbations of interest were to the boundary conditions and to the weights and points of the quadrature formulas used to perform the needed surface integrals. The applications for this software are in cases where the boundary conditions are likely known to a few digits (usually 3 digits), but we were interested to find some quantity in the computation that might indicate or measure the sensitivity of the solution to the boundary conditions that could be computed when the solution was not known.

The package of software is approximately 50K lines and over 300 procedures. The test harness has been installed in most of the computational components for the regular testing of the package but for this study, the test harness was used only in the solver routine and the procedures involved in the solution, representing approximately 15K lines and 100 procedures, of which only 24 procedures and 168 variables were monitored.

8.1 Measurements

For this study, only the results for the sphere, annular cylinder, and torus are reported here. In all cases the solution vector, consisting of either the Cartesian coordinates of the vector potential or the tangential components of magnetic flux at surface nodes, was examined to study its dependence on boundary data. Solution variation was measured using the maximum element norm of the difference between the vector solution computed with the unperturbed and perturbed data. The vectorial boundary condition data were perturbed by addition of uniform random variables to their Cartesian coordinates. The magnitude of the perturbations was scaled relative to coordinate value with a change up to 100 units in the last place of double precision, 1 unit in the last place in single precision, 10,000 units in the last place in single precision (roughly a change in the third digit), and 100,000 units (roughly a change in the second digit). The boundary conditions are different for each problem; for the sphere and torus, the boundary conditions were Dirichlet and for the annular cylinder, the boundary conditions were mixed Dirichlet and Neumann. The element shapes were quadrilaterals for the torus and annular cylinder and mixed quadrilaterals and triangles for the sphere.

Also, measured as part of the case study were perturbations in the Gaussian weights and points. For all formulas (formulas with more weights and points are used when the integrand is determined to be near singular), the weights and points were changed by random perturbations relative to themselves at the same levels of 100 units in the last place of double precision and 1, 10,000, and 100,000 units in the last place of single precision. Perturbations to both the boundary conditions and Gaussian quadrature parameters at the same time were not performed in the material for this demonstration although this is possible with the test harness tool.

As a general practice, the Laplace solvers estimate the condition number of the linear system [5]. The expectation was that the size of the perturbations of the computed results would depend on the condition number, larger for larger condition numbers of the linear system. The concern was that other commodities might contribute, like how close to singularity were the integrands or how often the higher order quadrature rules or the Telles transformations [6,7] were used to handle very singular integrands.

8.2 The Results

Fig. 3 to Fig. 7 plot the sizes of the perturbation of the solution with respect to the perturbations of the boundary conditions and the quadrature weights and points, tested separately. The plots show that the effect of perturbations of either of these quantities on the solution is relatively small in general, roughly of the size of the perturbation but roughly proportional to and dependent on the condition number. That is, for Fig. 3 (the sphere), the perturbations in the solution follow closely the perturbations in the "input"; for the sphere, the condition number of the linear system is approximately 250. Similarly, in Fig. 5

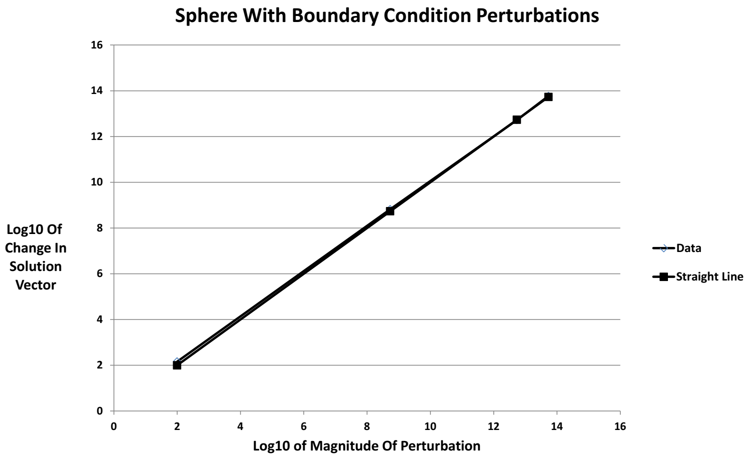

Fig. 3. For a sphere, measuring the perturbations of the solution where the boundary conditions are perturbed by a relative amount of approximately 10^2, 10^9, 10^{13}, and 10^{14} units in the last place of double precision. The solution is perturbed only slightly more than the perturbation in the boundary conditions (that is, the lines are on top of one another). The condition number of the linear system is approximately 250.

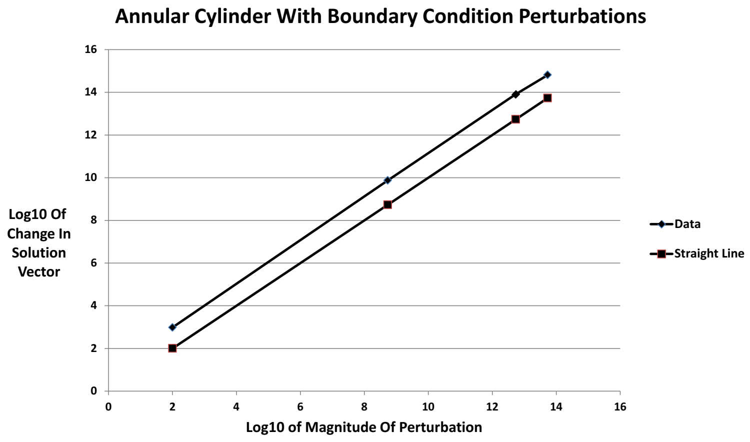

Fig. 4. For an annular cylinder, measuring the perturbation of the solution where the boundary conditions are perturbed by a relative amount of approximately 10^2, 10^9, 10^{13}, and 10^{14} units in the last place of double precision. The solution is perturbed more than the perturbation in the boundary conditions. The condition number of the linear system is approximately 6000.

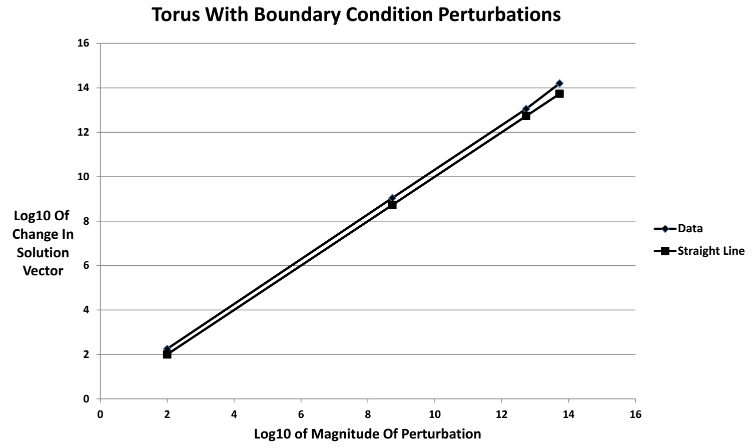

Fig. 5. For a torus, measuring the perturbations of the solution where the boundary conditions are perturbed by a relative amount of approximately 10^2, 10^9, 10^{13}, and 10^{14} units in the last place of double precision. The solution is perturbed more than the perturbation in the boundary conditions. The condition number of the linear system is approximately 170.

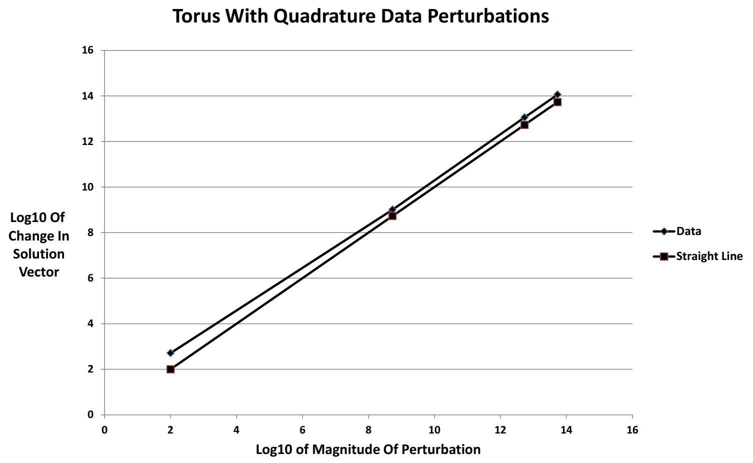

Fig. 6. For a torus, measuring the perturbations of the solution where the Gaussian quadrature weights and points are perturbed by a relative amount of approximately 10^2, 10^9, 10^{13}, and 10^{14} units in the last place of double precision. The solution is perturbed more than the perturbation of the Gaussian parameters and is more sensitive to weights/points perturbations than boundary condition perturbations. The condition number of the linear system is approximately 170.

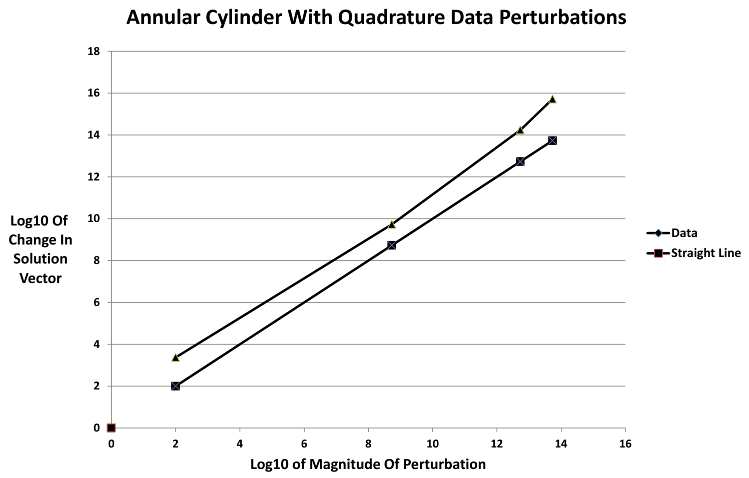

Fig. 7. For an annular cylinder, measuring the perturbations of the solution where the Gaussian quadrature weights and points are perturbed by a relative amount of approximately 10^2, 10^9, 10^{13}, and 10^{14} units in the last place of double precision. The solution is perturbed more than the perturbation of the Gaussian parameters and is more sensitive to weights/points perturbations than boundary condition perturbations. The condition number of the linear system is approximately 6000.

(the torus), the perturbations follow the perturbations in the input; the condition number approximately 170. However, in Figure 4 (the annular cylinder), the perturbations in the solution magnify those of the boundary conditions but still follow the perturbations in the boundary condition; the condition number of the linear system for the annular cylinder is approximately 6000, 25 to 40 times larger than that for the sphere and torus, but the perturbation in the solution is approximately 10 times larger, slightly smaller for the small perturbations and the factor increasing to slightly more than 10 times for the larger perturbations. The line labeled "Straight Line" (using square points) plots the perturbation in the solution as if the perturbation of the input data created the same perturbation in the solution. The line labeled "Data" (using diamond points) plots the measured perturbation in the solution.

Fig. 6 and Fig. 7 show similar behavior as a consequence of perturbations in the quadrature parameters for the torus and cylinder, with the magnification of the perturbations in the solutions being smaller for the torus where the condition number is smaller than that for the annular cylinder by a factor of approximately 40.

9 Conclusions

For this case study, it was relatively straightforward to make perturbations in the data and to measure the changes in the computed solution. The results of these measurements are consistent with the conjecture that the condition number of the linear system will indicate the sensitivity of the solution to changes in the input boundary conditions. Further experiments not reported here continue to confirm the significance of the condition number of the linear system when other boundary conditions are selected.

The test harness has provided a convenient tool to measure uncertainty due to data changes. It is conjectured that by changing the application code so that a version run in `generate` mode uses a slightly different model than the application code run in `check` mode would allow measurements to be made of the uncertainty in the solution caused by using a different model. The only requirement is that the perturbations in the model can be computed by specifying a user-supplied procedure to make the perturbations and a second user-supplied procedure can be written to measure the effect of the perturbations on the computed results.

Acknowledgements. The design of the test harness was supported in part by a DOE Phase 1 SBIR award DE-FG-02-04ER84028, July 14, 2004 – April 12, 2006.

References

1. National Physical Laboratory, UK: A Framework for Uncertainty in Measurement (2010), `http://www.npl.co.uk/mathematics-scientific-computing/mathematics-and-modelling-for-metrology/measurement-uncertainty-framework/a-framework-for-uncertainty-in-measurement` (accessed November 17, 2011)
2. Accurate Solutions In Applied Physics LLC, Albuquerque, New Mexico (2011), `http://www.manta.com/c/mtvfjt3/accurate-solutions-in-applied-physics-llc` (accessed November 17, 2011)
3. Smith, B.T.: A Test Harness TH For Numerical Applications and Libraries. In: Gaffney, P.W., Pool, J.C.T. (eds.) Grid-Based Problem Solving Environments. IFIP, vol. 239, pp. 227–241. Springer, Boston (2007)
4. Smith, B.T.: The Test Harness User's Guide, Version 0.6.9, Numerica 21 Incorporated (2010)
5. Anderson, E., et al.: LAPACK User's Guide, 3rd edn. SIAM, Philadelphia (1999), `http://www.netlib.org/lapack/lug`
6. Telles, J.C.F., Oliveira, R.F.: Third Degree Polynomial Transformation for Boundary Element Integrals: Further improvements. Eng. Anal. BEM 13, 135–141 (1994)
7. Baltz, B., Mammoli, A.A., Ingber, M.S.: Incremental Improvements to the Telles Third Degree Polynomial Transformation for the Evaluation of Nearly Singular Boundary Integrals. In: Chen, C.S., Brebbia, C.A., Pepper, D.W. (eds.) Boundary Element Technology XII, pp. 459–473. WIT Press, Southampton (1999)

Discussion

Speaker: Brian Smith

William Kahan: What if the tool's INCLUDEs insert statements that should not, but do, change the arithmetic because of over-agressive compiler optimizations triggered or inhibited by the INCLUDEs?

Brian Smith: The short answer is that this situation of the included text disturbing the optimization is very unfortunate but certainly possible. I have tried to minimize the likelihood of this happening for the default probe insertion locations where I have some expectation that the included text will not disturb the optimization. The included text is, in most cases, a CALL statement and typically the impact of a CALL statement on compiler optimization is predictable at an entry point or exit point of a procedure. However, when the user inserts a probe to monitor or change program variable values, the user must be aware of what impact the probe is having on optimization; the documentation in the users guide warns the user of this issue. For example, placing a probe in the middle of a loop will likely change the optimization and the user needs to be aware of this impact and how it affects the computed results.

John Reid: It looks as if you are working in Fortran 95 and do not support nested procedures. Is this true?

Brian Smith: No. The test harness supports nested procedures as restricted by Fortran 90/95/2003; that is, these versions of Fortran prohibit internal procedures nested in internal procedures so that the installation of a monitoring, evaluation, or uncertainty probe in an internal procedure must not create an internal procedure within an internal procedure. This is mainly an annoyance and inconvenience in that INCLUDE lines representing a probe are replaced by a CALL statement to an internal procedure that is created and inserted at the end of the procedure unit in all procedures except an internal procedure. For an internal procedure, the INCLUDE line is replaced by many lines of code that implements the probe.

If a version of Fortran removes this restriction, then the test harness code installer will be modestly modified to replace the INCLUDE line for a probe with a CALL statement to an internal procedure in all cases in the same way it treats all other procedures.

So the bottom line is the test harness currently supports nested procedures in all ways allowed by the Fortran 90/95/2003 standards.