THE IMPLEMENTATION OF AN AUTOMATED DIRECT SENSITIVITY ANALYSIS TOOL TO THERMALHYDRAULIC CODES

A. C. Morreale
McMaster University, Hamilton, Ontario, Canada

Abstract

The move towards probabilistic strategies and reliance on best estimate codes for thermalhydraulic (TH) analysis increases the importance of proper sensitivity and uncertainty analysis. Sensitivities can be determined using a direct analytical approach through the use of partial derivatives. Despite the high level of upfront effort required, this method, once accomplished, is applicable to almost any scenario modeled. The sensitivity system can be used separately to generate information without requiring full TH code runs. The design, methodology and application of an automated FORTRAN script that can edit existing FORTRAN coded TH simulation source to include sensitivity calculations are described herein.

1. Introduction

The use of Best Estimate and probabilistic methods is increasing in Nuclear Engineering and are starting to overtake the former conservative analysis methods as the tools of choice in safety analysis. These emerging methods, which strive for a more accurate representation of the real system, require the incorporation of detailed uncertainty quantification and sensitivity analysis. A proper evaluation of the sensitivities provides information on the inner workings of the system that can be utilized to understand the propagation of input uncertainties through the code. In addition, detailed sensitivity analysis provides ranking information to determine the most influential input parameters on the specific output responses.

Primarily, sensitivity analysis is performed using perturbation methods which treat the code as a closed system and measure the change in outputs when an input parameter is perturbed. This type of analysis is computationally intensive as multiple full code runs must be made for each case and must be repeated for each scenario modeled. In addition, this method avoids any investigation of the code structure itself only comparing the inputs and outputs.

Sensitivity analysis can also be performed using a direct method which explores the code structure and embeds sensitivity calculations within the code so the sensitivity case can be computed simultaneously. The equations and numerical methodology of the model are evaluated analytically in order to produce the sensitivity of the model to variations of the input parameters. This method relies on producing the partial derivatives of the model equations for each parameter that can affect the outcome. This method of parallel analysis does require detailed knowledge of the simulation program including all model equations and correlations used down to their base levels. Despite this, once the partial derivations are determined and integrated into the code, the derivatives for each input about any reference value can be computed. This database of partial derivatives can then be utilized separately to perform

sensitivity calculations without running the full code. The advantage is that the analytical process of computing partial derivatives need only be done once to assess the relations within the code for any possible transient. In addition, the determination of these partial derivatives provides valuable insight as to which parameters are most important in their affects on the code output. This information will reduce the experience needed and the engineering judgement requirements necessary when establishing which parameters are most important for a specific scenario. By taking the partial derivatives in parallel, a complete set of sensitivities for the full transient can be recorded.

The sensitivity and resulting uncertainty analysis can maintain either a local or global focus. Local analysis examines the interrelations in parameters at a specific local point within the simulation. Therefore, multiple local analyses can be performed to track the sensitivities throughout the simulation both temporally and spatially. Globally focused sensitivity seeks to determine the critical points of the system such as maxima, minima, bifurcations or saddle points.

If the direct sensitivity analysis procedures are developed in parallel with the construction of the modeling code, the knowledge of the modeling strategies is readily available and the process of integrating sensitivity computations into the modelling code requires minor additional efforts. However, if the modeling code has already been fully developed and there is a desire to modify it to insert direct sensitivity analysis extensive and detailed efforts are required. The modeling code must be fully investigated and dissected to develop an exact understanding as to how the simulation actually functions. The model equations must be explored and traced from high level equations right down to the dependence on the base variables of the system and the input parameters provided by the user. Once the code has been broken down the partial derivatives can be computed from the base level and carried back up the line to the top. When dealing with complex simulation codes such as those used in nuclear thermalhydraulic analyses this process requires considerable time an experience. This is due to the many interacting model equations and extensive lists of parameters utilized within the code.

2. The design of an automated direct analytical sensitivity system

A standard thermal hydraulics code package makes use of multiple interconnected subroutines to calculate the state variables that are simulated by the code. The state variable responses of the code are affected by the parameters present within the calculation. For example, the state variable of tank pressure in a simple gas blowdown experiment is dependent upon a large number of parameters such as break opening, tank dimensions, external environment conditions, flow characteristics and various properties of the subject gas. The calculation of pressure is made using components such as mass flow rate, heat transfer coefficients, volume and other intermediary factors that are computed from the system parameters. TH code packages utilize multi-level calculations where each of the constructs is calculated in a separate subroutine leading up to the main level calculation for the state variable. An example of this multi-level system is seen in Figure 1.

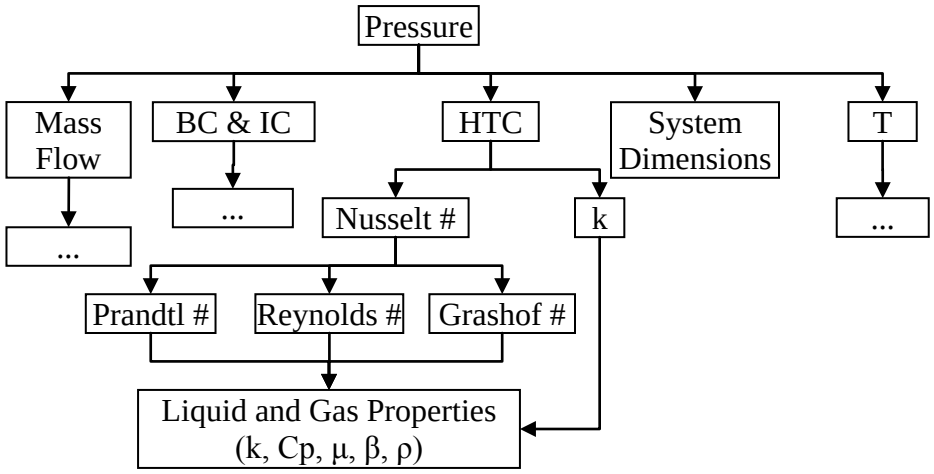

Figure 1: Example Calculation Flowchart for Simple Thermalhydraulic System

The multi-level nature is evident from the illustration in Figure 1. In order to produce a sensitivity analysis based on the partial derivatives all the parameters must be related to the response variable. Since each block of the multi-level system is calculated in a separate subroutine (general practice in TH codes) there are two avenues of approach. First the derivatives of the response for each subroutine can be calculated and then the derivative of the main response for each of the individual parameters can be calculated by way of the chain rule as shown in Equation 1.

$$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial g} \bullet \frac{\partial g}{\partial a} \qquad (1)$$

Second the individual parameters of the lower level subroutines can be upgraded sequentially to the main subroutine adding them to the subroutine arguments hence allowing a computation of the derivative $\frac{\partial f}{\partial a}$ directly. The derivatives themselves can be determined by way of an analytical formula, either coded manually or determined by an external script or by the use of a numerical finite difference derivative which measures the change in output based on the change in input. A staged analytical system is best used with the first method computing the derivatives stage by stage and producing the top level state variable derivative using the chain rule. The parameter upgrade method is well suited to an automated numerical approach since the parameters in questions are input arguments to the multi-level subroutine calculation and the state variable responses are outputs. Therefore, a numerical finite difference derivative can be easily determined using Equation 2.

$$\frac{\partial f}{\partial \alpha_i} = \frac{f(\alpha_i + \varepsilon\alpha_i) - f(\alpha_i)}{\varepsilon\alpha_i} \qquad (2)$$

The factor ε is very small and is chosen to produce an accurate and stable derivative solution in general a suitable value is approximately 1000 times the modelling code accuracy. Either method, the chain rule based analytical or the parameter upgrade numerical derivative can be

utilized to produce the necessary partial derivatives that will be required by the sensitivity space of the particular system.

Manual coding using analytical partial derivatives has been performed on simple TH code packages. An example, is the application by Petruzzi of direct analytical sensitivity analysis through partial derivatives to a simple nitrogen gas blow down system [1]. The system consisted of a tank of pressurized nitrogen depressurized into a stagnant air environment at atmospheric pressure. The main properties investigated were the nitrogen pressure transient, the nitrogen temperature and the heat transfer through the tank wall during the blow down. The blow down system equations were explored and converted to discrete form in order to apply a numerical solution code to the problem. The discretized equations were then broken down to the state variables of the system. Once the base level was reached the partial derivatives were taken back up the line until the main model equations were reached.

Automated codes that read in source subroutines and produce derivatives are in existence such as the ADIFOR code which given a single subroutine and its inputs and outputs will produce the associated analytical partial derivatives [2]. The code would be applied to each subroutine and then the chain rule used to relate the specific parameters to the state variables at the top level of the calculation.

Using the parameter upgrade and numerical derivative approach an automated code can be created that reads in the full text (or selected portions) of the source code and upgrades the desired parameters (as specified by the user) until the main calculation level is reached. The upgraded parameters are added to the arguments of the subroutines and their definitions are passed up the line. Once the upgrade is complete, the upper subroutine that outputs the state variable has inputs that contain all the pertinent parameters that are needed for the sensitivity space. The code can then call this subroutine multiple times to calculate the numerical finite difference derivatives for each parameter.

The automated code using parameter upgrading and numerical finite difference derivatives was created using FORTRAN for use on other FORTRAN coded scripts due to this code's wide spread use in the area of scientific modeling and especially nuclear engineering. The full code package consists of a pre-reading parameter upgrade program that is run multiple times and is followed by a derivative subroutine generator that produces the derivative subroutines. Once the partial derivatives are produced they can be compiled into a sensitivity space for the system.

The sensitivity system is created based on the DDDSM method designed by Petruzzi, [1], and the matrix calculation method is detailed for a two step semi-implicit system in Equations 3 and 4 with $\mathbf{s}$ being the set of state variables and $\boldsymbol{\alpha}$ being the set of parameters. A further code that is run after the parameter upgrade and derivative generation codes is used to convert the partial derivative values into the associated sensitivity space as per Equations 3 and 4.

$$\mathbf{J}_{\boldsymbol{\alpha}}^{n+1} = \left.\frac{\partial \mathbf{s}^{n+1}}{\partial \boldsymbol{\alpha}}\right|_{\mathbf{s}_0} = \frac{\left(\mathbf{I} + \left.\frac{\partial \boldsymbol{\Phi}}{\partial \mathbf{s}^n}\right|_{\mathbf{s}^{n+1}(\mathbf{s}_0,\boldsymbol{\alpha}),\boldsymbol{\alpha}} (\mathbf{s}^{n+1},\mathbf{s}^n,\boldsymbol{\alpha})\right) \cdot \mathbf{J}_{\boldsymbol{\alpha}}^n + \left.\frac{\partial \boldsymbol{\Phi}}{\partial \boldsymbol{\alpha}}\right|_{\mathbf{s}^{n+1}(\mathbf{s}_0,\boldsymbol{\alpha}),\mathbf{s}^n(\mathbf{s}_0,\boldsymbol{\alpha})} (\mathbf{s}^{n+1},\mathbf{s}^n,\boldsymbol{\alpha})}{\left(\mathbf{I} - \left.\frac{\partial \boldsymbol{\Phi}}{\partial \mathbf{s}^{n+1}}\right|_{\mathbf{s}^n(\mathbf{s}_0,\boldsymbol{\alpha}),\boldsymbol{\alpha}} (\mathbf{s}^{n+1},\mathbf{s}^n,\boldsymbol{\alpha})\right)} \qquad (3)[1]$$

$$\mathbf{J}_{\mathbf{s}_0}^{n+1} = \left.\frac{\partial \mathbf{s}^{n+1}}{\partial \mathbf{s}_0}\right|_{\boldsymbol{\alpha}} = \frac{\left(\mathbf{I} + \left.\frac{\partial \boldsymbol{\Phi}}{\partial \mathbf{s}^n}\right|_{\mathbf{s}^{n+1}(\mathbf{s}_0,\boldsymbol{\alpha}),\boldsymbol{\alpha}} (\mathbf{s}^{n+1},\mathbf{s}^n,\boldsymbol{\alpha})\right) \cdot \mathbf{J}_{\mathbf{s}_0}^n}{\left(\mathbf{I} - \left.\frac{\partial \boldsymbol{\Phi}}{\partial \mathbf{s}^{n+1}}\right|_{\mathbf{s}^n(\mathbf{s}_0,\boldsymbol{\alpha}),\boldsymbol{\alpha}} (\mathbf{s}^{n+1},\mathbf{s}^n,\boldsymbol{\alpha})\right)} \qquad (4)[1]$$

This code identifies the state variables within the calculation and using the two time step form seen in Equation 5 produces the matrices of the partial derivatives based on the Φ functions.

$$P^{n+1} = P^n + \Phi \qquad (5)$$

By utilizing these three scripts in sequence and applying them to a FORTRAN coded TH simulation the user can produce a new simulation code that has imbedded sensitivity calculations than can be computed concurrently during the simulation execution and provide sensitivity values at each time step. The following sections describe the AutoSense code package and its associated components, SUPRDR and SENSOR.

3. Description of the SUPRDR and SENSOR codes

The AutoSense FORTRAN scripts used to generate new versions of FORTRAN coded TH models that include sensitivities are three fold. The first script, SUPRDR contains both the parameter upgrading module, PRERDR, and the derivative generation module, RDR. The second script is SENSOR, which contains the sensitivity conversion module. The scripts are written in FORTRAN-95 and are designed to read and interpret script written in FORTRAN-77 or FORTRAN-95. The codes and their pertinent subroutines are described herein.

3.1 SUPRDR CODE

The SUPRDR code is made up of two parts PRERDR and RDR which respectively upgrade any selected parameters to the top level and produce the derivative generation subroutines for each calculation subroutine. The parameters and responses included in the upgrading and derivative work are restricted to floating point variables and the partial derivatives are computed using a scaled finite difference method as per Equation 2. A flowchart of the SUPRDR code is shown in Figure 2.

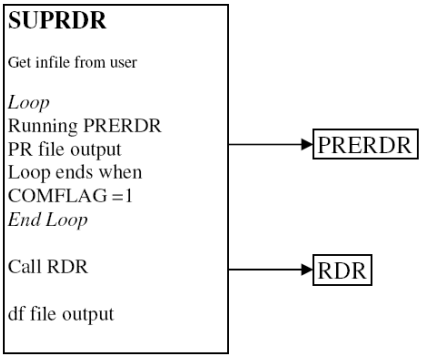

Figure 2: Flowchart for the SUPRDR Code

The parameter upgrade code, PRERDR, is designed to take locally defined variables from a low level subroutine and upgrade them to the top of the calculation. This allows the selected parameter to be directly related to the state variable output response being calculated. The upgrading is done iteratively one subroutine level up at a time and the process is complete once the parameter has reached the main program level. The PRERDR module is made up of the main program which is used to modify and assemble the edited code and PRERDMAN which is the subroutine that reads the individual source subroutine codes, determines the parameters to be upgraded, copies the necessary declaration lines and assignment statements to be inserted in the next level up and compiles the additional variables to be added to the subroutine arguments. In the process, PRERDMAN calls common subroutines RESEARCH, ARAY and AOUT, which are described later. The iterative process runs PRERDR multiple times and at each stage produces a new source file. Once the upgrade cycle is complete the COMPFLAG variable is set to 1 meaning that there are no more parameters left to be upgraded. The parameters to be upgraded are identified automatically as any real variable that has an assign (‘a = x’) statement that is not already an output of the subroutine. The user can avoid upgrading a certain parameter if desired via a comment in the input source code. A flowchart of the PRERDR module is provided in Figure 3.

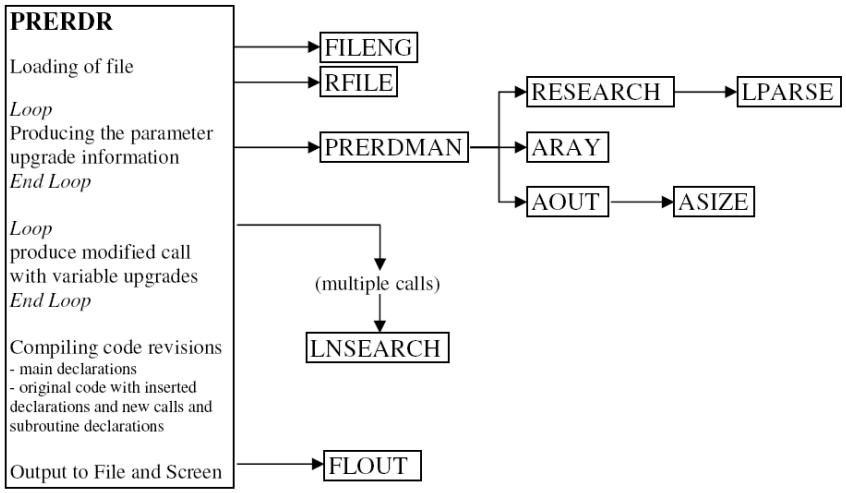

Figure 3: Flowchart of the PRERDR module

Once the PRERDR module has completed its cycle and COMFLAG = 1, the RDR module is executed on the most recently generated file from PRERDR. The RDR subroutine is responsible for collating and reassembling the input source to integrate the newly created derivative generating subroutines. The subroutine RDMAN is called from RDR and constructs the derivative computation subroutines. RDMAN calls the common subroutines AOUT, ARAY and RESEARCH.

The derivative subroutines are created by first determining all the inputs and outputs of the original subroutine and then computing numerical partial derivatives for all floating point values (as per Equation 2). The inputs and outputs are determined by creating a list of the arguments and common block variables (using the RESEARCH subroutine) and then comparing them to a list of all the real variables generated from RESEARCH the real variables that are contained within the arguments and common blocks are considered the "variables under question" as they are potential inputs or outputs.

The outputs are determined using the AOUT subroutine which searches for assignment statements in the code and compares the assigned variables to the variables under question. Therefore, if a variable is changed within the subroutine it is considered an output. All variables under question, including the outputs themselves are considered inputs.

The partial derivatives are produced by calling the original subroutine with a modified input value and comparing the change in the output value. The partial derivatives are created for all inputs with respect to all the inputs. The derivative values are collected into an array which holds the values as the computation takes place and outputs to the screen or a file each time the derivative subroutine is called. The newly created derivative subroutine has a unique structure which includes the loading of all original variable values into control arrays, perturbing each single input for a given output, measuring the change and then resetting the input back to its original value. The derivatives are stored in an array which is added to the arguments and is available to the calling subroutine. At the end of the derivative subroutine the derivatives are outputted to a file or the screen and the original subroutine is called one last time with all the inputs at their control value to ensure the normal simulation calculation is not affected.

Once all the derivative subroutines are created the RDR module assembles the code back together and places calls to the derivative subroutines directly after any call to the associated original subroutine. Since multi-layered subroutines that are called from these derivative subroutines may also have derivative subroutines of their own, these call statements are placed within an "if" condition, which is tied to a flag variable. If the call to a subroutine is made within a derivative subroutine the flag is set to zero and the derivative subroutine calls present will not be executed. However, if the call is part of the normal execution of the code, the flag is set equal to 1 and the derivative subroutine call will be made. The RDR module also adds on the flag variable to the arguments of the original subroutines to allow this functionality. With the calls inserted at the proper places, the derivative subroutines are appended to the end of the code and the newly assembled derivative script is outputted to a file. The flowchart for the RDR module is provided in Figure 4.

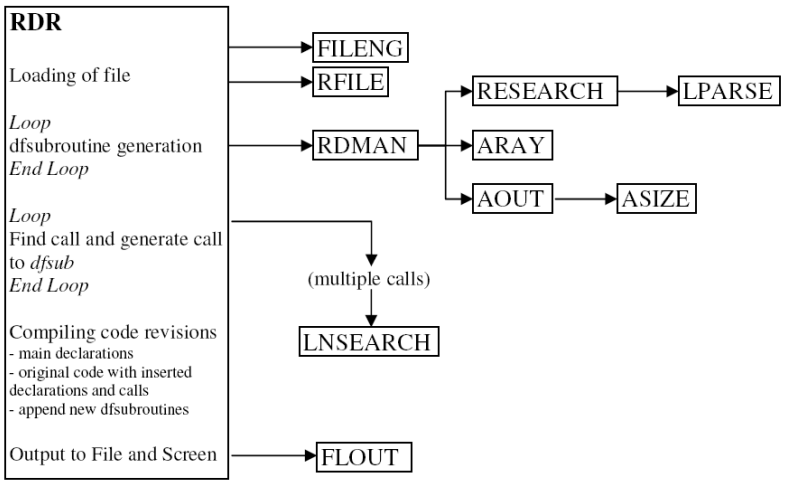

Figure 4: Flowchart of the RDR module

There are several common subroutines used within the SUPRDR code, they can be classified into three groups: file handling, code interpretation, line reading utilities. The file handling subroutines are FLENG, RFILE and FLOUT. FLENG reads the input file and determines its length by checking the first characters up until end of file and returns the length. The RFILE subroutine uses the length value and reads the input file storing it line by line in the array "Line" which is then returned. FLOUT takes in the modified script which is stored in the array "outfile" and writes it to a given output filename. The code interpretation subroutines are RESEARCH, ARAY and AOUT. The RESEARCH subroutine gathers variables from the original subroutine in question collecting the names of all the real variables. In addition, this subroutine collects all common variables when the 'keyword' input is selected as "common". The ARAY subroutine checks for arrays in the common blocks or arguments and expands them out to individual variables so the derivatives can be computed. The AOUT subroutine explores the source subroutine and records all the variables that are assigned a value, this list is compared with the "variables under question" list and any matches are deemed to be outputs. AOUT calls the subroutine ASIZE in order to compute the sizes of any arrays that are encountered in the assignment statements. These arrays are then expanded out in the AOUT subroutine. The line reading utilities are called multiple times by all other subroutines and are used throughout the code, they include LPARSE and LNSEARCH. LPARSE is a simple subroutine that takes in a large character variable of comma separated strings and produces and array of strings, which is then returned. LNSEARCH is used to search the original FORTRAN script held in the array "Line" for specific character strings using a given keyword input and a supplied upper and lower bound for the search. It returns the line location of the first incidence of the keyword (incidences within commented lines are not included) from the starting line and can function in both forward and backward modes based on the upper and lower bound values.

The SUPRDR code does all the background work necessary for creating a sensitivity space for a given problem and the file output is passed on to the SENSOR code to allow for final conversion from derivatives to sensitivities. The SENSOR code takes as its input the file outputted at the end of the SUPRDR run along with user defined information on the specific state variables in question and the locations of the characteristic equations. Many of the common subroutines used in SUPRDR are also used in the SENSOR code including FLENG, RFILE, FLOUT,

LNSEARCH, LPARSE, RESEARCH, ARAY, and ASIZE. The two codes have been left separate to ensure ease of use, to observe if conversions are taking place properly and because of the less automated nature of the SENSOR code which requires more user input. The sensitivity conversion relates the partial derivatives computed using the newly made derivative subroutines to the characteristic equations of the simulation.

3.2 SENSOR CODE

The SENSOR code analyzes the top level of the simulation and separates out the variables into state variables, parameters and constructs. It then takes the user indicated characteristic equations and relates them to the derivative variables created by the derivative subroutines. In general the code will recognize a characteristic equation of the form seen in Equation 5 and then find a relation between Φ and the derivative variables that are available to produce the list of Φ derivatives (dΦ). Once these are determined they are placed into the matrices defined in Equations 3 and 4 in order to compute the sensitivity space. The dΦ lists are split into 3 groups, derivatives of state variables at time n, state variables at time n+1 and parameters. For each characteristic equation the 3 dΦ lists are created and assembled into three matrices PhiSVC, PhiSVN and PhiPAR that compute the sensitivity space. The SENSOR code inserts the code necessary to convert the derivative variables to dΦ values, assemble the necessary matrices and calculate the sensitivity space matrices J_α and J_{s0}. The calculation process uses Equation 6 and 7 to determine the sensitivity space for the parameters and initial values of the state variables (J_α and J_{s0}). The two matrices are then outputted at each time step.

$$\text{Parameters: } (I-\phi_{SVN})(J_\alpha^{n+1}) = (I+\phi_{SVC})(J_\alpha^{n}) + \phi_{PAR} \quad (6)$$

$$\text{Initial State Variables: } (I-\phi_{SVN})(J_{s0}^{n+1}) = (I+\phi_{SVC})(J_{s0}^{n}) \quad (7)$$

The SENSOR code adds the necessary script to perform the conversion to sensitivity space as described above and reassembles the code outputting the new file. This is the final step in the conversion of the original source code to a modified source that includes a sensitivity space calculation. A flow chart of the SENSOR code is provided in Figure 5.

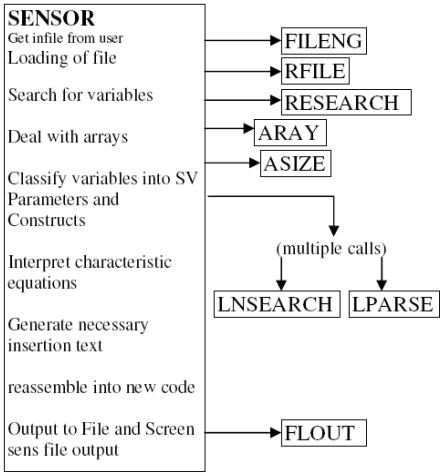

Figure 5: Flowchart for the SENSOR code

4. Application of SUPRDR and SENSOR to simple test simulations

The sensitivity conversion software was tested through the application to two problems of increasing difficulty. The initial test was an analytical system of two differential equations as depicted in Equation 8 with the solution defined in Equation 9.

$$\text{System: } \begin{cases} \frac{\partial P}{\partial t} = -P + aT + b \\ \frac{\partial T}{\partial t} = -T + c \\ P(t=0) = P_0, T(t=0) = T_0 \end{cases} \quad (8)$$

$$\text{Solution: } \begin{cases} P = P_0 e^{-t} + at(T_0 - c)e^{-t} - (ac + b)e^{-t} + ac + b \\ T = T_0 e^{-t} + c(1 - e^{-t}) \end{cases} \quad (9)$$

The program outputs for P, T and the sensitivity profile for the set of parameters (a, b, c) were compared to the predicted analytical results with excellent agreement. The analytical and code sensitivity comparisons are shown in Figure 6.

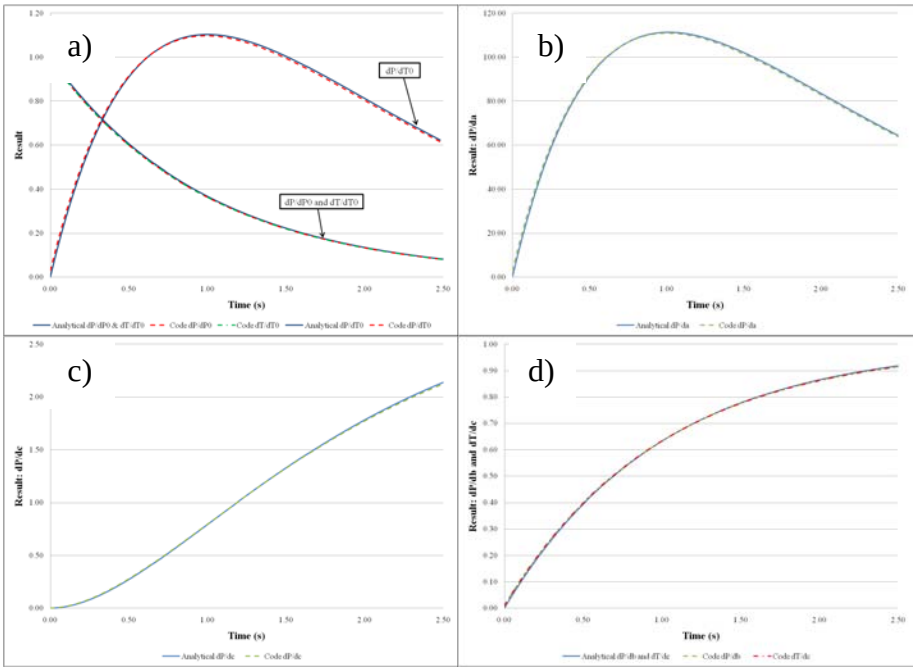

Figure 6: Comparison of analytical and code results for sensitivity system test
[(a): Initial Conditions (b, c, d): Parameters]

The second test of the code was the application to a simple Gas Blowdown simulation coded in FORTRAN-95 and defined as per reference 1. The problem simulates the venting of a cylindrical tank of Nitrogen into external standard atmospheric conditions. The simulation tracks state variables including the pressure and temperature in the tank and the wall temperature values of the tank wall at various points within the wall.

The nominal blowdown simulation code was then fed into the SUPRDR code pack which iterated 6 times upgrading parameters and then ran the derivative generation module. The completed derivative output file was then sent to the SENSOR code and was processed to produce the sensitivity space code and then outputted. The completed edited code was then tested. The sensitivities were calculated at each time concurrently with the running of the code and the results were compared with the results data from analytical sensitivity calculations by Petruzzi [1]. The auto-sense blowdown code showed good agreement in the sensitivity space for all relevant values. The data for both the initial state values and the parameters are in good agreement and the code seems to work well, albeit slower than the basic simulation.

5. Future application to RELAP 5 full system thermalhydraulics code

As stated earlier, if direct analytical methods are employed in conjunction with the development of a system simulation code the extra effort needed is minimal. Unfortunately, the standard thermalhydraulics codes used in nuclear analysis were not developed in this manner. Therefore, in order to apply direct methods to these codes, efforts must be made to dissect the codes and determine the proper way to integrate the necessary components into the code. For complex full system codes that are in use today this requires significant effort and skill to properly accomplish. Large system thermalhydraulics codes in use in nuclear analysis such as RELAP5, CATHARE and CATHENA have very high levels of complexity and must be carefully examined in order to breakdown the model equations to the level of state variables and user defined input parameters.

RELAP5, like any other complex full system code, is made up of multiple subroutines designed to model various thermalhydraulic phenomenon. The level of complexity present would necessitate extensive efforts to fully employ direct methods throughout all parts of the code. However, it may be feasible to integrate a direct analytical method into a specific subroutine with an acceptable level of effort. This can lead to the development of a defined procedure to employ direct analytical analysis which can then be applied to other subroutines relatively quickly and easily, building the sensitivity system step by step.

The SUPRDR and SENSOR code packages are to be applied to RELAP 5. This application is an involved and complex undertaking and hence will require much skill and effort over time. The first stage is the application of the codes to the choked flow subroutine. The choked flow subroutine derives information from the hydrodynamic and heat structure modules of RELAP so these must also be assessed in the pursuit of the main goal.

The AutoSense code package, SUPRDR and SENSOR, are being applied to very small portions of the RELAP code. Building on the success with the blowdown simulation the application of the sensitivity codes in a limited fashion to the choked flow subroutine, specifically the two phase Henry-Fauske model, subroutine gctpm, is being attempted. This study is being performed by upgrading a small number of internal parameters in the gctpm subroutine up to the jchoke parent subroutine and computing the partial derivatives through a newly created dfgctpm subroutine. Preliminary testing with 5 parameters has proved effective in an operational sense. Data is required to compare the output to ensure its validity. This effort was made possible by

running the SUPRDR code on the jchoke and gctpm subroutines in isolation from the rest of the RELAP source and producing modified subroutines and the new dfgctpm subroutine. The next step to continue this effort is to upgrade the parameters a few more levels up to the point where state variables such as Pressure and internal energy and constructs such as temperature and mass flow are calculated. Once this upgrading is achieved the partial derivatives for these responses with respect to the parameters can be computed. If this works out the SENSOR code can be applied and direct sensitivities can be produced. This effort will serve as a proof of concept and can be used to define a detailed procedure for applying the sensitivity calculation package to various subroutines in the RELAP5 Mod 3.3 source code. Eventually given enough time and effort the sensitivity package can be applied to large portions of the code resulting in a new version of RELAP with a detailed sensitivity space that is calculated concurrently with simulations and would work for any nodalization design.

6. Conclusion

Direct uncertainty and sensitivity analysis is an increasingly vital research area to provide proper assessment of probabilistic best estimate codes. Direct methods promote detailed investigation of the inner workings of the simulation to produce sensitivity measurements through the use of partial derivatives. Despite the skill and effort required to apply direct methods to complex simulation codes, the procedure is only required once to cover all scenarios that the code is able to simulate greatly reducing the computational efforts necessary. By integrating direct sensitivity calculations into the simulation code, consistent analysis is made available to all users and a wealth of sensitivity information is provided.

An automated sensitivity generation code package, AutoSense, has been created that reads in FORTRAN coded TH simulations and outputs a modified simulation that includes sensitivity space calculations. The package has been successfully applied to a simple gas blowdown simulation with positive results. The code is in the process of being applied to portions of the RELAP5 Mod 3.3 TH simulation software in the choked flow subroutine. These efforts will be used to build a procedure for application of the sensitivity package in RELAP which could eventually lead to a new RELAP source that includes detailed sensitivity calculations concurrent with the base simulation.

7. References

[1] Petruzzi, A. “Development and Application of Methodologies for Sensitivity Analysis and Uncertainty Evaluation of the Results of the Best Estimate System Codes Applied in Nuclear Technology”, Doctoral Thesis, University of Pisa, Pisa PI Italy, 2008, Sec. 6.2.

[2] Bischof, C. et. al. "ADIFOR - Generating Derivative Codes from Fortran Programs", Journal of Scientific Programming, Vol. 1, Iss. 1, 1992, pp. 1-29.