



International Agreement Report

Development of a Computer Tool for In-Depth Analysis and Post Processing of the RELAP5 Thermal Hydraulic Code

Prepared by:

R. H. Santos, J. M. Izquierdo Rocha

Consejo de Seguridad Nuclear
Justo Dorado, 11
28040 Madrid, Spain

A. Calvo, NRC Project Manager

**Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001**

April 2011

Prepared as part of
The Agreement on Research Participation and Technical Exchange
Under the International Code Assessment and Maintenance Program (CAMP)

**Published by
U.S. Nuclear Regulatory Commission**

AVAILABILITY OF REFERENCE MATERIALS IN NRC PUBLICATIONS

NRC Reference Material

As of November 1999, you may electronically access NUREG-series publications and other NRC records at NRC=s Public Electronic Reading Room at <http://www.nrc.gov/reading-rm.html>. Publicly released records include, to name a few, NUREG-series publications; *Federal Register* notices; applicant, licensee, and vendor documents and correspondence; NRC correspondence and internal memoranda; bulletins and information notices; inspection and investigative reports; licensee event reports; and Commission papers and their attachments.

NRC publications in the NUREG series, NRC regulations, and *Title 10, Energy*, in the Code of *Federal Regulations* may also be purchased from one of these two sources.

1. The Superintendent of Documents
U.S. Government Printing Office
Mail Stop SSOP
Washington, DC 20402B0001
Internet: bookstore.gpo.gov
Telephone: 202-512-1800
Fax: 202-512-2250
2. The National Technical Information Service
Springfield, VA 22161B0002
www.ntis.gov
1B800B553B6847 or, locally, 703B605B6000

A single copy of each NRC draft report for comment is available free, to the extent of supply, upon written request as follows:

Address: Office of the Chief Information Officer,
Reproduction and Distribution
Services Section
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001
E-mail: DISTRIBUTION@nrc.gov
Facsimile: 301B415B2289

Some publications in the NUREG series that are posted at NRC=s Web site address <http://www.nrc.gov/reading-rm/doc-collections/nuregs> are updated periodically and may differ from the last printed version. Although references to material found on a Web site bear the date the material was accessed, the material available on the date cited may subsequently be removed from the site.

Non-NRC Reference Material

Documents available from public and special technical libraries include all open literature items, such as books, journal articles, and transactions, *Federal Register* notices, Federal and State legislation, and congressional reports. Such documents as theses, dissertations, foreign reports and translations, and non-NRC conference proceedings may be purchased from their sponsoring organization.

Copies of industry codes and standards used in a substantive manner in the NRC regulatory process are maintained at:

The NRC Technical Library
Two White Flint North
11545 Rockville Pike
Rockville, MD 20852B2738

These standards are available in the library for reference use by the public. Codes and standards are usually copyrighted and may be purchased from the originating organization or, if they are American National Standards, from:

American National Standards Institute
11 West 42nd Street
New York, NY 10036B8002
www.ansi.org

Legally binding regulatory requirements are stated only in laws; NRC regulations; licenses, including technical specifications; or orders, not in NUREG-series publications. The views expressed in contractor-prepared publications in this series are not necessarily those of the NRC.

The NUREG series comprises (1) technical and administrative reports and books prepared by the staff (NUREGBXXXX) or agency contractors (NUREG/CRBXXXX), (2) proceedings of conferences (NUREG/CPBXXXX), (3) reports resulting from international agreements (NUREG/IABXXXX), (4) brochures (NUREG/BRBXXXX), and (5) compilations of legal decisions and orders of the Commission and Atomic and Safety Licensing Boards and of Directors= decisions under Section 2.206 of NRC=s regulations (NUREGB0750).

212B642B4900

DISCLAIMER: This report was prepared as an account of work sponsored by an agency of the U.S. Government. Neither the U.S. Government nor any agency thereof, nor any employee, makes any warranty, expressed or implied, or assumes any legal liability or responsibility for any third party=s use, or the results of such use, of any information, apparatus, product, or process disclosed in this publication, or represents that its use by such third party would not infringe privately owned rights.

Development of a Computer Tool for In-Depth Analysis and Postprocessing of the RELAP5 Thermal Hydraulic Code

Manuscript Completed: April 2003
Date Published: March 2011

Prepared by: Roberto Herrero Santos* and José María Izquierdo Rocha*

*Consejo de Seguridad Nuclear
Justo Dorado, 11
28040 Madrid – Spain

A. Calvo, NRC Project Manager

Prepared for:
Division of Systems Analysis
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001



ABSTRACT

The Spanish Nuclear Safety Council (CSN) has sponsored the development of a computer tool for the error diagnosis and interpretation of the results obtained with the code RELAP5/MOD3.2.

The report describes the tests used to check the consistency of the results and illustrates some application examples that point out some deficiencies in the code.

CONTENTS

	<u>Page</u>
ABSTRACT	iii
LIST OF FIGURES	viii
EXECUTIVE SUMMARY	xiii
1 PROBLEM SETUP	1-1
1.1 Problems concerning the use of thermal hydraulic codes in Nuclear Safety	1-1
1.1.1 Analysis of thermal hydraulic codes	1-1
1.1.2 Checking the solutions computed by the TH codes	1-2
1.1.3 Interpretation of results	1-4
1.1.4 User effect	1-4
1.2 Objectives	1-5
2 VERIFICATIONS IMPLEMENTED IN THE ANALYSIS TOOL	2-1
2.1 Introduction	2-1
2.2 Mass conservation equation	2-1
2.2.1 Analytical derivation	2-2
2.2.2 Discretized form	2-2
2.3 Mixture momentum equation	2-3
2.4 Pressure and volume equations	2-5
2.4.1 Analytical derivation	2-5
2.4.2 Discretized form	2-8
2.5 Non-equilibrium mass equation	2-9
2.5.1 Analytical derivation	2-9
2.5.2 Discretized form	2-11
2.5.3 Uncoupling of the non-equilibrium mass equations	2-12
2.6 Correspondence of variables	2-15

3	DESCRIPTION OF THE ANALYSIS TOOL	3-1
3.1	Introduction	3-1
3.2	Description of the programs used in the analysis tool	3-1
3.2.1	The RELAP5 code	3-1
3.2.2	The BABIECA simulation language	3-1
3.2.3	Parallel Virtual Machine (PVM)	3-2
3.2.4	The text editor AWK	3-2
3.2.5	XMGR5	3-3
3.3	Linkage of the codes	3-3
3.3.1	The BABIECA-RELAP5 coupling	3-3
3.3.2	The role of the AWK editor	3-4
3.3.3	The role of the XMGR5 program	3-5
3.4	Functioning of the analysis tool	3-5
4	APPLICATION EXAMPLES	4-1
4.1	Introduction	4-1
4.2	Compression caused by heating in a closed pipe	4-1
4.2.1	Example description	4-1
4.2.2	Analysis of results	4-2
4.3	Depressurization in a circuit with liquid and vapor	4-5
4.3.1	Justification of the example	4-5
4.3.2	Verification of the balance equations	4-12
4.3.3	Interpretation of results	4-13
4.3.4	Verification of the simplified non-equilibrium model	4-17
4.4	Over specification of boundary conditions	4-18
4.4.1	Motivation and background	4-18
4.4.2	Original case	4-21
4.4.3	Zhang's problem with PV term	4-21
4.4.4	Over specification suppression with pressure feedback	4-23
4.5	Full plant model. Flow mixing problem	4-29
4.5.1	Introduction	4-29
5	CONCLUSIONS	5-1

6	RECOMMENDATIONS FOR FUTURE DEVELOPMENTS	6-1
6.1	Analysis of other codes and models	6-1
6.2	Identification of lower order models	6-1
6.3	Development of codes with error control	6-2
7	REFERENCES	7-1
Appendix A. Nomenclature		A-1
Appendix B. Partial derivatives of thermodynamic properties		B-1
Appendix C: File babpost.awk		C-1

Figures

	<u>Page</u>
1 Functioning diagram of the analysis tool.	3-5
2 Nodalization used in the example.	4-1
3 Power introduced in the pipe.	4-2
4 Verification of the pressure equation. Pipe 402.	4-3
5 Verification of the level equation. Pipe 402.	4-3
6 Verification of the vapor mass. Pipe 402.	4-3
7 Verification of the liquid mass. Pipe 402.	4-3
8 Non-equilibrium vapor mass. Pipe 402.	4-4
9 Non-equilibrium liquid mass. Pipe 402.	4-4
10 Terms of the pressure equation. Pipe 402.	4-4
11 Verification of the pressure equation. Pipe 500.	4-6
12 Verification of the level equation. Pipe 500.	4-6
13 Verification of the vapor mass. Pipe 500.	4-6
14 Verification of the liquid mass. Pipe 500.	4-6
15 Non-equilibrium vapor mass. Pipe 500.	4-7
16 Non-equilibrium liquid mass. Pipe 500.	4-7
17 Terms of the pressure equation. Pipe 500.	4-7
18 Verification of the pressure equation. Pipe 402. Revised example.	4-8
19 Verification of the level equation. Pipe 402. Revised example.	4-8
20 Verification of the vapor mass. Pipe 402. Revised example.	4-8
21 Verification of the liquid mass. Pipe 402. Revised example.	4-8
22 Non-equilibrium vapor mass. Pipe 402. Revised example.	4-9
23 Non-equilibrium liquid mass. Pipe 402. Revised example.	4-9
24 Terms of the pressure eq. Pipe 402. Revised example.	4-9
25 Verification of the pressure equation. Pipe 500. Revised example.	4-10
26 Verification of the level equation. Pipe 500. Revised example.	4-10
27 Verification of the vapor mass. Pipe 500. Revised example.	4-10
28 Verification of the liquid mass. Pipe 500. Revised example.	4-10
29 Non-equilibrium vapor mass. Pipe 500. Revised example.	4-11
30 Non-equilibrium liquid mass. Pipe 500. Revised example.	4-11

31	Terms of the pressure eq. Pipe 500. Revised example.	4-11
32	Circuit nodalization.	4-12
33	Pressure evolution.	4-13
34	Pressure analysis. Break in vapor region.	4-14
35	Liquid volume analysis. Break in vapor region.	4-14
36	Vapor mass analysis. Break in vapor region.	4-14
37	Liquid mass analysis. Break in vapor region.	4-14
38	Pressure analysis. Break in liquid region.	4-15
39	Level analysis. Break in liquid region.	4-15
40	Vapor mass analysis. Break in liquid region.	4-15
41	Liquid mass analysis. Break in liquid region.	4-15
42	Energy flow released through the breaks.	4-16
43	Enthalpy flow released through the breaks.	4-16
44	Terms of pressure eq. Break in vapor region.	4-16
45	Terms of pressure eq. Break in liquid region.	4-16
46	System compressibility.	4-17
47	Neglection of term with Γ_i in the non-equilibrium eq. Vapor phase. Volume 10001. Break in vapor region.	4-19
48	Approximation of the interfacial heat power in the vapor phase. Volume 10001. Break in vapor region.	4-19
49	Neglection of term with Γ_i in the non-equilibrium eq. Liquid phase. Volume 20001. Break in vapor region.	4-19
50	Approximation of the interfacial heat power in the liquid phase. Volume 20001. Break in vapor region.	4-19
51	Neglection of term with Γ_i in the non-equilibrium eq. Vapor phase. Volume 10001. Break in liquid region.	4-20
52	Approximation of the interfacial heat power in the vapor phase. Volume 10001. Break in liquid region.	4-20
53	Neglection of term with Γ_i in the non-equilibrium eq. Liquid phase. Volume 20001. Break in liquid region.	4-20
54	Approximation of the interfacial heat power in the liquid phase. Volume 20001. Break in liquid region.	4-20
55	Zhang's problem.	4-21
56	Zhang's problem: pressure postprocessing.	4-22

57	Zhang's problem: level postprocessing.	4-22
58	Zhang's problem: vapor mass postprocessing.	4-22
59	Zhang's problem: non-equilibrium vapor mass postprocessing.	4-22
60	Zhang's problem: pressure drop postprocessing.	4-23
61	Zhang's problem: vapor temperature.	4-24
62	Zhang's problem: vapor enthalpy.	4-24
63	Zhang's problem: vapor density.	4-24
64	Zhang's problem: pressure.	4-24
65	Vapor temperature with PV term.	4-25
66	Vapor enthalpy with PV term.	4-25
67	Vapor density with PV term.	4-25
68	Pressure with PV term.	4-25
69	Pressure equation with momentum calculation.	4-30
70	Liquid volume equation with momentum calculation.	4-30
71	Mass equation with momentum calculation.	4-30
72	Non-equilibrium mass equation with momentum calculation.	4-30
73	Pressure drop equation with momentum calculation.	4-31
74	Vapor temperature with momentum calculation.	4-32
75	Enthalpy with momentum calculation.	4-32
76	Vapor density with momentum calculation.	4-32
77	Pressure with momentum calculation.	4-32
78	Volumetric flow in junction 5 of the downcomer.	4-33
79	Analysis of the pressure equation in the downcomer.	4-34
80	Analysis of the liquid volume equation in the downcomer.	4-34
81	Analysis of the vapor mass equation.	4-34
82	Analysis of the liquid mass equation in the downcomer.	4-34
83	Analysis of the non-equilibrium vapor mass equation in the downcomer.	4-35
84	Analysis of the non-equilibrium liquid mass equation in the downcomer.	4-35
85	Terms of the pressure equation in the downcomer.	4-35
86	Analysis of the pressure equation in downcomer subsystem.	4-36
87	Analysis of the level equation in downcomer subsystem.	4-36
88	Analysis of the vapor mass equation in downcomer subsystem.	4-36

89	Analysis of the liquid mass equation in downcomer subsystem.	4-36
90	Analysis of the non-equilibrium vapor mass equation in downcomer subsystem. .	4-37
91	Analysis of the non-equilibrium liquid mass equation in downcomer subsystem. .	4-37
92	Pressure equation terms in downcomer subsystem.	4-37
93	One-dimensional flow mixing.	4-38
94	Vapor mass flows in the downcomer.	4-40
95	Liquid mass flows in the downcomer.	4-40
96	Vapor energy flows in the downcomer.	4-40
97	Liquid energy flows in the downcomer.	4-40
98	Verification and validation processes.	6-3
99	Collapsed verification and validation processes.	6-3

EXECUTIVE SUMMARY

A computer tool has been developed to assist RELAP5 users in the interpretation and qualification of pressure and level transients.

For that purpose some balance equations have been rigorously derived from the original mass and energy equations. Those balances include the pressure, level and non-equilibrium mass equations. The balances are used to check the consistency of the solution and to interpret the pressure and level evolution in terms of physically meaningful variables.

Verification of the equations is implemented in the tool with the aid of an AWK script and the general purpose simulation language BABIECA. The AWK script automatically generates the BABIECA input file necessary to check the RELAP5 transient. The execution of BABIECA with the corresponding input file implies the spawning of the RELAP5 transient. RELAP5 information is transferred to BABIECA through PVM, so that the balance equations can be verified.

The tool has been applied to some application examples, which have revealed some misconceptions in RELAP5, such as the overspecification of boundary conditions and the flow mixing problems.

Some guidelines for the development of future codes, with full control of the discretization errors, are pointed out.

1 PROBLEM SETUP

1.1 Problems concerning the use of thermal hydraulic codes in Nuclear Safety

1.1.1 Analysis of thermal hydraulic codes

Nuclear Safety specialists face very often the need of simulating the transients expected to occur in nuclear power plants, in order to determine and quantify the associated risks. This need can be found in a wide variety of activities such as the conception and design of plant systems, optimization of procedures and set-points, operator training simulators, success criteria in PSAs, accident analyses, licensing tasks, etc.

The simulation of the operating and accidental transients that may happen in power plants based on a nuclear fission process requires, as a previous step, the description of the physical systems involved in the plant by means of mathematical models. A mathematical model could be defined as a set of, generally partial, differential equations capable of describing and predicting certain phenomena in a range of applicability. The predictive capability of the models must be considered as a mandatory requirement, since a set of equations that can only reproduce some transients by tuning certain parameters must never be regarded as a model. The predictive capability is inherent to the concept of model.

The solution of the equations involved in the models that describe a nuclear power plant is a very difficult problem. An analytical solution of the equations can never be achieved in practice. The most common approach consists in solving a discretized version of the differential equations by means of a programmable computer algorithm. The resulting program is usually called a code. The nuclear industry counts on many codes focused on the different disciplines involved in Nuclear Safety: neutronics, containment building, structural analysis, etc. thermal hydraulic codes, and specially the RELAP5 code [95], must not be forgotten.

The thermal hydraulic codes simulate the behavior of water and steam circuits used to transport the energy generated in the reactor. Despite their undeniable advantages, the use of computer codes implies a numeral of problems which are very frequently underestimated or not taken into account. For instance, the solution of the discretized equations is subject to errors that make very difficult to ensure how good the solution is. Interpretation of results is another remarkable problem. Very often the users regard the codes as black boxes that give an answer to the problems set through the input file, forgetting that they are formed by mathematical models and that they never substitute the plants perfectly. Other problems are related to the interaction between the codes and the users: build-up of the input file, correct interpretation of the data supplied to the code, nodalization, decisions about the different options available in the code, etc. As a consequence, the code may be told to solve a problem different from that the user would like to solve. On the other hand, the user may obtain wrong conclusions because of a misunderstanding of the output information computed by the code.

Some of the problems mentioned above are described in a greater detail in the following sections.

1.1.2 Checking the solutions computed by the TH codes

Disagreement between experimental results and the output data computed by a code is not unusual at all. The causes of the discrepancies, usually referred to as *uncertainties*, may be classified as follows:

1. *External* sources, originated in the modelling process. Two different types can be defined in turn:
 - (a) The discrepancies between the analytical equations of the models and the real world, which may be originated in deficiencies in the instrumentation used to obtain the experimental correlations, inappropriate curve fitting, wrong modelling hypotheses, etc.
 - (b) Hypersensitivity of the model to certain parameters, initial conditions or boundary conditions, which cannot be measured with the required precision. As a result, some output variables may lie within a very broad range as the value of the above mentioned parameters is changed within the measuring error interval.
2. *Internal* sources, related to the discretization process of the analytical equations and the discrete nature of digital computers. Three types can be noticed:
 - (a) Round-off errors, caused by the finite precision with which floating-point numbers are represented in a digital computer. The representation may be different depending on the machine and even on the compiler [99, 25]. Round-off errors are the root cause of the uncertainties attributed to machines and compilers in references such as [16, 65].
 - (b) Truncation errors, which arise whenever a Taylor series is used. A discrete numerical method is a good example, since it implies in most cases a Taylor expansion up to a certain order. Truncation errors also appear when trying to represent in a digital computer non-rational numbers such as e , π , square roots, trigonometric functions, exponential functions, or any other transcendental function. Computation of that kind of numbers is always achieved by means of Taylor expansions. From the theoretical point of view, truncation errors can be as small as desired. For numerical methods, refinement of the spatio-temporal discretization mesh should decrease the error. In the case of the computation of non-rational numbers, calculation of higher order terms in the series should yield the same effect. But in real practice, the refinement of the mesh or the increment of the terms in the expansion results in more mathematical operations and hence in greater round-off errors. From a certain point on, round-off errors could even compensate for the decreasing truncation errors. The native floating-point precision with

fixed finite length used by digital computers is then the main source of errors in computer simulations.

The so called external sources of uncertainty can be hardly suppressed, since they are closely related to the very nature of mathematical modelling. Ensuring that the analytical equations model the real world accurately is the process known as *validation*. This work does not pay attention to that kind of uncertainties.

On the other hand, internal uncertainties could be eliminated more easily. Round-off errors can be decreased as much as desired getting rid of machine native floating-point arithmetic, and using instead a software implemented multiple precision arithmetic, fitted to the particular needs of each problem [1]. Once the round-off errors are controlled, the solution can be improved by progressive refinement of the discretization mesh.

Assuming that current codes do not implement multiple precision arithmetic, the quality of the computed solution must be ensured by other methods. Reference [110] considers that the numerical solution computed by a code is uniquely related to the analytical equations as long as the following items are taken into account:

1. Estimation of the associated errors.
2. Comparison with well known analytical solutions.
3. Replacement of the solution into the original equations.

The two first items above imply an *a priori* verification of the solution. On the other hand, item 3 is an *a posteriori* method.

With respect to item 1, binding of the truncation error may be a cumbersome task. First of all, it may be difficult to find an analytical expression for the error associated to a numerical method. Secondly, finding upper bounds of the error in certain ranges of the spatio-temporal domain might not be trivial.

Moreover, simulation codes result from coupling several sections, each of them accounting for certain physical models, yielding feedback loops. For instance, the RELAP5 code comprises a hydrodynamic section and a heat structure section among many others. Each of them is solved with its own numerical method. As a consequence, the boundary conditions of each section are already computed with a certain error, making it difficult to bind the error of the overall system.

With respect to item 2, the solution computed by the code can be compared to the analytical solution only in some cases. The water faucet and the oscillating manometer problems described in [83] are typical examples. Unfortunately, analytical solutions can be found in these cases at the expense of oversimplifying of the modelling hypotheses. For instance, constant and uniform pressure to avoid the use of water property tables, separated phases, no mass, momentum and energy interchange between phases, and incompressible and non viscid flows, are common hypotheses. The range of physical phenomena tested with those examples

is then rather constrained, specially when compared to the range of all possible phenomena that may occur during a transient. Comparison with analytical solutions seems to be suitable just to test the numerical methods during the earlier stages of the developmental process. Fulfilment of those tests is anyway a necessary but not a sufficient condition to ensure the quality of the code.

An *a posteriori* verification of each particular calculation is much more advisable according to [85, 50, 70]. Substitution of the solution into the original analytical equations seems to be the natural way to check the results. However, this approach faces a great obstacle: solutions in the discrete domain must be substituted in equations defined in a continuous domain. Either continuous functions must be obtained from the discrete solutions, or the analytical equations must be discretized before replacement.

The first choice implies fitting the discrete results to continuous curves with a predefined shape. The curves should be then replaced into the analytical equations. The most important problem with this approach is that the higher the order of a derivative, the less reliable it is. The comparison could lead thus to negative conclusions even if the discrete solution is acceptable.

The second choice is the replacement of the discrete solution into a discretized form of the analytical equations. If the discretization is just the numerical method used to obtain the discrete solution, the problem becomes trivial. That is the reason why the discretized equations in this work are not the original ones but other rigorously derived from them. Section 2 is mainly devoted to the derivation of these new equations.

1.1.3 Interpretation of results

Another problem that the code users face usually is the right interpretation of the results computed by the code [48]. The codes supply a large amount of output information. However, only a small fraction is used by the user to analyze the results. The right interpretation of the phenomena occurred during the transient may be hindered by several factors:

- Ignorance of the analysts about the models actually included in the code.
- Tendency of the analysts to explain the results in terms of models simpler than those embedded in the code (the homogeneous equilibrium model is perhaps the most representative example).
- Tendency of the analysts to explain the code results in terms of physical phenomena not modelled at all in the code.

1.1.4 User effect

Thermal hydraulic codes have several options, many of which are taken by default, that must be chosen by the users in the input file. These options allow for the selection of different

models, solution procedures, convergence criteria, etc. Furthermore, different users may solve the same problem by means of different strategies and procedures (nodalization, time step, cross-flow or normal junctions, models, etc.), leading to dramatically different results. This is what is called the undesirable *user effect*, which has caused a great concern in the international community [6].

1.2 Objectives

The main objective of this work is the development of a computer tool to help the users of thermal hydraulic codes, and particularly of RELAP5, to solve or alleviate the problems described in section 1.1. The proposed tool will be achieved by integrating the RELAP5 code in a higher level computing environment that allows for the on-line calculation of new variables, balances, verifications and information in addition to those available in the code output files. The tool will comprise a number of computer programs linked with each other according to a given standard for code connection. Such a standard is the object of reference [36].

In a finer detail, the tool will be capable of:

- Finding errors in the input file.
- Determining the local contribution of the several contributors to the global result of the code.
- Checking the consistency of the calculation with the original equations.
- Comparing different codes, models and even machines and compilers. This may help the user to ensure that his assumptions to interpret the code results are valid at least in the particular transient.

As a summary, the proposed tool will execute what we shall call *in-depth postprocessing* of the RELAP5 solutions, which must not be confused with the more traditional concept of postprocessing that makes reference to user friendly output visualization.

The analysis tool will comprise the following codes:

- The modular simulation language BABIECA [49], which have been coupled to RELAP5, providing new computation capabilities.
- The Parallel Virtual Machine (PVM) [28, 27] message passing library functions. These functions will perform the information exchange between RELAP5 and BABIECA at every time step, so the latter code can check the solution computed by the former one.
- The graphical visualization program XMGR5 [101, 52], which will be used to plot the results obtained by both RELAP5 and the analysis tool.

- The standard UNIX, non-interactive text processor AWK, which will build automatically the BABIECA input file needed to execute the analysis of a given RELAP5 case, once the user has specified the hydraulic subsystems and control volumes to be checked.

2 VERIFICATIONS IMPLEMENTED IN THE ANALYSIS TOOL

2.1 Introduction

This section shows the verifications and additional computations that have been implemented in the analysis tool up to now. These verifications will enable us to achieve the goals stated in section 1, i.e., interpretation of results, assurance of the quality of the solution computed by RELAP5, fulfillment of the balance equations, comparison of different models and machines, etc.

The starting point for the verifications is the following set of time and space averaged equations for the hydrodynamic section of the RELAP5 code. The derivation of these equations can be found in [41, 82]. The nomenclature used hereinafter can be found in appendix 6.3.

CONSERVATION OF MASS:

$$\frac{\partial}{\partial t}(\alpha_k A \rho_k) + \frac{\partial}{\partial x}(\alpha_k A \rho_k v_k) = A \Gamma_k \quad (1)$$

CONSERVATION OF MOMENTUM:

$$\begin{aligned} \frac{\partial}{\partial t}(\alpha_k A \rho_k v_k) + \frac{\partial}{\partial x}(\alpha_k A \rho_k v_k^2) = & -\alpha_k A \frac{\partial P}{\partial x} + \alpha_k \rho_k B_x A \\ & - \alpha_k \rho_k A F W_k v_k + A \Gamma_k v_i - \alpha_k \rho_k A F I_k (v_k - v_{k'}) \\ & - C \alpha_k \alpha_{k'} \rho_m A \left[\frac{\partial}{\partial t}(v_k - v_{k'}) + v_{k'} \frac{\partial v_k}{\partial x} - v_k \frac{\partial v_{k'}}{\partial x} \right] \\ & - \frac{1}{2} K \alpha_k \rho_k |v_k| v_k \delta(x - x_0) \end{aligned} \quad (2)$$

CONSERVATION OF ENERGY:

$$\begin{aligned} \frac{\partial}{\partial t}(\alpha_k A \rho_k u_k) + \frac{\partial}{\partial x}(\alpha_k A \rho_k u_k v_k) = & -A P \frac{\partial \alpha_k}{\partial t} - P \frac{\partial}{\partial x}(\alpha_k A v_k) \\ & + A Q_{wk} + A Q_{ik} + A \Gamma_{ik} h_k^* + A \Gamma_{wk} h_k' + A \text{DISS}_k \end{aligned} \quad (3)$$

In the following sections the new equations used to analyse the RELAP5 results, both in analytical and discretized form, will be derived.

2.2 Mass conservation equation

The RELAP5 code currently checks the conservation of the mixture mass in full thermal hydraulic systems. This way of controlling the calculation can fail because of two reasons:

- Positive mass errors in certain control volumes may be compensated by negative mass errors in other volumes. In other words, the mass may be globally well computed, although the local distribution may be wrong.
- Relatively large mass errors in the vapor phase may be masked when compared to the total mass, since the mass of liquid is in most cases much larger.

As a conclusion, checking of the mass errors in individual control volumes and small subsystems, for each phase, seems to be necessary. However, we must say that the first item has already been implemented in the 3.2.2beta version of RELAP5.

2.2.1 Analytical derivation

If we integrate equation (1) along a thermal hydraulic system of length L we obtain:

$$\frac{dm_k}{dt} = \int_0^L A\Gamma_k dx + W_k^{\text{in}} - W_k^{\text{out}} \quad (4)$$

where

- $m_k \equiv \int_0^L \alpha_k A \rho_k dx$ is the mass of phase k in the system and
- $W_k \equiv \alpha_k A \rho_k v_k$ is the mass flow.

Integrating with respect to time we obtain the integral form:

$$m_k(t) = m_k(0) + \int_0^t \left(\int_0^L A\Gamma_k dx + W_k^{\text{in}} - W_k^{\text{out}} \right) dt \quad (5)$$

2.2.2 Discretized form

The fulfillment of the mass conservation equation in a RELAP5 calculation must be checked using a discretized form of equation (5), since the output information is only available at given points of the spatio-temporal mesh (nodes of the spatial discretization and time steps.) The mass in the thermal hydraulic system to be checked is approximated as follows:

$$m_k \approx \sum_{l=1}^N \alpha_k^l V^l \rho_k^l \quad (6)$$

where the superindex l ranges over the control volumes in the subsystem. In a similar way the space integral in the right hand side of equation (5) can be approximated by

$$\int_0^L A\Gamma_k dx \approx \sum_{l=1}^N V^l \Gamma_k^l \quad (7)$$

The inlet mass flow is computed as

$$W_k^{\text{in}} \equiv \alpha_k^{\text{in}} A^{\text{in}} \rho_k^{\text{in}} v_k^{\text{in}} \quad (8)$$

If several junctions are attached to the inlet side of the control volume, the following definitions hold, according to the flow mixing scheme described on section 3.1.7 of [95]:

$$\alpha_k^{\text{in}} \equiv \frac{\sum_{j=1}^M \alpha_k^j A^j}{\sum_{j=1}^M A^j} \quad (9)$$

$$\rho_k^{\text{in}} \equiv \frac{\sum_{j=1}^M \alpha_k^j A^j \rho_k^j}{\sum_{j=1}^M \alpha_k^j A^j} \quad (10)$$

$$v_k^{\text{in}} \equiv \frac{\left(\sum_{j=1}^M \alpha_k^j A^j \rho_k^j v_k^j \right) \left(\sum_{j=1}^M A^j \right)}{\left(\sum_{j=1}^M \alpha_k^j A^j \rho_k^j \right) A^{\text{in}}} \quad (11)$$

The superindex j ranges over all the junctions attached to the inlet face of control volume l . A^{in} is the inlet area. Similar definitions hold for the outlet face.

The discretized form of the mass conservation equation remains then as follows:

$$\frac{dm_k}{dt} \approx \sum_{l=1}^N V^l \Gamma_k^l + W_k^{\text{in}} - W_k^{\text{out}} \quad (12)$$

The right hand side of this equation may be integrated with respect to time with a trapezoidal rule. Its comparison with (6) provides the way to check the conservation of the mass of each phase.

2.3 Mixture momentum equation

Checking this equation could seem trivial at first sight, since it is one of the original equations solved by RELAP5. However, it will be useful to ensure that the pressure drop along a pipe equals the pressure difference between the inlet and outlet sides.

The sum of the two equations (2) yields:

$$\begin{aligned} \frac{\partial}{\partial t}(A\rho_m v_m) + \frac{\partial}{\partial x}(A\rho_m v_m^2) = & -\frac{\partial}{\partial x} \left(\frac{\alpha_g \alpha_f \rho_g \rho_f A}{\rho_m} (v_g - v_f)^2 \right) \\ & - A \frac{\partial P}{\partial x} + \rho_m A B_x - \alpha_g \rho_g A F W_g v_g - \alpha_f \rho_f A F W_f v_f \\ & - \frac{1}{2} K \alpha_g \rho_g |v_g| v_g \delta(x - x_0) - \frac{1}{2} K \alpha_f \rho_f |v_f| v_f \delta(x - x_0) \end{aligned} \quad (13)$$

where mixture density and mixture velocity are respectively defined as:

$$\rho_m = \alpha_g \rho_g + \alpha_f \rho_f \quad (14)$$

$$v_m = \frac{\alpha_g \rho_g v_g + \alpha_f \rho_f v_f}{\rho_m} \quad (15)$$

and x_0 stands for the location of the local pressure drop.

The convective term $\frac{\partial}{\partial x} \left(\frac{\alpha_g \alpha_f \rho_g \rho_f A}{\rho_m} (v_g - v_f)^2 \right)$ accounts for the difference of the two fluid model with respect to the homogeneous equilibrium model.

Integrating (13) along the direction x we get:

$$\begin{aligned} \frac{\partial}{\partial t} \int_0^L A \rho_m v_m dx + A \rho_m v_m^2 \Big|_{\text{out}} - A \rho_m v_m^2 \Big|_{\text{in}} = & \\ & \frac{\alpha_g \alpha_f \rho_g \rho_f A}{\rho_m} (v_g - v_f)^2 \Big|_{\text{in}} - \frac{\alpha_g \alpha_f \rho_g \rho_f A}{\rho_m} (v_g - v_f)^2 \Big|_{\text{out}} \\ & - \int_0^L A \frac{\partial P}{\partial x} dx + \int_0^L \rho_m A B_x dx \\ & - \int_0^L \alpha_g \rho_g A F W_g v_g dx - \int_0^L \alpha_f \rho_f A F W_f v_f dx \\ & - \frac{1}{2} K \alpha_g \rho_g |v_g| v_g - \frac{1}{2} K \alpha_f \rho_f |v_f| v_f \end{aligned} \quad (16)$$

We can assume that the cross-sectional is uniform within a control volume. Furthermore, if the integrands are also considered as uniform within the control volume, the previous equation becomes:

$$\begin{aligned} P^{\text{in}} - P^{\text{out}} = & \frac{\partial}{\partial t} (\rho_m v_m L) + \rho_m v_m^2 \Big|_{\text{out}} - \rho_m v_m^2 \Big|_{\text{in}} \\ & + \frac{\alpha_g \alpha_f \rho_g \rho_f}{\rho_m} (v_g - v_f)^2 \Big|_{\text{out}} - \frac{\alpha_g \alpha_f \rho_g \rho_f}{\rho_m} (v_g - v_f)^2 \Big|_{\text{in}} \\ & - \rho_m H g + \alpha_g \rho_g L F W_g v_g + \alpha_f \rho_f L F W_f v_f \\ & - \frac{1}{2} \frac{K}{A} \alpha_g \rho_g |v_g| v_g - \frac{1}{2} \frac{K}{A} \alpha_f \rho_f |v_f| v_f \end{aligned} \quad (17)$$

where the body force term has been transformed as follows:

$$\rho_m L B_x = \rho_m L g \cos(\theta) = \rho_m H g \quad (18)$$

The right hand side of the previous equation, computed at every time step with the output information provided by RELAP5, can be compared with the left hand side. Any discrepancy between both sides of the equation is an indicator of error in the solution.

2.4 Pressure and volume equations

These are new integral balances, derived from the mass and energy equations. It provides a new way of checking the RELAP5 results and interpreting the pressure and level evolution in TH systems.

2.4.1 Analytical derivation

First of all, we derive the enthalpy equation from the energy equation (3), taking into account that $h_k = u_k + P\vartheta_k$ and neglecting hereinafter the dissipation term, which is taken into account by RELAP5 only if a control volume contains a pump:

$$\begin{aligned} \frac{\partial}{\partial t}(\alpha_k A \rho_k h_k) + \frac{\partial}{\partial x}(\alpha_k A \rho_k h_k v_k) &= \alpha_k A \frac{\partial P}{\partial t} + \alpha_k A v_k \frac{\partial P}{\partial x} \\ &+ A Q_{wk} + A Q_{ik} + A \Gamma_{ik} h_k^* + A \Gamma_{wk} h_k' \end{aligned} \quad (19)$$

Taking into account the relationship

$$\frac{\partial h_k}{\partial t} + v_k \frac{\partial h_k}{\partial x} = \left[\frac{\partial \rho_k}{\partial t} + v_k \frac{\partial \rho_k}{\partial x} \right] \left(\frac{\partial h_k}{\partial \rho_k} \right)_P + \left[\frac{\partial P}{\partial t} + v_k \frac{\partial P}{\partial x} \right] \left(\frac{\partial h_k}{\partial P} \right)_{\rho_k} \quad (20)$$

the enthalpy conservation equation (19) is transformed into

$$\begin{aligned} \alpha_k A \left[\frac{\partial \ln \rho_k}{\partial t} + v_k \frac{\partial \ln \rho_k}{\partial x} \right] &= \\ \alpha_k A \vartheta_k \left[\frac{\partial P}{\partial t} + v_k \frac{\partial P}{\partial x} \right] \left(\frac{\partial \rho_k}{\partial h_k} \right)_P \left[\vartheta_k - \left(\frac{\partial h_k}{\partial P} \right)_{\rho_k} \right] & \\ + A \vartheta_k^2 \left(\frac{\partial \rho_k}{\partial h_k} \right)_P [Q_{wk} + Q_{ik} + \Gamma_{ik} h_k^* + \Gamma_{wk} h_k' - \Gamma_k h_k] & \end{aligned} \quad (21)$$

The last two factors of the first term in the right hand side of equation (21) can be simplified as follows:

$$\begin{aligned} \left(\frac{\partial \rho_k}{\partial h_k} \right)_P \left[\vartheta_k - \left(\frac{\partial h_k}{\partial P} \right)_{\rho_k} \right] &= \vartheta_k \left(\frac{\partial \rho_k}{\partial h_k} \right)_P + \left(\frac{\partial \rho_k}{\partial P} \right)_{h_k} = \\ \left(\frac{\partial h_k}{\partial P} \right)_{s_k} \left(\frac{\partial \rho_k}{\partial h_k} \right)_P + \left(\frac{\partial \rho_k}{\partial P} \right)_{h_k} &= \left(\frac{\partial \rho_k}{\partial P} \right)_{s_k} \end{aligned} \quad (22)$$

where the following relationships have been used:

$$\begin{aligned}\left(\frac{\partial x}{\partial y}\right)_z \left(\frac{\partial y}{\partial z}\right)_x \left(\frac{\partial z}{\partial x}\right)_y &= -1 \\ \left(\frac{\partial x}{\partial y}\right)_z &= \left(\frac{\partial x}{\partial y}\right)_t + \left(\frac{\partial t}{\partial y}\right)_z \left(\frac{\partial x}{\partial t}\right)_y \\ \rho_k &= \left(\frac{\partial P}{\partial h_k}\right)_{s_k}\end{aligned}$$

If the *isobaric expansion coefficient* is defined as

$$\pi_k \equiv \left(\frac{\partial \vartheta_k}{\partial h_k}\right)_P \quad (23)$$

and the *isentropic compressibility coefficient* as

$$\gamma_k \equiv \left(\frac{\partial \ln P}{\partial \ln \rho_k}\right)_{s_k} \quad (24)$$

the final form of the enthalpy conservation equation is obtained:

$$\begin{aligned}\alpha_k A \left[\frac{\partial \ln \rho_k}{\partial t} + v_k \frac{\partial \ln \rho_k}{\partial x} \right] &= \frac{\alpha_k}{\gamma_k} A \left[\frac{\partial \ln P}{\partial t} + v_k \frac{\partial \ln P}{\partial x} \right] \\ &- A \pi_k [Q_{wk} + Q_{ik} + \Gamma_{ik} h_k^* + \Gamma_{wk} h_k' - \Gamma_k h_k]\end{aligned} \quad (25)$$

Moreover, if the derivatives in the mass conservation (1) are expanded, and the whole equation is divided by ρ_k , we obtain

$$\frac{\partial}{\partial t}(\alpha_k A) + \alpha_k A \frac{\partial \ln \rho_k}{\partial t} + \frac{\partial}{\partial x}(\alpha_k A v_k) + \alpha_k A v_k \frac{\partial \ln \rho_k}{\partial x} = A \frac{\Gamma_k}{\rho_k} \quad (26)$$

The substitution of (26) into (25) and the integration along the x axis yields the combined pressure and level equation for phase k :

$$\frac{dV_k}{dt} + C_k \frac{d\langle \ln P \rangle_k}{dt} = J_k + Sw_k + D_k + Ac_k \quad (27)$$

The following variables have been defined in the previous equation:

- V_k is the volume occupied by the phase k ,
- $C_k \equiv \int_0^L A \frac{\alpha_k}{\gamma_k} dx$ is the phase k system compressibility,

- $\langle \ln P \rangle_k \equiv \frac{\int_0^L A \frac{\alpha_k}{\gamma_k} \ln P dx}{\int_0^L A \frac{\alpha_k}{\gamma_k} dx},$
- $J_k \equiv \alpha_k A v_k|_{\text{in}} - \alpha_k A v_k|_{\text{out}}$ is the difference between the inlet and outlet flows in the TH system,
- $Sw_k \equiv \int_0^L A \frac{\Gamma_k}{\rho_k^{\text{sat}}} dx$ is the expansion/contraction due to phase k appearance/dissappearance,
- $D_k \equiv \int_0^L A [\pi_k (Q_{wk} + Q_{ik} + \Gamma_{ik} h_k^* + \Gamma_{wk} h_k' - \Gamma_k h_k) + \Gamma_k (\vartheta_k - \vartheta_k^{\text{sat}})] dx$

is the expansion/contraction of phase k due to enthalpy variations,

- $Ac_k \equiv \int_0^L A \left[(\ln P - \langle \ln P \rangle_k) \frac{\partial}{\partial t} \left(\frac{\alpha_k}{\gamma_k} \right) - \frac{\alpha_k}{\gamma_k} v_k \frac{\partial \ln P}{\partial x} \right] dx$ is a term dominated by pressure gradients.

The relation

$$\int_0^L A \frac{\alpha_k}{\gamma_k} \left(\frac{\partial \ln P}{\partial t} + v_k \frac{\partial \ln P}{\partial x} \right) dx = C_k \frac{d \langle \ln P \rangle_k}{dt} - Ac_k \quad (28)$$

has been also taken into account.

If the two equations (27) are summed, one for each phase, the following pressure equation is obtained:

$$\frac{d \langle \ln P \rangle}{dt} = \frac{\sum_k J_k + Sw_k + D_k + Ac_k}{\sum_k C_k} \quad (29)$$

where

$$\frac{d \langle \ln P \rangle}{dt} \equiv \frac{\sum_k C_k \frac{d \langle \ln P \rangle_k}{dt}}{\sum_k C_k} \quad (30)$$

Equation (29) shows that the pressure evolution in a TH system is the result of the accommodation of volume changes. The greater the compressibility of the system, the smaller the impact of those changes in the pressure. The sources of volume change are the inlet and outlet volumetric flows, the swelling of the phases, the thermal expansions and changes caused by the pressure distribution along the system.

Assuming that $\frac{d\langle \ln P \rangle_k}{dt} \approx \frac{d\langle \ln P \rangle}{dt}$, we obtain the following *level equation*:

$$\frac{dV_f}{dt} = J_f + Sw_f + D_f + Ac_f - C_f \frac{d\langle \ln P \rangle}{dt} \quad (31)$$

The equation above clearly shows that variations of the volume of liquid in the system, which is closely related to the level, can be caused by a mismatch between the inlet and outlet volumetric flows, by the appearance/dissapearance of liquid phase, by the thermal expansion of the liquid, by spatial variations of the pressure and, finally, by temporal variations of the global pressure. The greater the compressibility of the liquid, the greater the influence of the variations of the global pressure in the evolution of the level.

2.4.2 Discretized form

The terms in the right hand side of the pressure and level equation can be discretized as follows:

$$C_k \approx \sum_{l=1}^N \frac{\alpha_k^l}{\gamma_k^l} V^l \quad (32)$$

$$Sw_k \approx \sum_{l=1}^N V^l \frac{\Gamma_k^l}{\rho_k^{\text{sat}^l}} \quad (33)$$

$$D_k \approx \sum_{l=1}^N V^l \left[\pi_k^l (Q_{wk}^l + Q_{ik}^l + \Gamma_{ik}^l h_k^{*l} + \Gamma_{wk}^l h_k^l - \Gamma_k^l h_k^l) + \Gamma_k^l (\vartheta_k^l - \vartheta_k^{\text{sat}^l}) \right] \quad (34)$$

$$Ac_k \approx \sum_{l=1}^N V^l (\ln P^l - \langle \ln P \rangle_k) \frac{\partial}{\partial t} \left(\frac{\alpha_k^l}{\gamma_k^l} \right) + \sum_{l=1}^N A^l \frac{\alpha_k^l}{\gamma_k^l} v_k^l \ln \left(\frac{P^{\text{in}^l}}{P^{\text{out}^l}} \right) \quad (35)$$

The volumetric flow J_k is computed with the equations (9) and (11).

The partial derivative with respect to time in equation (35) is estimated by finite differences:

$$\frac{\partial}{\partial t} \left(\frac{\alpha_k^l}{\gamma_k^l} \right) \approx \frac{1}{\Delta t} \left(\frac{\alpha_k^{l,n}}{\gamma_k^{l,n}} - \frac{\alpha_k^{l,n-1}}{\gamma_k^{l,n-1}} \right) \quad (36)$$

where the superindices n and $n - 1$ stand for two successive time steps.

The right hand side of equation (29) is integrated with respect to time with a BABIECA module that implements the trapezoidal rule. The result is compared to

$$\langle \ln P \rangle_k \approx \frac{\sum_{l=1}^N V^l \frac{\alpha_k^l}{\gamma_k^l} \ln P^l}{\sum_{l=1}^N V^l \frac{\alpha_k^l}{\gamma_k^l}} \quad (37)$$

Similarly, the right hand side of equation (31) is numerically integrated and compared with

$$V_f \approx \sum_{l=1}^N \alpha_f^l V^l \quad (38)$$

2.5 Non-equilibrium mass equation

This integral balance, derived also from the mass and energy equations, has been developed with two purposes:

- Provide an alternative way to check the consistency of the solution with the conservation equations.
- Under certain hipoteses, a form of the conservation equations that shows the tendence of two-phase systems towards thermal equilibrium can be obtained. This tendence is specially strong in the emerging phase, even if the constitutive equations do not indicate this behavior in principle. If this tendence towards equilibrium in the emerging phase is confirmed, simplified models can take advantage of that feature. This assumption is used, for instance, in the HIPA code [108], which leads to more simplified models. Checking of the equilibrium hipotesis in the emerging phase in a wide variety of transients would confirm the validity of those simplified models, in which the two phases are decoupled from each other. More efficient numerical schemes can then be devised.

2.5.1 Analytical derivation

The continuity equation can be rewritten as follows:

$$\begin{aligned} \frac{\partial}{\partial t}(\alpha_k A \rho_k) + \frac{\partial}{\partial x}(\alpha_k A \rho_k v_k) &= \rho_k \left[\frac{\partial}{\partial t}(\alpha_k A) + \frac{\partial}{\partial x}(\alpha_k A v_k) \right] \\ &+ \alpha_k A \left[\frac{\partial \rho_k}{\partial t} + v_k \frac{\partial \rho_k}{\partial x} \right] = A \Gamma_k \end{aligned} \quad (39)$$

Defining

$$\delta \rho_k^{\text{ng}} \equiv \rho_k^{\text{sat}} - \rho_k \quad (40)$$

$$A \Delta \Gamma_k \equiv \frac{\partial}{\partial t}(\alpha_k A \delta \rho_k^{\text{ng}}) + \frac{\partial}{\partial x}(\alpha_k A \delta \rho_k^{\text{ng}} v_k) \quad (41)$$

and taking into account the continuity equation (1), the saturation density fulfills the following conservation equation:

$$\begin{aligned} \frac{\partial}{\partial t}(\alpha_k A \rho_k^{\text{sat}}) + \frac{\partial}{\partial x}(\alpha_k A \rho_k^{\text{sat}} v_k) &= \frac{\partial}{\partial t}(\alpha_k A \delta \rho_k^{\text{ng}}) + \frac{\partial}{\partial x}(\alpha_k A \delta \rho_k^{\text{ng}} v_k) \\ &+ \frac{\partial}{\partial t}(\alpha_k A \rho_k) + \frac{\partial}{\partial x}(\alpha_k A \rho_k v_k) = A(\Gamma_k + \Delta \Gamma_k) \end{aligned} \quad (42)$$

If we expand the left hand side of the previous equation we get:

$$\rho_k^{\text{sat}} \left[\frac{\partial}{\partial t} (\alpha_k A) + \frac{\partial}{\partial x} (\alpha_k A v_k) \right] + \alpha_k A \left[\frac{\partial \rho_k^{\text{sat}}}{\partial t} + v_k \frac{\partial \rho_k^{\text{sat}}}{\partial x} \right] = A (\Gamma_k + \Delta \Gamma_k) \quad (43)$$

Subtracting equations (43) and (39), previously divided by ρ_k^{sat} and ρ_k , respectively, we obtain

$$\alpha_k A \left[\frac{\partial \ln \rho_k^{\text{sat}}}{\partial t} + v_k \frac{\partial \ln \rho_k^{\text{sat}}}{\partial x} - \frac{\partial \ln \rho_k}{\partial t} - v_k \frac{\partial \ln \rho_k}{\partial x} \right] = A \Gamma_k (\vartheta_k^{\text{sat}} - \vartheta_k) + A \frac{\Delta \Gamma_k}{\rho_k^{\text{sat}}} \quad (44)$$

If we define the compressibility along the saturation line as

$$\lambda_k^{\text{sat}} \equiv \frac{d \ln P}{d \ln \rho_k^{\text{sat}}} \quad (45)$$

and we use equations (25) and (41), we get

$$\frac{\partial}{\partial t} (\alpha_k A \delta \rho_k^{\text{nq}}) + \frac{\partial}{\partial x} (\alpha_k A \delta \rho_k^{\text{nq}} v_k) = \rho_k^{\text{sat}} \left[\left(\frac{1}{\lambda_k^{\text{sat}}} - \frac{1}{\gamma_k} \right) \alpha_k A \left(\frac{\partial \ln P}{\partial t} + v_k \frac{\partial \ln P}{\partial x} \right) + d_k \right] \quad (46)$$

where d_k is the integrand of the thermal expansion term, i.e., $d_k \equiv \frac{\partial D_k}{\partial x}$.

Defining

$$d_k^{\text{eq}} \equiv - \left(\frac{1}{\lambda_k^{\text{sat}}} - \frac{1}{\gamma_k} \right) \alpha_k A \left(\frac{\partial \ln P}{\partial t} + v_k \frac{\partial \ln P}{\partial x} \right) \quad (47)$$

and

$$d_k^{\text{nq}} \equiv d_k - d_k^{\text{eq}} \quad (48)$$

we obtain the following differential form of the non-equilibrium mass conservation equation:

$$\frac{\partial}{\partial t} (\alpha_k A \delta \rho_k^{\text{nq}}) + \frac{\partial}{\partial x} (\alpha_k A \delta \rho_k^{\text{nq}} v_k) = \rho_k^{\text{sat}} d_k^{\text{nq}} \quad (49)$$

Integrating along the x axis:

$$\frac{dm_k^{\text{nq}}}{dt} = \int_0^L \rho_k^{\text{sat}} d_k^{\text{nq}} dx + W_k^{\text{nq}}|_{\text{in}} - W_k^{\text{nq}}|_{\text{out}} \quad (50)$$

where the *non-equilibrium mass* m_k^{nq} has been defined as

$$m_k^{\text{nq}} \equiv \int_0^L \alpha_k A \delta \rho_k^{\text{nq}} dx \quad (51)$$

and the *non-equilibrium mass flow* as

$$W_k^{\text{nq}} \equiv \alpha_k A \delta \rho_k^{\text{nq}} v_k \quad (52)$$

The definitions (47) and (48) are physically meaningful. If phase k is in equilibrium, the left hand side of equation (49) vanishes, yielding

$$d_k = d_k^{\text{eq}} = - \left(\frac{1}{\lambda_k^{\text{sat}}} - \frac{1}{\gamma_k} \right) \alpha_k A \left(\frac{\partial \ln P}{\partial t} + v_k \frac{\partial \ln P}{\partial x} \right) \quad (53)$$

This is the reason for the definition of (47). The difference between the actual term d_k and that defined as the equilibrium term is just the definition (48).

When phase k is in thermal equilibrium, the terms that include any of the Γ variables in the expresion of d_k vanish. d_k becomes then

$$d_k^{\text{eq}} = A \pi_k (Q_{wk} + Q_{ik}^{\text{eq}}) \quad (54)$$

where Q_{ik}^{eq} is the interfacial heat in those conditions. The heat transferred to phase k is then donnored to the other phase through the interfacial surface, subtracting the heat necessary to accomodate the new saturation conditions imposed by the spatio-temporal variations of the pressure. The contribution of the pressure is accounted for with the term d_k^{eq} .

2.5.2 Discretized form

The integral in the right hand side of equation (50) is approximated, within a control volume l , as follows:

$$\begin{aligned} \int_0^{L^l} \rho_k^{\text{sat}} d_k^{\text{nq}} dx &\approx \\ \rho_k^{\text{sat}^l} D_k^l + \int_0^{L^l} \rho_k^{\text{sat}} \left(\frac{1}{\lambda_k^{\text{sat}}} - \frac{1}{\gamma_k} \right) \alpha_k A \left(\frac{\partial \ln P}{\partial t} + v_k \frac{\partial \ln P}{\partial x} \right) dx &\approx \\ \rho_k^{\text{sat}^l} \left[D_k^l + \gamma_k^l \left(\frac{1}{\lambda_k^{\text{sat}^l}} - \frac{1}{\gamma_k^l} \right) \int_0^{L^l} A \frac{\alpha_k}{\gamma_k} \left(\frac{\partial \ln P}{\partial t} + v_k \frac{\partial \ln P}{\partial x} \right) dx \right] &= \\ \rho_k^{\text{sat}^l} \left[D_k^l + \gamma_k^l \left(\frac{1}{\lambda_k^{\text{sat}^l}} - \frac{1}{\gamma_k^l} \right) \left(J_k^l + Sw_k^l + D_k^l - \frac{dV_k^l}{dt} \right) \right] & \end{aligned} \quad (55)$$

The relations (28) and (31) have been used in the last part of the previous equation.

The right hand side of the non-equilibrium mass equation (50) is approximated in a whole TH system by the sum of all the equations (55) that represent each control volume:

$$\begin{aligned} \int_0^L \rho_k^{\text{sat}} d_k^{\text{nq}} dx &\approx \sum_{l=1}^N \rho_k^{\text{sat}^l} \frac{\gamma_k^l}{\lambda_k^{\text{sat}^l}} \left[J_k^l + Sw_k^l + D_k^l - \frac{dV_k^l}{dt} \right] \\ &\quad - \sum_{l=1}^N \rho_k^{\text{sat}^l} \left[J_k^l + Sw_k^l - \frac{dV_k^l}{dt} \right] \\ &\quad + W_k^{\text{nq}}|_{\text{in}} - W_k^{\text{nq}}|_{\text{out}} \end{aligned} \quad (56)$$

The saturation density in the definition of the non-equilibrium mass flows is that corresponding to the inlet (or outlet) pressure of the system. The equation (56), integrated with respect to time, is compared then with the following approximation of the non-equilibrium mass of phase k in the whole system:

$$m_k^{\text{nq}} \approx \sum_{l=1}^N V_k^l \left(\rho_k^{\text{sat}^l} - \rho_k^l \right) \quad (57)$$

The comparison of the two sides of equation (50) can be used to check the consistency of the solutions provided by the RELAP5 code, as already declared at the beginning of this section.

2.5.3 Uncoupling of the non-equilibrium mass equations

The thermal expansion term d_k in the non-equilibrium mass equation can be simplified under certain hypotheses, leading to the uncoupling of the two phases. If those hypotheses are confirmed, the uncoupled equations could be used as the basis for more efficient numerical schemes.

The term d_k can be rewritten as follows:

$$\begin{aligned} d_k = & A \left[\pi_k (Q_{wk} + Q_{ik} + \Gamma_{ik} h_k^* + \Gamma_{wk} h_k' - \Gamma_k h_k) + \Gamma_k (\vartheta_k - \vartheta_k^{\text{sat}}) \right] = \\ & A \pi_k (Q_{wk} + Q_{ik}^w + Q_{ik}^b) + A \Gamma_{ik} [\pi_k (h_k^* - h_k) + (\vartheta_k - \vartheta_k^{\text{sat}})] \\ & + A \Gamma_{wk} [\pi_k (h_k' - h_k) + (\vartheta_k - \vartheta_k^{\text{sat}})] \end{aligned} \quad (58)$$

where both the interfacial heat and the generation rate of phase k have been split into their contributions in the wall and the bulk of the fluid, according to the equations

$$Q_{ik} = Q_{ik}^w + Q_{ik}^b \quad (59)$$

$$\Gamma_k = \Gamma_{wk} + \Gamma_{ik} \quad (60)$$

If $\Gamma_{ik} < 0$ then $h_k^* = h_k$ (see the definition of h_k^* in section 2.6), and the term with Γ_{ik} in the left hand side of equation (58) remains in the form

$$A\Gamma_{ik} [\pi_k (h_k^* - h_k) + (\vartheta_k - \vartheta_k^{\text{sat}})] = A\Gamma_{ik} (\vartheta_k - \vartheta_k^{\text{sat}}) \quad (61)$$

In this case the two phases cannot be uncoupled, since the expresion of the generation rate of vapor in the bulk

$$\Gamma_{ik} = -\frac{H_{ik}(T^{\text{sat}} - T_k) + H_{ik'}(T^{\text{sat}} - T_{k'})}{h_k^* - h_{k'}} \quad (62)$$

contains variables relative to the other phase.

On the other hand, if $\Gamma_{ik} > 0$ then $h_k^* = h_k^{\text{sat}}$, and the term with Γ_{ik} in the left hand side of equation (58) takes the form

$$\begin{aligned} A\Gamma_{ik} [\pi_k (h_k^* - h_k) + (\vartheta_k - \vartheta_k^{\text{sat}})] = \\ A\Gamma_{ik} [\pi_k (h_k^{\text{sat}} - h_k) + (\vartheta_k - \vartheta_k^{\text{sat}})] \end{aligned} \quad (63)$$

If the specific volumes are expanded in a Taylor series:

$$\vartheta_k^{\text{sat}} = \vartheta_k + \left(\frac{\partial \vartheta_k}{\partial h_k} \right)_P (h_k^{\text{sat}} - h_k) + \frac{1}{2} \left(\frac{\partial^2 \vartheta_k}{\partial h_k^2} \right)_P (h_k^{\text{sat}} - h_k)^2 + \dots \quad (64)$$

Neglecting the second and higher order terms in equation (64), and taking into account the definition of π_k , equation (63) becomes trivial, simplifying the non-equilibrium mass equation.

Moreover, if $\Gamma_{wk} > 0$ then $h'_k = h_k^{\text{sat}}$ and the term Γ_{wk} in the right hand side of equation (58) can also be neglected. In this case the non-equilibrium mass equation remains as follows:

$$\begin{aligned} \frac{\partial}{\partial t} (\alpha_k A \delta \rho_k^{\text{ng}}) + \frac{\partial}{\partial x} (\alpha_k A \delta \rho_k^{\text{ng}} v_k) = \\ \rho_k^{\text{sat}} A \pi_k (Q_{wk} + Q_{ik}^w + Q_{ik}^b) - \rho_k^{\text{sat}} d_k^{\text{eq}} \end{aligned} \quad (65)$$

Since we assume that $\Gamma_{wk} > 0$, this phase appears in equilibrium in the wall, and hence $Q_{ik}^w = 0$. The interfacial heat in the bulk is defined by the expresion:

$$Q_{ik}^b = H_{ik}(T^{\text{sat}} - T_k) \quad (66)$$

Expanding the previous equation by a Taylor series up to the first order:

$$Q_{ik}^b \approx H_{ik} \delta \rho_k^{\text{ng}} \left(\frac{\partial T}{\partial \rho} \right)_P^{\text{sat}} = -\frac{H_{ik}}{\rho_k^{\text{sat}} \beta_k^{\text{sat}}} \delta \rho_k^{\text{ng}} \quad (67)$$

The final form of the non-equilibrium mass equation, assuming that $\Gamma_{ik} > 0$ y $\Gamma_{wk} > 0$, is:

$$\begin{aligned} \frac{\partial}{\partial t} (\alpha_k A \delta \rho_k^{\text{ng}}) + \frac{\partial}{\partial x} (\alpha_k A \delta \rho_k^{\text{ng}} v_k) \approx & -\frac{H_{ik}}{\beta_k^{\text{sat}}} \pi_k A \delta \rho_k^{\text{ng}} + \rho_k^{\text{sat}} A \pi_k Q_{wk} \\ & + \rho_k^{\text{sat}} \left(\frac{1}{\lambda_k^{\text{sat}}} - \frac{1}{\gamma_k} \right) \alpha_k A \left(\frac{\partial \ln P}{\partial t} + v_k \frac{\partial \ln P}{\partial x} \right) \end{aligned} \quad (68)$$

where all the variables are defined for the phase k itself, demonstrating the uncoupling from the other phase.

Let's remind that the validity of this model is guaranteed only under the following assumptions:

- $\Gamma_{ik} [\pi_k (h_k^{\text{sat}} - h_k) + (\vartheta_k - \vartheta_k^{\text{sat}})] \approx 0$
- $\Gamma_{wk} [\pi_k (h_k^{\text{sat}} - h_k) + (\vartheta_k - \vartheta_k^{\text{sat}})] \approx 0$
- $T^{\text{sat}} - T_k \approx -\frac{\delta \rho_k^{\text{ng}}}{\rho_k^{\text{sat}} \beta_k^{\text{sat}}}$

On the other hand, if $\Gamma_{wk} < 0$ then $h'_k = h_k$ and the terms that involve the interfacial heat in the wall and the generation rate of phase k in the wall become

$$\begin{aligned} A [\pi_k Q_{ik}^w + \Gamma_{wk} (\vartheta_k - \vartheta_k^{\text{sat}})] = & A \Gamma_{wk} ((\vartheta_k - \vartheta_k^{\text{sat}}) - \pi_k (h'_k - h'_{k'})) = \\ & A \Gamma_{wk} ((\vartheta_k - \vartheta_k^{\text{sat}}) - \pi_k (h_k - h_k^{\text{sat}})) \end{aligned} \quad (69)$$

where the following relation taken from the page 4-130 of reference [97] has been used :

$$\Gamma_{wk} = -\frac{Q_{ik}^w}{h'_k - h'_{k'}} \quad (70)$$

The reference [97] also suggests on page 4-132 the following expressions for the generation rate of vapor and liquid in the wall:

$$\Gamma_{wg} = \frac{Q_{wg}}{\max(h_g - h_f^{\text{sat}}, 10^4 J/Kg)} \quad (\text{wall condensation}) \quad (71)$$

$$\Gamma_{wf} = -\frac{Q_{wf}}{\max(h_g^{\text{sat}} - h_f, 10^4 J/Kg)} \text{Mul} \quad (\text{wall boiling}) \quad (72)$$

Assuming that $|h_k - h_{k'}^{\text{sat}}| > 10^4 J/Kg$, and taking into account that $\vartheta_k - \vartheta_k^{\text{sat}} \approx \frac{\delta \rho_k^{\text{ng}}}{\rho_k^{\text{sat}}}$, the following expression of the non-equilibrium mass equation is obtained:

$$\begin{aligned} \frac{\partial}{\partial t} (\alpha_k A \delta \rho_k^{\text{ng}}) + \frac{\partial}{\partial x} (\alpha_k A \delta \rho_k^{\text{ng}} v_k) \approx & \left[\frac{Q_{wk} \text{Mul}}{\rho_k^{\text{sat}} (h_k - h_{k'}^{\text{sat}})} - \pi_k \frac{H_{ik}}{\beta_k^{\text{sat}}} \right] A \delta \rho_k^{\text{ng}} \\ & + \rho_k^{\text{sat}} A \pi_k Q_{wk} (1 - \text{Mul}) \\ & + \rho_k^{\text{sat}} \left(\frac{1}{\lambda_k^{\text{sat}}} - \frac{1}{\gamma_k} \right) \alpha_k A \left(\frac{\partial \ln P}{\partial t} + v_k \frac{\partial \ln P}{\partial x} \right) \end{aligned} \quad (73)$$

Like in the case in which $\Gamma_{wk} > 0$, the previous equation is decoupled from the other phase, since h_k^{sat} depends only on the pressure. The multiplier Mul depends only on the pressure as well. This multiplier equals 1 for the vapor phase. In this case the first term with the variable Q_{wk} in equation (73) vanishes.

It is convenient to remind here that these conclusions are valid only for the phase in which $\Gamma_{ik} > 0$ and $\Gamma_{wk} < 0$, under the following assumptions:

- $\Gamma_{ik} [\pi_k (h_k^{\text{sat}} - h_k) + (\vartheta_k - \vartheta_k^{\text{sat}})] \approx 0$
- $T^{\text{sat}} - T_k \approx -\frac{\delta \rho_k^{\text{ng}}}{\rho_k \beta_k}$
- $\vartheta_k - \vartheta_k^{\text{sat}} \approx \frac{\delta l^{\text{ng}}}{\rho_k^{\text{sat}2}}$

Confirmation of these assumptions in a wide range of transients will allow to validate the new model based on the non-equilibrium mass equations.

2.6 Correspondence with the RELAP5 output variables

To evaluate the integral balances described in the sections 2.2 to 2.5 it is necessary to associate the variables involved in those balances with the output variables provided by the RELAP5 output files. The partial derivatives are obtained from the three partial derivatives obtained in RELAP5, i.e.:

- $C_P \equiv \left(\frac{\partial h}{\partial T}\right)_P$
- $\beta \equiv \frac{1}{\vartheta} \left(\frac{\partial \vartheta}{\partial T}\right)_P$
- $\eta \equiv \left(\frac{\partial \rho}{\partial P}\right)_u$

On the contrary, the partial derivatives available in the BABIECA module **r5prop**, needed to evaluate the properties along the saturation line, are the following:

- $C_P \equiv \left(\frac{\partial h}{\partial T}\right)_P$
- $C_\vartheta \equiv \left(\frac{\partial u}{\partial T}\right)_\vartheta$
- $\beta \equiv \frac{1}{\vartheta} \left(\frac{\partial \vartheta}{\partial T}\right)_P$

Several derivatives of thermodynamic properties, computed as function of the two groups on variables above, are shown in Appendix 6.3.

The correspondences are shown below, with the RELAP5 variables on the right hand side of the equivalences.

Variables concerning control volumes

$$C_{Pg}^l \equiv \text{csubpg}$$

$$C_{Pf}^l \equiv \text{csubpf}$$

$$\beta_g^l \equiv \text{betagg}$$

$$\beta_f^l \equiv \text{betaff}$$

$$\eta_g^l \equiv \text{drgdp}$$

$$\eta_f^l \equiv \text{drfdp}$$

$$V^l \equiv \text{vvol}$$

$$A^l \equiv \text{avol}$$

$$P^l \equiv \text{p}$$

$$\alpha_g^l \equiv \text{voidg}$$

$$\alpha_f^l \equiv \text{voidf}$$

$$\rho_g^l \equiv \text{rhog}$$

$$\rho_f^l \equiv \text{rhof}$$

$$\vartheta_g^l \equiv \frac{1}{\text{rhog}}$$

$$\vartheta_f^l \equiv \frac{1}{\text{rhof}}$$

$$u_g^l \equiv \text{ug}$$

$$u_f^l \equiv \text{uf}$$

$$v_g^l \equiv \text{velg}$$

$$v_f^l \equiv \text{velf}$$

$$\Gamma_{ig}^l \equiv \text{gammai}$$

$$\Gamma_{if}^l \equiv -\text{gammai}$$

$$\Gamma_{wg}^l \equiv \text{gammaw}$$

$$\Gamma_{wf}^l \equiv -\text{gammaw}$$

$$\Gamma_g^l \equiv \text{gammaw} + \text{gammai}$$

$$\Gamma_f^l \equiv -(\text{gammaw} + \text{gammai})$$

$$h_g^l \equiv \text{ug} + \frac{\text{p}}{\text{rhog}}$$

$$h_f^l \equiv \text{uf} + \frac{\text{p}}{\text{rhof}}$$

$$h_g^{*l} \equiv \begin{cases} h_g^{\text{sat}^l}(\text{p}) & \text{if } \text{gammai} > 0 \\ h_g^l & \text{if } \text{gammai} < 0 \end{cases}$$

$$h_f^{*l} \equiv \begin{cases} h_f^{\text{sat}^l}(\text{p}) & \text{if } \text{gammai} < 0 \\ h_f^l & \text{if } \text{gammai} > 0 \end{cases}$$

$$h_g'^l \equiv \begin{cases} h_g^{\text{sat}^l}(\text{p}) & \text{if } \text{gammaw} > 0 \\ h_g^l & \text{if } \text{gammaw} < 0 \end{cases}$$

$$h_f'^l \equiv \begin{cases} h_f^{\text{sat}^l}(\text{p}) & \text{if } \text{gammaw} < 0 \\ h_f^l & \text{if } \text{gammaw} > 0 \end{cases}$$

$$H_{ig}^l \equiv \text{hig}$$

$$H_{if}^l \equiv \text{hif}$$

$$T^{\text{sat}^l} \equiv \text{satt}$$

$$T_g^l \equiv \text{tempg}$$

$$T_f^l \equiv \text{tempf}$$

$$Q_{wg}^l \equiv \frac{\text{qwg}}{\text{vvol}}$$

$$Q_{wf}^l \equiv \frac{\text{q} - \text{qwg}}{\text{vvol}}$$

$$Q_{ig}^{wl} \equiv \begin{cases} -\text{gammaw}(h_g'^l - h_f'^l) & \text{if } \text{gammaw} < 0 \\ 0 & \text{if } \text{gammaw} > 0 \end{cases}$$

$$Q_{if}^{wl} \equiv \begin{cases} -\text{gammaw}(h_g'^l - h_f'^l) & \text{if } \text{gammaw} > 0 \\ 0 & \text{if } \text{gammaw} < 0 \end{cases}$$

$$Q_{ig}^{bl} \equiv \text{hig}(\text{satttemp} - \text{tempg})$$

$$Q_{if}^{bl} \equiv \text{hif}(\text{satttemp} - \text{tempf})$$

$$\text{FW}_g^l \equiv \text{voidg} * \text{rhog} * \text{fwalg}$$

$$\text{FW}_f^l \equiv \text{voidf} * \text{rhof} * \text{fwalf}$$

Variables concerning junctions

$$v_g^j \equiv \text{velgj}$$

$$v_f^j \equiv \text{velfj}$$

$$\alpha_g^j \equiv \text{voidgj}$$

$$\alpha_f^j \equiv \text{voidfj}$$

$$\rho_g^j \equiv \text{rhogj}$$

$$\rho_f^j \equiv \text{rhofj}$$

$$A^j \equiv \text{ajun}$$

$$K_f^j \equiv \text{formg}j$$

$$K_r^j \equiv \text{formf}j$$

3 DESCRIPTION OF THE ANALYSIS TOOL

3.1 Introduction

This section describes the codes and computer programs used to build the proposed analysis tool and the way they are connected to perform the in-depth postprocess of the results obtained in RELAP5 calculations.

3.2 Description of the programs used in the analysis tool

3.2.1 The RELAP5 code

RELAP5 is the thermalhydraulic code whose results will be checked with the analysis tool. Thus, it is the central code of it. Please refer to [95] for details on the code.

3.2.2 The BABIECA simulation language

BABIECA is the driver of the continuous, general purpose simulation language integrated in the package CAMPEADOR [49]. BABIECA will be the core of the analysis tool. It is the central program to which the rest of the codes will be connected. It performs the additional calculations described in section 2.

The BABIECA simulation language is modular, which means that the problem to be solved is defined as a block diagram. Each block may be considered as a multi input-multi output relation between time-evolving variables, and represents a physical system or a mathematical procedure. The input-output relation consists of a set of differential equations plus a numerical solution algorithm (implicit). There is no common solution algorithm for all the blocks, as it is the norm in most simulation languages. Blocks are particular instances of computational entities called modules. The set of private data of the block defines it, while it is the connection among the blocks and the overall computation order that constitute the global numerical scheme. Each block has a number, defined by the user in the input file. The block outputs are numbered successively starting from the block number. The user must take care of avoiding output number overlapping. The driver routine of BABIECA manages the time step control and calls the modules sequentially in the user-defined order. When all the blocks have been successfully computed the time step is considered to be finished, and a new time step calculation begins.

Modules cover a broad range of mathematical tools, physical models and special components, and are mainly oriented, but not limited, to the solution of large thermal hydraulic networks like those appearing in nuclear power plants. Among all the modules available, only few will be used in this tool. The most important one is the module called FUNIN, which is a parser that processes, at each time step, a FORTRAN-like mathematical expression dependent upon the block inputs. Other important modules are LOGATE, which obtains logical variables from

analogical inputs, **R5STEAM**, which contains the RELAP5 water properties tables, **FINT**, which implements time dependent functions, **WRITES**, used to write column formatted output files and finally **SNDPCODE** and **RCVPCODE**, used for the connection with other codes.

3.2.3 Parallel Virtual Machine (PVM)

Parallel Virtual Machine (PVM) [27, 28] is a software for the parallelization and exchange of information between different computer processes. It runs on top of the UNIX sockets. PVM is interoperable, i.e., the computer processes to be connected can run even on different machines of different nature. It is only required that the machines be connected through the TCP/IP protocol.

PVM comprises a library of C and FORTRAN functions and a daemon program called **pvmd**. The library provides functions to spawn a new process from a running one, to pack data in buffers, to send these buffers, to receive them and to unpack the data. These functions, when properly inserted in the source code of the original programs, allow the developers to make the programs work concurrently to perform an overall task.

It is convenient here to make a reference to the Message Passing Interface (MPI) software [32]. It is more oriented towards parallelization of codes running in multiprocessor machines than towards distributed computing. It only can work between machines of the same type. Another drawback is the incapability to spawn processes. On the other hand, it shows a better performance in the exchange of information between processes. The University of Tennessee and the Oak Ridge National Laboratory are planning to develop a product that would support all the capabilities of both PVM and MPI, called PVMPI [23]. If this product is finally released, it is almost sure that it will become the new standard for parallelization and connection of programs.

3.2.4 The text editor AWK

AWK is a non-interactive text editor included in all standard distributions of UNIX-like operating systems. Details and reference manual can be found in any UNIX reference manual. Generally speaking, AWK is a C-like interpreted language that allows users to use and operate variables *on the fly* without previous definition. The AWK command are usually written in a file. These commands read information from a set of files, process the information, and generate another set of files. The file containing the AWK commands comprises three parts:

- A section called **BEGIN**, where the user can initialize variables and define parameters used along the rest of the file.
- A main body, in which AWK searches for lines containing given patterns in a set of files specified by the user. Depending on the matching of these patterns, and the information contained in the files, AWK can generate new information, in form of new variables, operations between variables, etc.

- An **END** section, in which AWK can generate new files depending on the information generated in the main body.

Due to its special features, AWK is specially useful for automatic generation of particular purpose input files of general purpose programs.

3.2.5 XMGR5

XMGR [101] is a program for the treatment and graphical display of scientific data. The XMGR5 program [52] is an extension of the previous one, which is capable of reading the information stored in the RELAP5 restart-plot files. In addition, XMGR5 can access the RELAP5 water property tables. For these reasons, XMGR5 has been chosen to display the results of the RELAP5 in-depth postprocessing.

3.3 Linkage of the codes

In this section we describe how the previous codes are linked to build the postprocessing tool.

3.3.1 The BABIECA-RELAP5 coupling

BABIECA is the central program of the postprocessing tool. It will be connected to RELAP5 via the PVM library functions, allowing the exchange of information between them. Output information from RELAP5 will be transferred to properly built BABIECA blocks in order to perform additional verifications of the RELAP5 calculations.

The coupling between BABIECA and RELAP5 was established in a general way that could be used to couple any other code. This may allow to postprocess other codes in the future, aside from RELAP5, or just to run combined simulations where each code is responsible for the calculation of certain phenomena. The first step towards generalization of code coupling was the definition of a standard specifying how a scientific code must be written to facilitate the connection to a general purpose modular simulation language such as BABIECA. A full description of this standard will be the object of another report. A description of the code connection philosophy can be found in [38]. However, some general ideas will be given here.

The proposed standard assumes that the codes will be coupled to a modular simulation language with capabilities similar to those of BABIECA. The connected codes will behave then as blocks of the modular simulation language. The user can now connect the inputs and outputs of the codes by defining a connection topology in the modular language.

Instead of writing a different module for each code coupled to the system, the standard establishes that the modular language will link to any other code by means of two modules called respectively **sndcode** and **rcvcode**. When using the module **sndcode**, the user must specify

in the BABIECA input file the code to be used, with the proper command line including options, and the names of the variables that will be modified externally by BABIECA. These variables can be boundary or initial conditions. The `sndcode` spawns the remote code, using the command line defined in the input file, and sends it the names of the boundary and initial conditions that will be supplied by BABIECA. At each time step, BABIECA also sends the remote code the value of the step, the values of the boundary conditions for the current time step, and the values of the initial conditions if some flag is activated. `rcvcode` is the counterpart of `sndcode`. The user must specify in the BABIECA input file the names of the variables whose values will be received from the remote file at the end of each time step. `rcvcode` receives from the corresponding `sndcode` the process identification number of the remote code, provided by PVM at the spawning time.

The standard also determines how the codes to be coupled must be written in order to facilitate a proper inclusion of PVM functions, that will allow a synchronized information exchange with the modules `sndcode` and `rcvcode`. The PVM library functions will be executed only if the code detects that it has a parent process, and hence it has been spawned by BABIECA. This allows to use the same executable file of the code for coupling with BABIECA and for standalone calculations. After being spawned, and after reading the corresponding input file, the remote code will receive from `sndcode` the names of the boundary and initial conditions to be transferred by BABIECA. These names must be in agreement with a established nomenclature, which is code dependent. With these names the coupled code must be capable of finding a pointer to the values of the corresponding boundary and initial conditions. The code also receives from `rcvcode` the names of the output variables to be sent to BABIECA, in order to obtain pointers to the memory allocation of those variables. At each time step, the code receives from `sndcode` the time step size and the values of the boundary conditions. Initial conditions are also received if necessary. The received values are used to overwrite those computed from the code input file and stored in the memory positions pointed by the pointers mentioned above. Once the solution is advanced one time step, the code sends to `rcvcode` the values of the demanded outputs.

Although the RELAP5 code does not comply with the standard, PVM routines have been properly inserted into the source code to make it behave as if it met the requirements. Not all the capabilities specified in the standard have been implemented in the RELAP5-BABIECA coupling. For instance, only boundary conditions specified by means of time dependent volume or junctions can be overwritten. No initial conditions can be supplied, since this feature is not necessary for postprocessing purposes.

3.3.2 The role of the AWK editor

The non-interactive text editor AWK is used to build the BABIECA input file necessary to check the results of a given RELAP5 case and once the user has specified the control volumes to which the checks must be applied. The checks are hence written in a file called `babpost.awk`, which contains AWK commands. The file is listed in appendix 6.3. With this

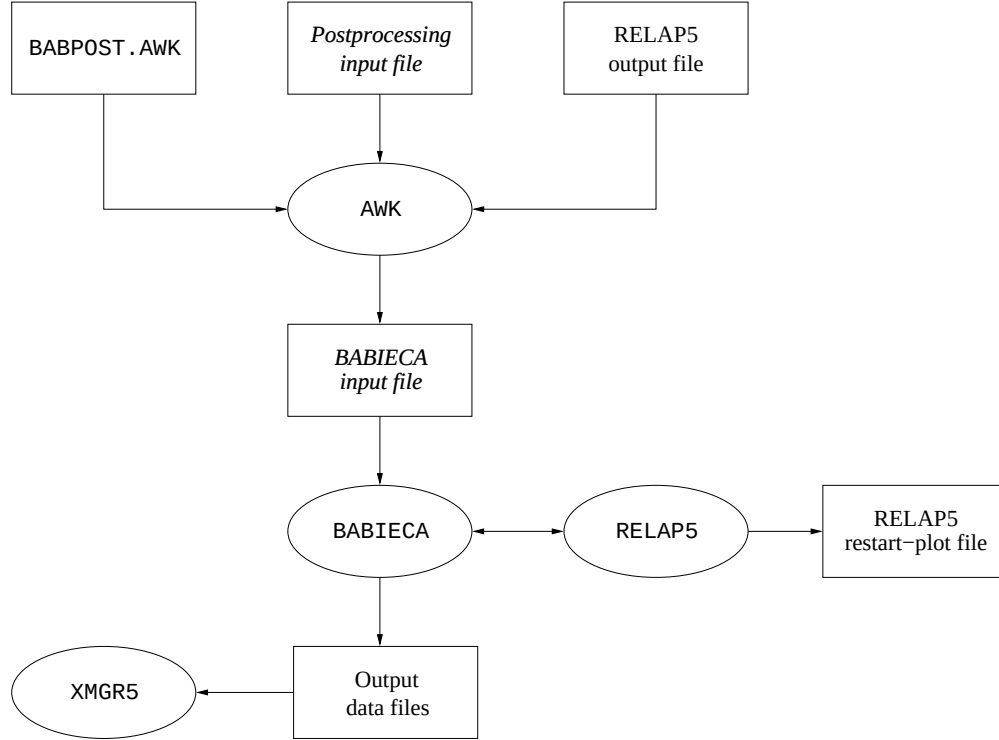


Figure 1: Functioning diagram of the analysis tool.

file, and depending on the particular RELAP5 calculation and the volumes to be checked, the AWK editor builds the corresponding BABIECA input file.

3.3.3 The role of the XMGR5 program

XMGR5 is just used to graphically display the RELAP5 and the postprocessing results after the end of the calculations. The tool takes advantage of the XMGR5 capabilities to read RELAP5 restart-plot files.

3.4 Functioning of the analysis tool

The analysis tool described in this report works as depicted in figure 1.

First of all, the user must write an ASCII file containing the following information in successive lines:

```

FILE = rootword
HOST = host
INITTIME = inittime
STARTTIME = starttime
ENDTIME = endtime
FREQ = freq
NSYSTEMS = nsystems
For each thermalhydraulic subsystem the file must contain also:
SYSNAME = sysname
NVOLS = nvols
cccvv0000

```

The words in typewriter type must be written literally. The words in italics have the following meaning:

- *rootword* is the root name of the RELAP5 case to be checked. It is assumed that the input file has the extension *.i* appended to the root name. The RELAP5 output file will be called *rootword.o* and the restart-plot file will be *rootword.r*.
- *host* is the name of the host where the RELAP5 executable file is allocated.
- *inittime* is a real number specifying the transient instant, in seconds, at which RELAP5 will be spawned by BABIECA. It is used to synchronize the times of BABIECA and RELAP5, specially when the RELAP5 calculation is a restart case.
- *starttime* is a real number specifying the transient instant, in seconds, at which the checks begin.
- *endtime* is a real number specifying the transient instant, in seconds, at which the checks end.
- *freq* is an integer number specifying the frequency used to write output data, in time steps.
- *nsystems* is the number of thermalhydraulic subsystems to be checked. A TH subsystem is a set of RELAP5 control volumes defined by the user. They may be connected or not.
- *sysname* is a name given to each TH subsystem. The results of the verifications in each subsystem will be written in a file called *rootword_sysname.dat*.
- *nvols* is the number of control volumes in the current TH subsystem.
- *cccvv0000* is a nine digit string denoting each control volume, according to the RELAP5 nomenclature. The three digit substring *ccc* refers to the RELAP5 hydrodynamic

component and the two digit substring *vv* refers to the number of the volume within the component (01 for a single volume). The last four digits are always 0000.

In the following lines an example of a file of this type is shown:

```
FILE = liq_pres
HOST = spp2k
INITTIME = 0.0
STARTTIME = 200.0
ENDTIME = 400.0
FREQ = 5
NSYSTEMS = 1
SYSNAME = mainpipe
NVOLS = 10
404010000
404020000
404030000
404040000
404050000
404060000
404070000
404080000
404090000
404100000
```

After writing a file like this, the user must execute from the UNIX command prompt the following sentence:

```
awk -f babpost.awk inpfile rootword.o > babfile.bab
```

The previous line performs the following actions: the AWK editor reads the file of AWK commands contained in the file ***babpost.awk*** and looks for the information in the file *inpfile*. Once this information has been read, and AWK knows about the control volumes to be checked, the AWK editor looks into the RELAP5 output file *rootword.o* to identify the junctions connecting the control volumes to be checked. This file must be available by running the RELAP5 input file *rootword.i*, preferably in check mode. Finally, AWK writes the BABIECA input file *babfile.bab*, which performs the ckeking of the RELAP5 calculation.

The following step consists in running PVM with a configuration file specifying the hosts that will constitute the virtual machines, executable paths, working directories, etc. Here is an example of such a file:

```
# Configuration file for pvm3
#
spp2k dx=/opt/pvm3/lib/pvmd
      ep=$HOME/bin:$HOME/relap/pvmcomp/selap
      bx=/users/rhs/casos/testpvm/debugger
      wd=$HOME/relap/postcases/liq_pres
```

The PVM daemon is activated just by typing

`pvm pvm_conf_file`

from the command prompt, where *pvm_conf_file* is the configuration file.

The postprocess of the RELAP5 case is then executed by typing

`BABIECA -n babfile.bab`

BABIECA spawns then a RELAP5 process and sets the communication between them. After finishing a time step calculation, RELAP5 sends the required variables to BABIECA. The latter computes then the proper verifications. Once the calculation is finished, PVM must be deactivated. The results of each TH subsystem are stored in files named *root-word_sysname.dat*. The results concerning individual volumes are stored in files *root-word_cccv.dat*. The variables in these files can be displayed with XMGR5.

4 APPLICATION EXAMPLES

4.1 Introduction

This section illustrates the capabilities of the analysis tool by means of some application examples.

Example 4.2 reveals an error in the input file due to the user. Section 4.3 is a good example of interpretation of the results computed by the code, which demonstrates a fake argument commonly used to explain the different depressurization rates obtained when a breach appears in the vapor or liquid region of a hydraulic system. The same example is used to verify the non-equilibrium model derived in section 2.5. Example 4.4 points out the boundary condition overspecification problem in several TH codes. Finally, application example 4.5 is devoted to other problem usually found in TH codes: flow mixing and splitting in converging pipes.

4.2 Compression caused by heating in a closed pipe

4.2.1 Example description

This example consists in an 18 node horizontal pipe. Although all the nodes are equal, it is convenient to discriminate between the ten central nodes and the two groups of 4 nodes each one located at the tips of the pipe. The noding is shown in figure 2. The pipe is closed in both sides, and initially filled with liquid and vapor in equilibrium conditions, in a horizontally stratified regime. The pressure is similar to that of the primary circuit in a PWR. After running a null transient, to let the system reach a steady state, a heat power is introduced in each of the ten central nodes. The total power is represented in figure 3.

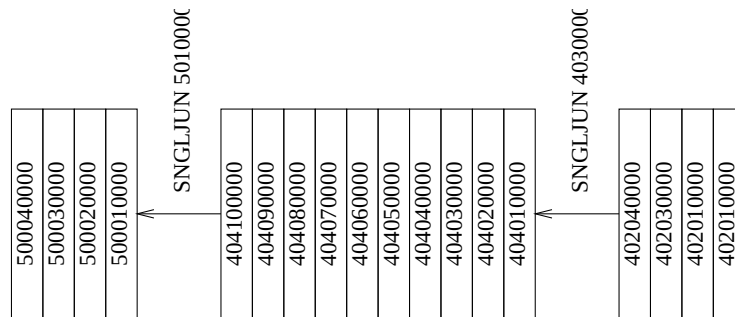


Figure 2: Nodalization used in the example.

As a result of the heating, and being the pipe closed in both sides, we expect the pressure to rise, along with the void fraction. Since the problem shows a clear spatial symmetry, symmetric results are so expected.

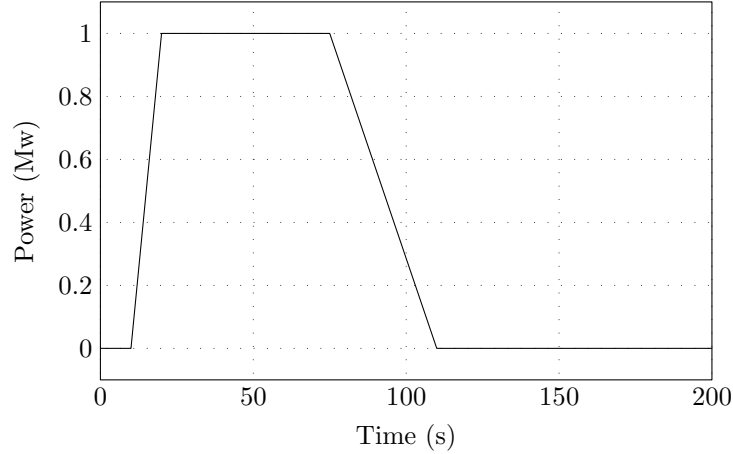


Figure 3: Power introduced in the pipe.

4.2.2 Analysis of results

Figure 4 shows the two sides of the pressure equation (29), in the TH subsystem defined by the ten central nodes that constitute the pipe 402. Both sides of the equation are identical. No anomaly in the solution can be inferred from this results. The figure also shows the expected increase of the system pressure.

The comparison of the two sides of the level equation is fully satisfactory either, according to figure 5. Nevertheless, an increase of the liquid level in the system is observed, contrary to what is expected. This observation suggest an error in the simulation of the example.

Figure 6 shows the expected increase of the vapor mass, even though great discrepancies between the two sides of the equation can be noticed. Mass errors in the vapor phase are very frequently observed in RELAP5 calculations. The mass error is negligible in the liquid phase, represented in figure 7. This figure also shows, contrary to what is expected, an increase of the liquid mass.

The two sides of the non-equilibrium mass equation, for the vapor phase, shown in figure 8, are not coincidental because of the mistmach of the vapor mass. On the other hand, the verification of the liquid non-equilibrium mass equation is satisfactory, as shown in figure 9.

The terms taking part in the pressure and level equations are displayed in figure 10. The only non-zero terms are the volumetric ones, which have surprisingly the same shape in both phases.

Figures 11 to 17 illustrate the same analysis in pipe 500. The postprocessing of the pressure equation leads to results identical to those obtained in the pipe 402. However, the liquid volume decreases during the transient, unlike in pipe 402. The simmetry of the problems does not seem to allow a result like this, reenforcing the suspects of an error in the solution. The two sides of the vapor mass equation are quite diferent either.

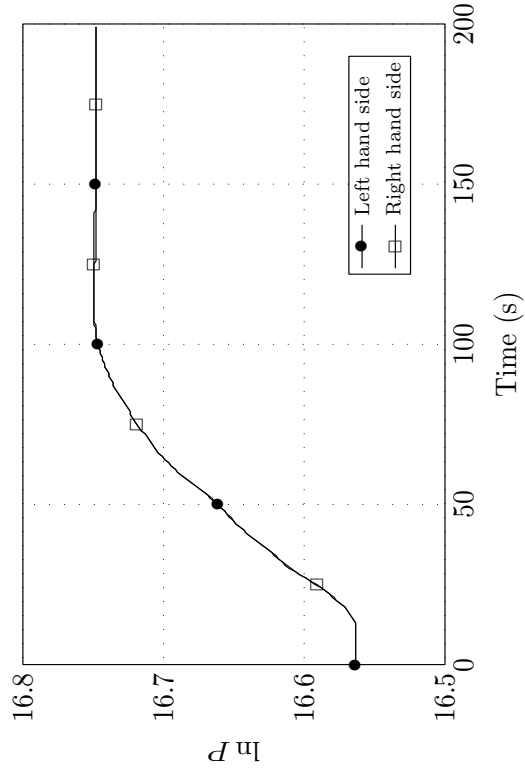


Figure 4: Verification of the pressure equation. Pipe 402.

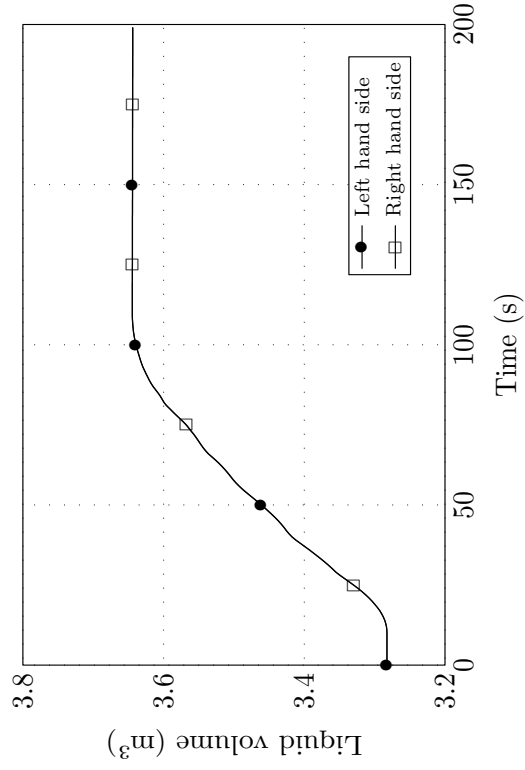


Figure 5: Verification of the level equation. Pipe 402.

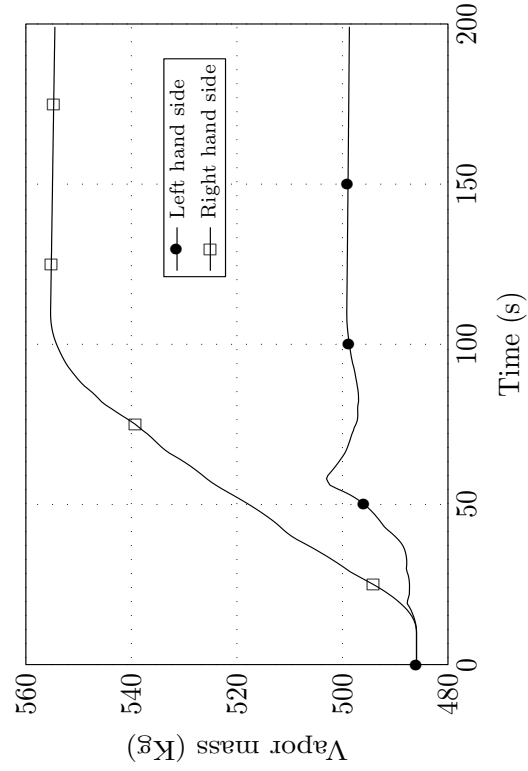


Figure 6: Verification of the vapor mass. Pipe 402.

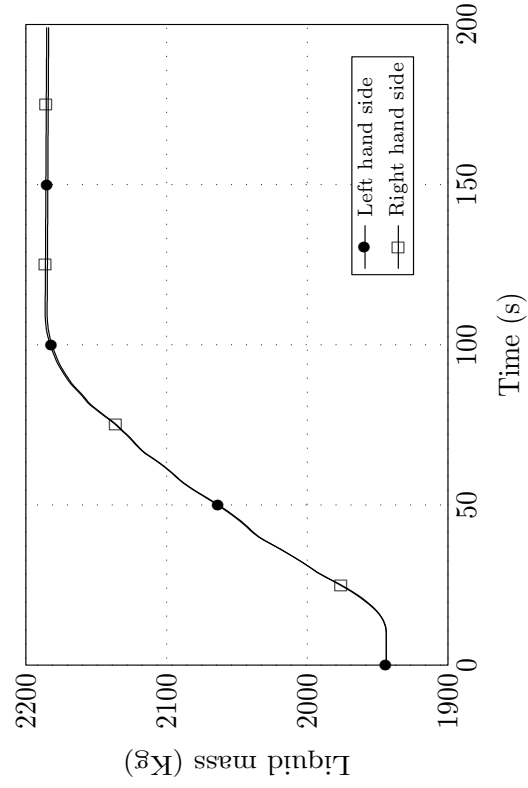


Figure 7: Verification of the liquid mass. Pipe 402.

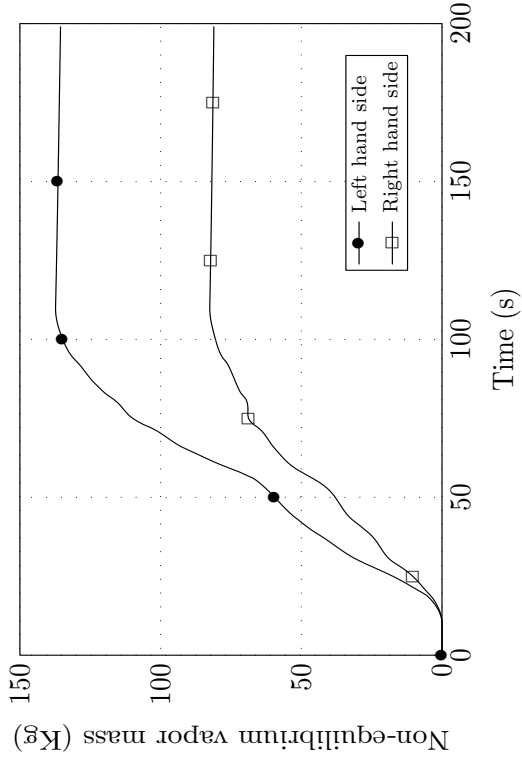


Figure 8: Non-equilibrium vapor mass. Pipe 402.

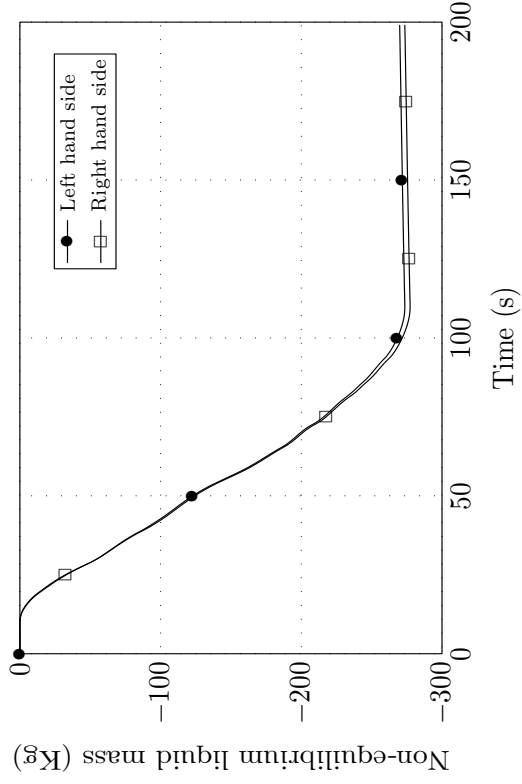


Figure 9: Non-equilibrium liquid mass. Pipe 402.

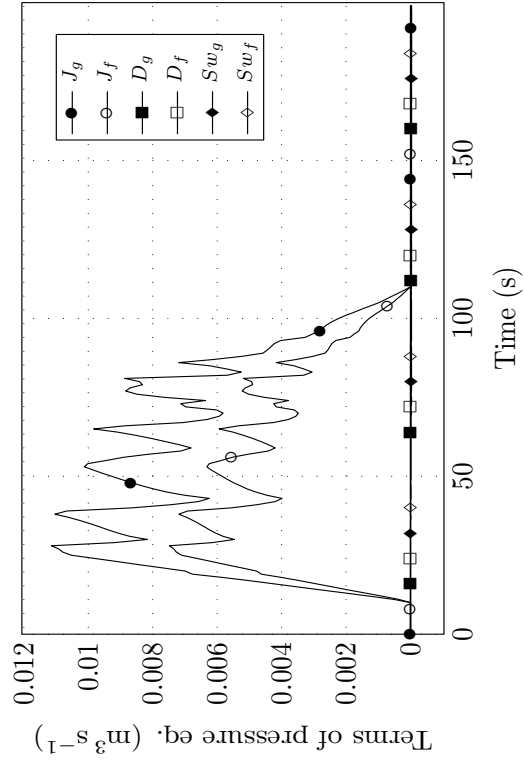


Figure 10: Terms of the pressure equation. Pipe 402.

It can also be noticed that the mass vapor is different in pipes 402 and 500. The liquid mass decreases as well, unlike what happens in pipe 402. Discrepancies in the non-equilibrium mass equation, caused by the errors in the vapor mass, are still observed.

The only non-negligible terms in the pressure and volume equations are still the volumetric ones (see figure 17). However, the shapes of the vapor and liquid volumetric terms are quite different, contrary to pipe 402. This additional element of asymmetry is another indicator of an error in the definition of the problem, very probably induced through the input file.

Looking thoroughly into the input file, it was found out that the homogeneous equilibrium model had been specified in the junction 403, very probably as a result of a *cut and paste* operation. This specification introduces an element of asymmetry in the problem, which is enough to justify the different results obtained in the pipes 402 and 500. The inadvertent use of the homogeneous equilibrium model in the junction 403 explains why the volumetric flows in figure 10 have the same shape. In a homogeneous model the velocity of each phase is identical. Thus the difference in the volumetric flows is caused by the scaling of the void fraction. Once the error in junction 403 has been fixed, the example and the postprocessing tool have been rerun.

The figure 18 shows that, once more, the analysis of the pressure equation is satisfactory. Both sides of the level equation (see figure 19) are also in good agreement. It must be realized that the level decreases as expected. Significant mass errors (figure 20) are still observed in the vapor phase, suggesting that the solution is not completely right even when the homogeneous model flag has been corrected. On the other hand, the liquid mass shown in figure 21 is properly preserved. As a consequence of the mass errors, the non-equilibrium vapor mass analysis is not satisfactory either. The non-equilibrium liquid mass is preserved, because the liquid mass is. The terms of the pressure and level equations are plotted in figure 24. The only non-negligible terms are the volumetric ones. It is very curious that in the permanent regime at the end of the transient the vapor volumetric flow is compensated by the vapor flow, resulting in a steady pressure. On the contrary, the level never reaches a steady state, remaining in an oscillating permanent regime, since only the liquid volumetric flow plays a role in the corresponding equation.

The results in the pipe 500 are identical to those obtained in pipe 402, as shown in figures 25 to 31. This confirms that, as expected, the results are symmetric with respect to the spatial variable x .

4.3 Depressurization in a circuit with liquid and vapor

4.3.1 Justification of the example

The example shown in this section illustrates the use of the analysis tool to interpret the results computed by the RELAP5 code. It will be also used to check the hypotheses that lead to the simplified non-equilibrium mass equations derived in section 2.5. The example consists in a closed hydraulic circuit, nodalized as represented in figure 32, initially filled with

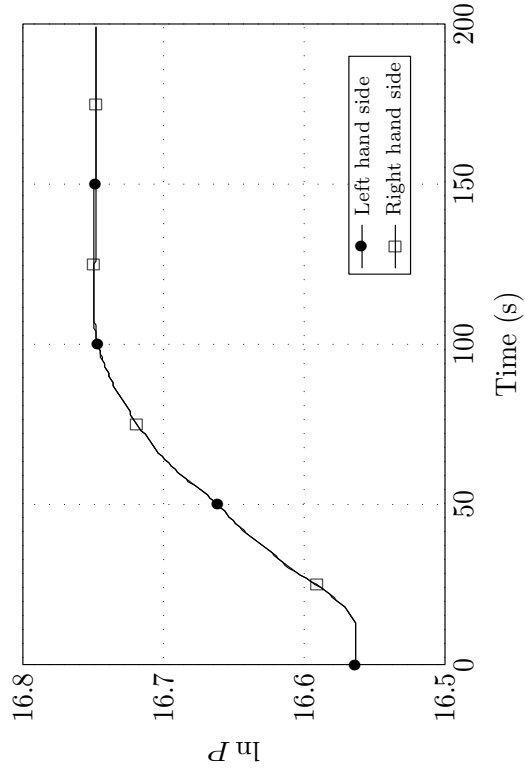


Figure 11: Verification of the pressure equation. Pipe 500.

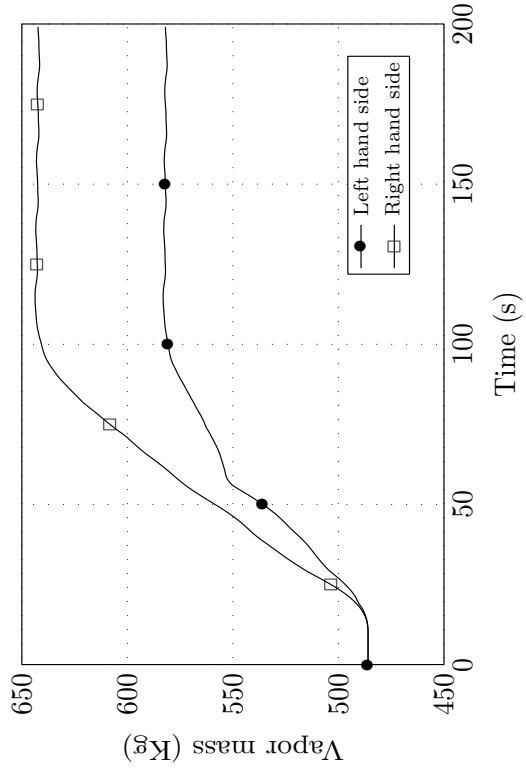


Figure 13: Verification of the vapor mass. Pipe 500.

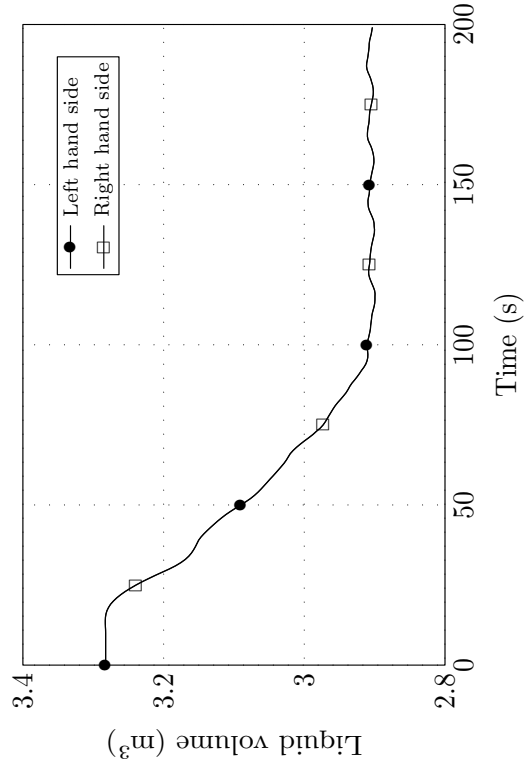


Figure 12: Verification of the level equation. Pipe 500.

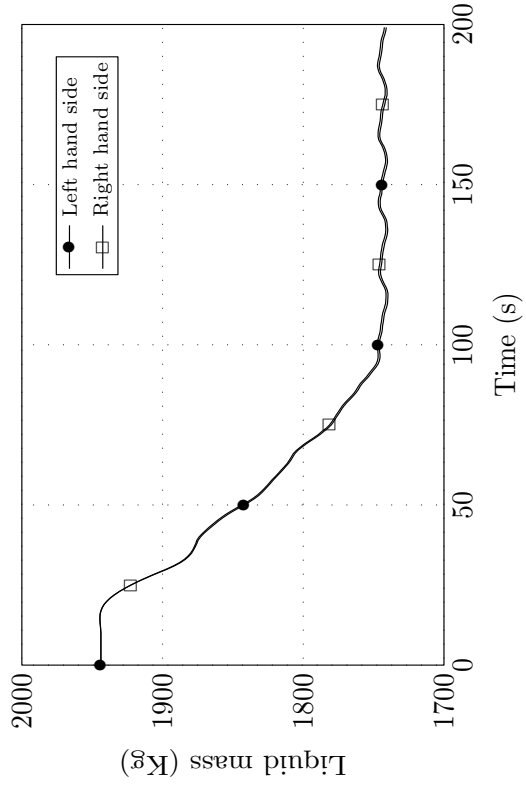


Figure 14: Verification of the liquid mass. Pipe 500.

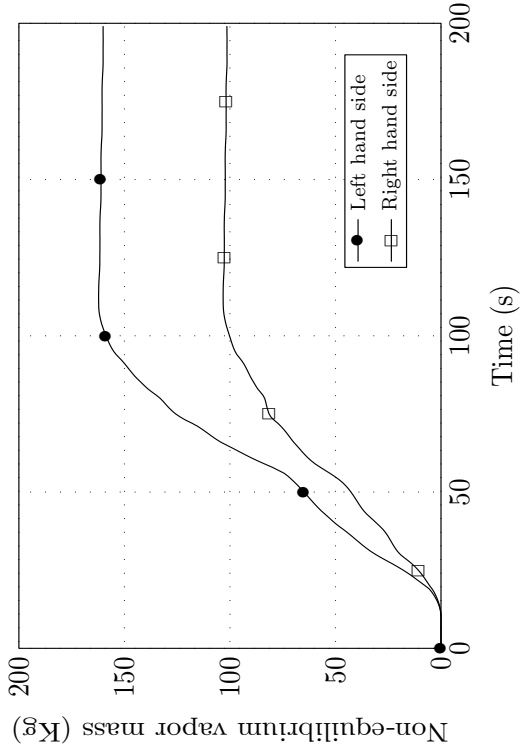


Figure 15: Non-equilibrium vapor mass. Pipe 500.

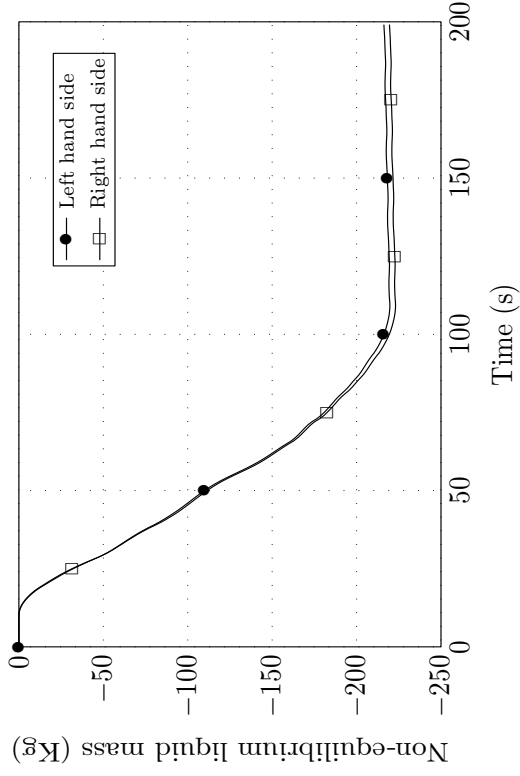


Figure 16: Non-equilibrium liquid mass. Pipe 500.

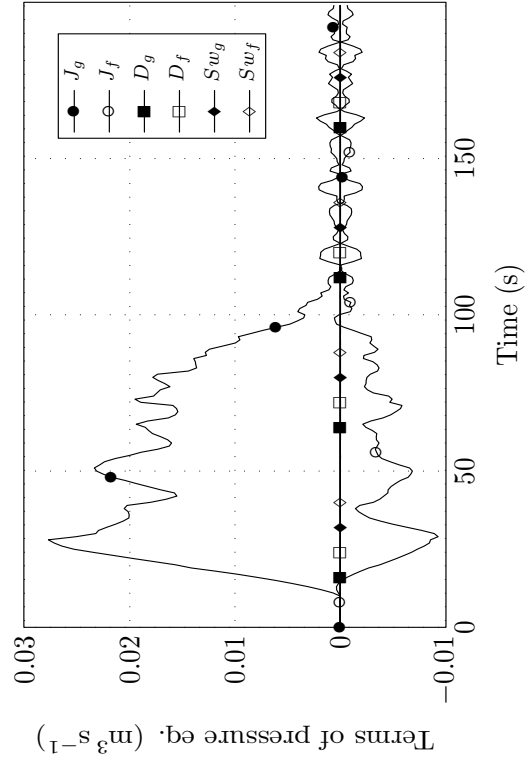


Figure 17: Terms of the pressure equation. Pipe 500.

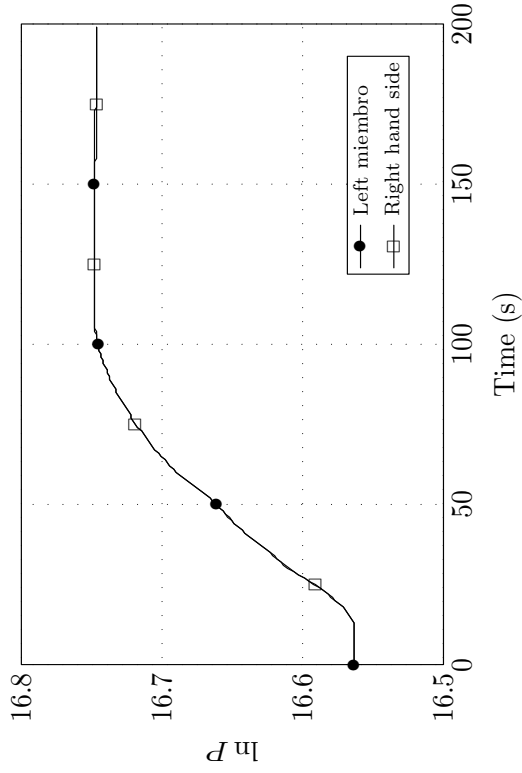


Figure 18: Verification of the pressure equation. Pipe 402. Revised example.

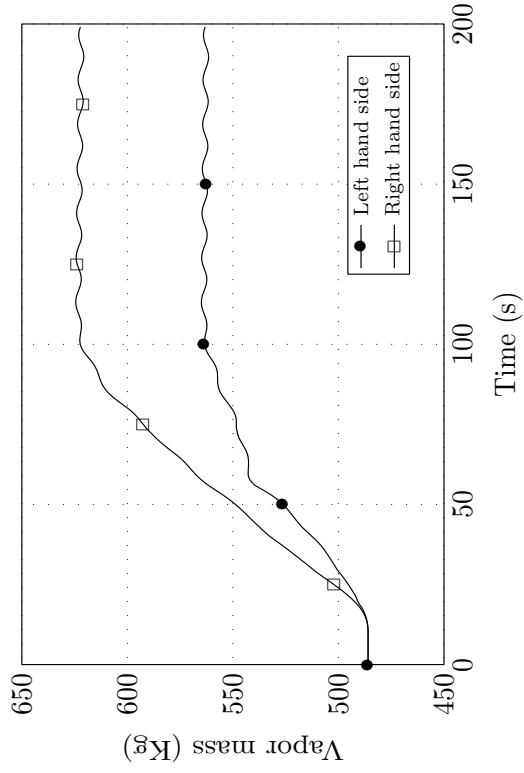


Figure 20: Verification of the vapor mass. Pipe 402. Revised example.

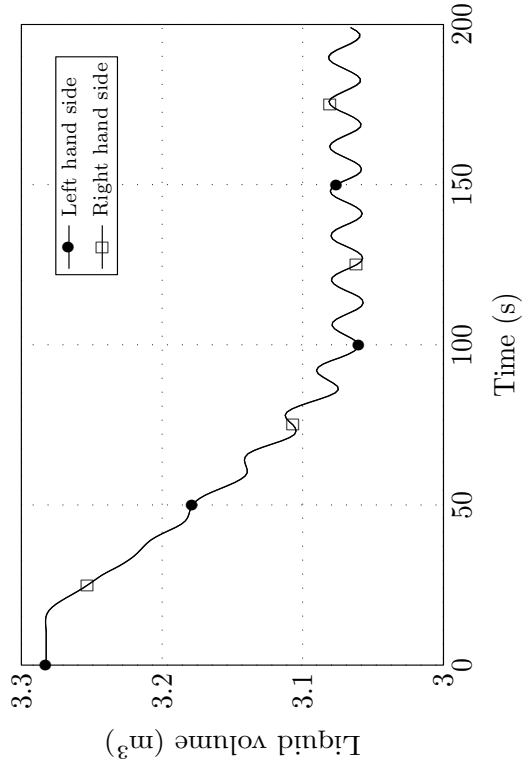


Figure 19: Verification of the level equation. Pipe 402. Revised example.

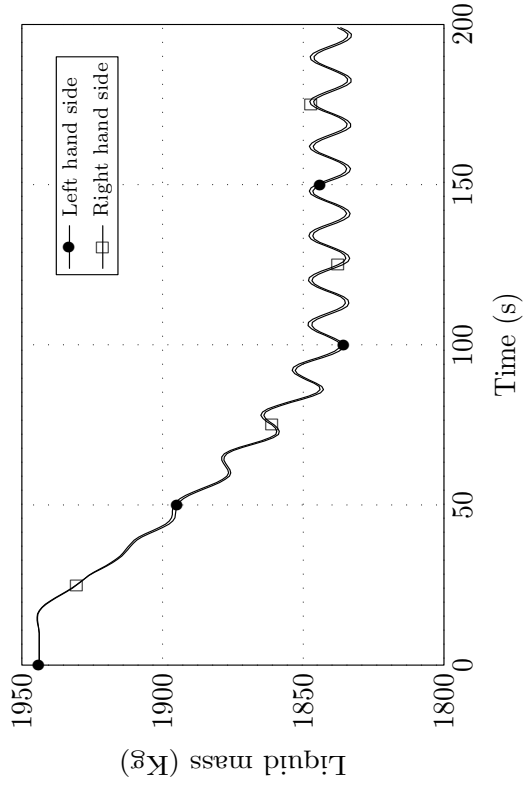


Figure 21: Verification of the liquid mass. Pipe 402. Revised example.

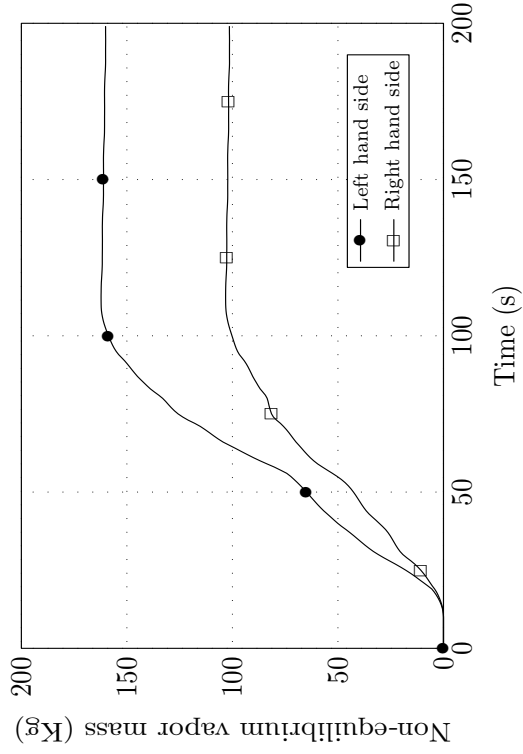


Figure 22: Non-equilibrium vapor mass. Pipe 402. Revised example.

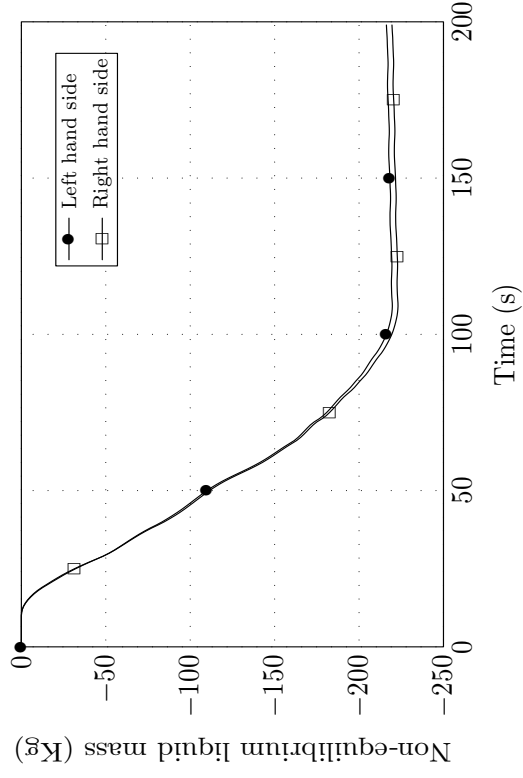


Figure 23: Non-equilibrium liquid mass. Pipe 402. Revised example.

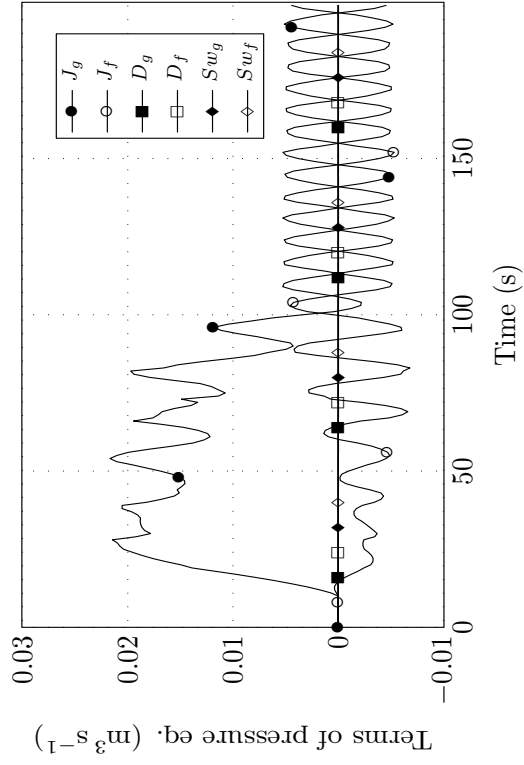


Figure 24: Terms of the pressure eq. Pipe 402. Revised example.

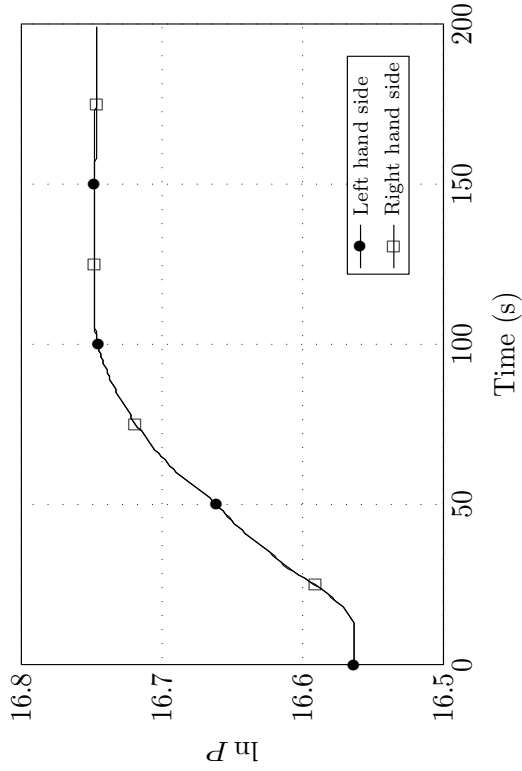


Figure 25: Verification of the pressure equation. Pipe 500. Revised example.

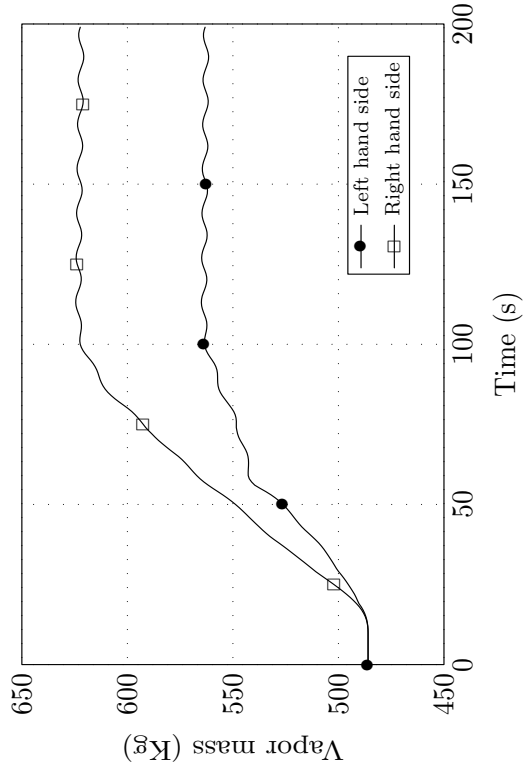


Figure 27: Verification of the vapor mass. Pipe 500. Revised example.

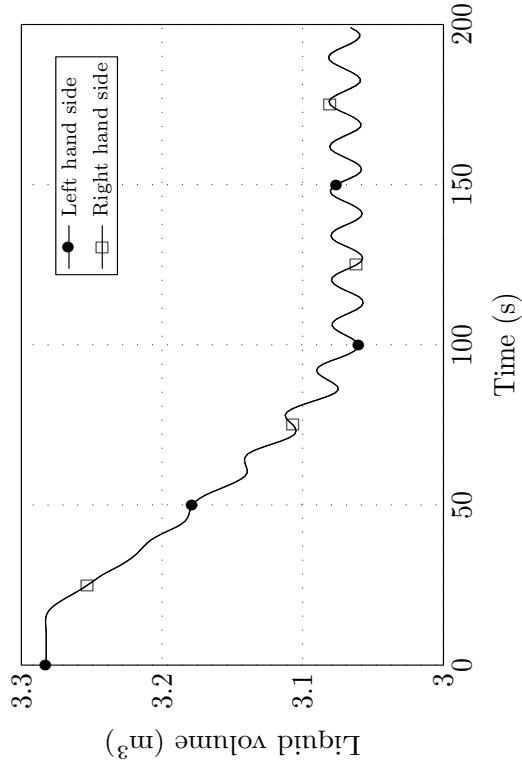


Figure 26: Verification of the level equation. Pipe 500. Revised example.

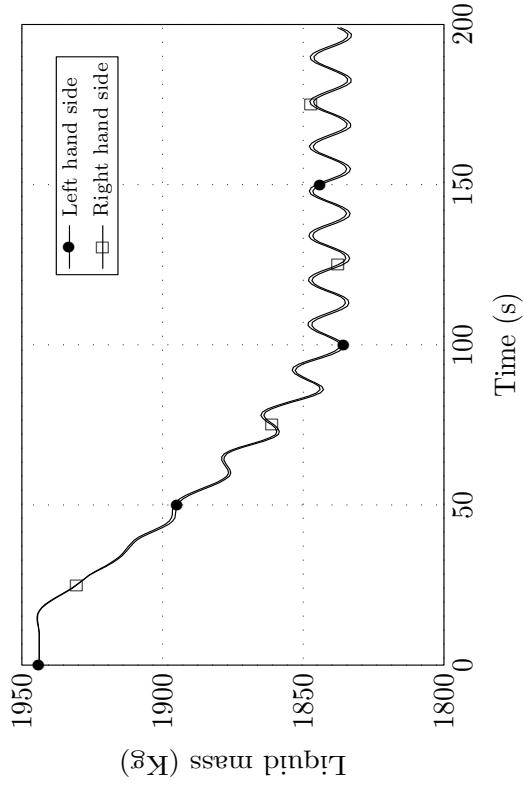


Figure 28: Verification of the liquid mass. Pipe 500. Revised example.

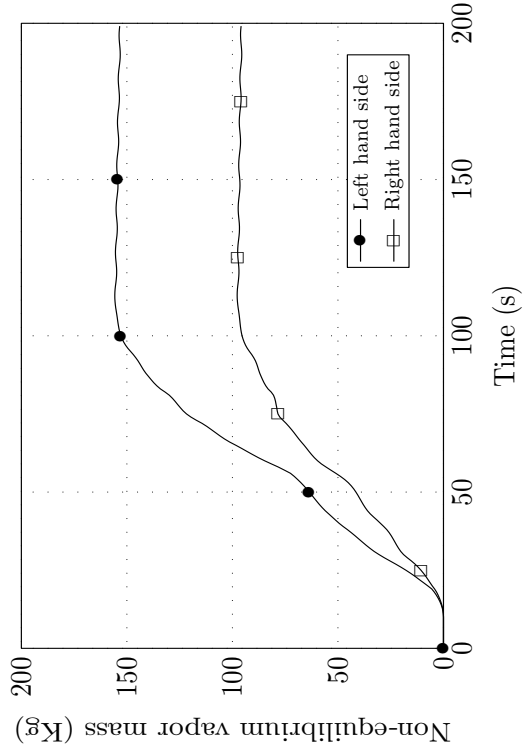


Figure 29: Non-equilibrium vapor mass. Pipe 500. Revised example.

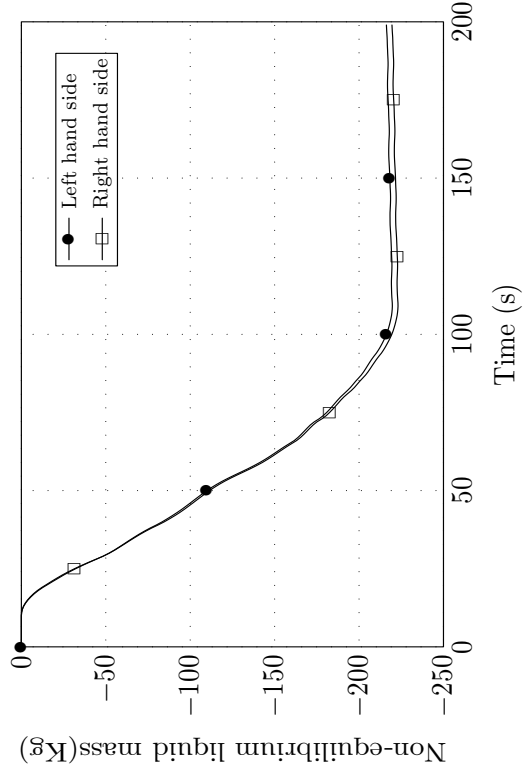


Figure 30: Non-equilibrium liquid mass. Pipe 500. Revised example.

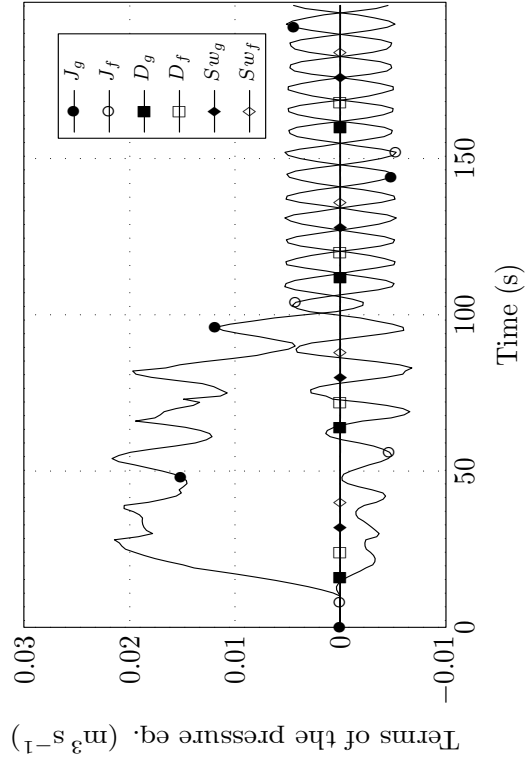


Figure 31: Terms of the pressure eq. Pipe 500. Revised example.

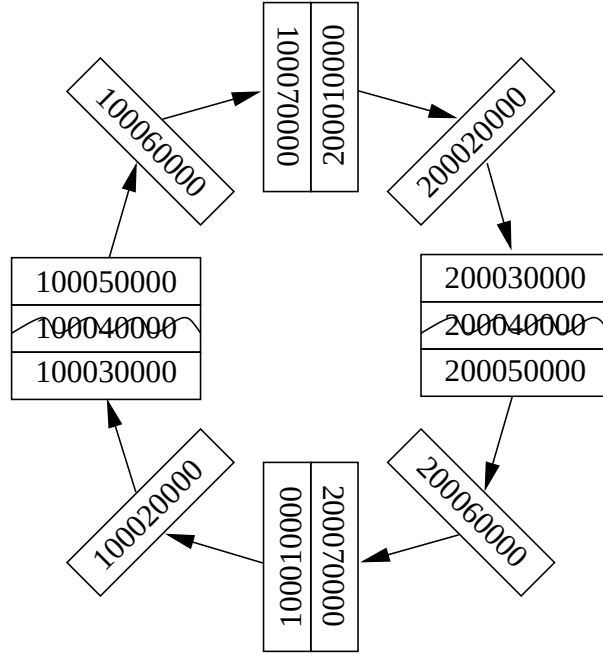


Figure 32: Circuit nodalization.

liquid until approximately half the level of the circuit. The other half is filled with vapor. The thermodynamic conditions are similar to those of the secondary circuit of a PWR. Being in stagnant conditions, the liquid occupies the lower half of the circuit and the vapor occupies the upper half. The exercise is focused in the comparison of the depressurization rates obtained when a breach of the circuit takes place either at the upper part or at the lower part. It is well known that a break in the steam line of a secondary circuit leads to a greater depressurization rate than a break in the feedwater line. The explanation given by many authors ([3] among them), is that the energy released through the break is greater if steam leaks. This argument assumes a direct relationship between the specific enthalpy of the fluid leaking through the break and the system pressure which is not demonstrated.

Figure 33 shows how, as predicted, the depressurization rate is greater if the break is induced in the upper part of the circuit, filled with vapor.

4.3.2 Verification of the balance equations

Before proceeding to interpret and explain the results, the fulfillment of the mass, pressure and level equations will be checked, according to the equations (1), (29) and (31).

Figure 34 shows the two sides of the pressure equation, when the break takes place in the vapor region. The agreement between them is fair. No objections can be made to the results from this viewpoint at least.

The two sides of the level equation (31), shown in figure 35, are also in good agreement.

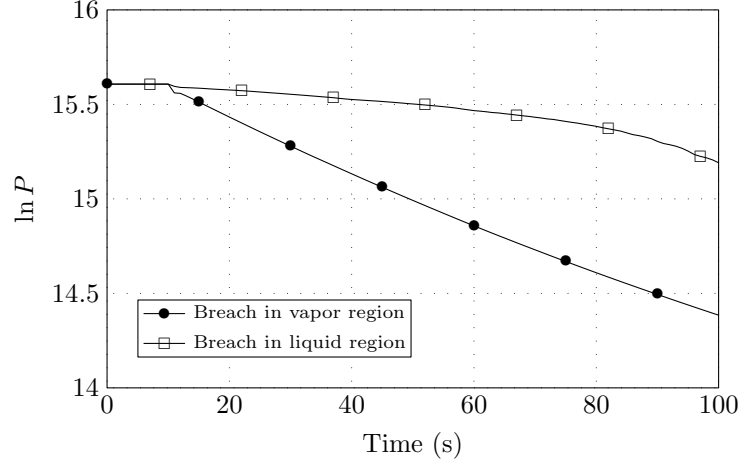


Figure 33: Pressure evolution.

On the contrary, the postprocess of the vapor and liquid mass equations, shown respectively in figures 36 and 37, leads to dissapointing results, questioning the quality of the solution. Note that the errors in the vapor and the liquid phase are of similar order, but with opposite sign. This is a good example of compensation of the mass errors between the phases, which is not detected by the RELAP5 mass verification.

If the break takes place in the liquid region, the analysis of the pressure, level and vapor and liquid mass, represented respectively in figures 38, 39, 40 and 41, leads to similar results. Error compensation between the liquid and vapor phases is observed as well, although it is smaller than when the break occurs in the vapor region.

4.3.3 Interpretation of results

Although the specific internal energy and enthalpy are greater for the vapor phase than for the liquid phase, the energy and enthalpy flows released through the break are greater when the break takes place in the liquid region, because of the higher density of this phase. This is confirmed in figures 42 and 43. As a conclusion, the higher depressurization rate when the break occurs in the vapor region cannot be explained in terms of the released energy. Another explanation must be found.

The results computed by RELAP5 can be interpreted according to the pressure equation (29). Figure 44 shows the terms in the numerator of the pressure equation, except the negligilbe Ac term, when the break is induced in the vapor region. The same terms, for the case of break in the liquid region, are shown in figure 45.

It is easy to realize that the depressurization is mainly driven by two term, in the first part of the transient at least. Those are the volumetric term J (for the vapor phase in the first case and for the liquid phase in the second one) and the swelling term Sw_g (caused by the appearance of vapor). Both terms tend towards mutual compensation, although the

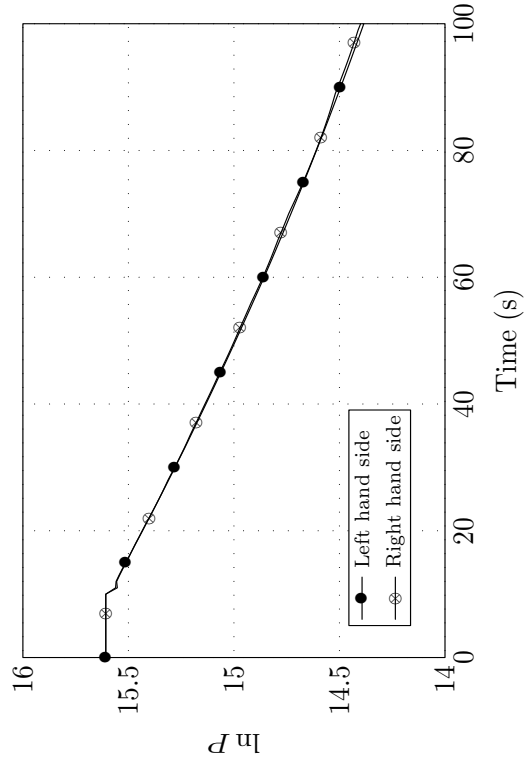


Figure 34: Pressure analysis. Break in vapor region.

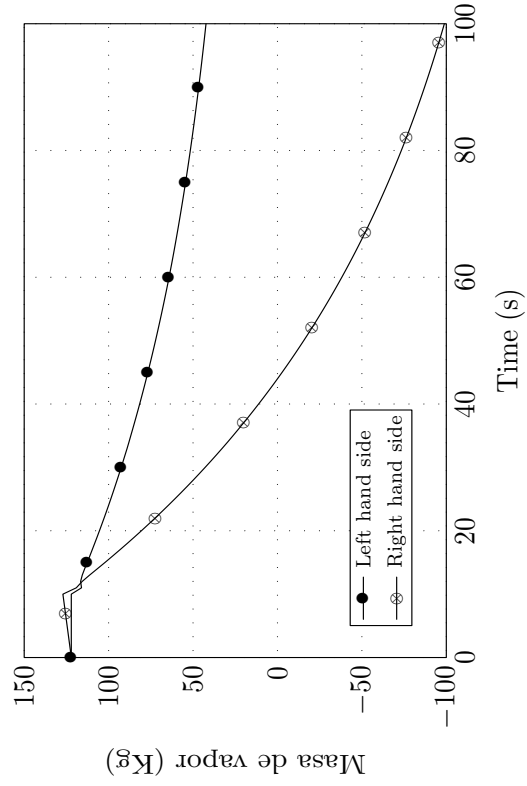


Figure 36: Vapor mass analysis. Break in vapor region.

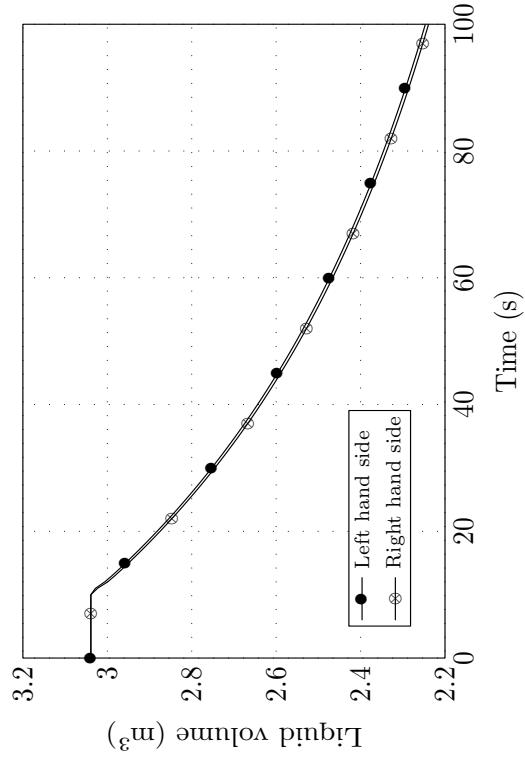


Figure 35: Liquid volume analysis. Break in vapor region.

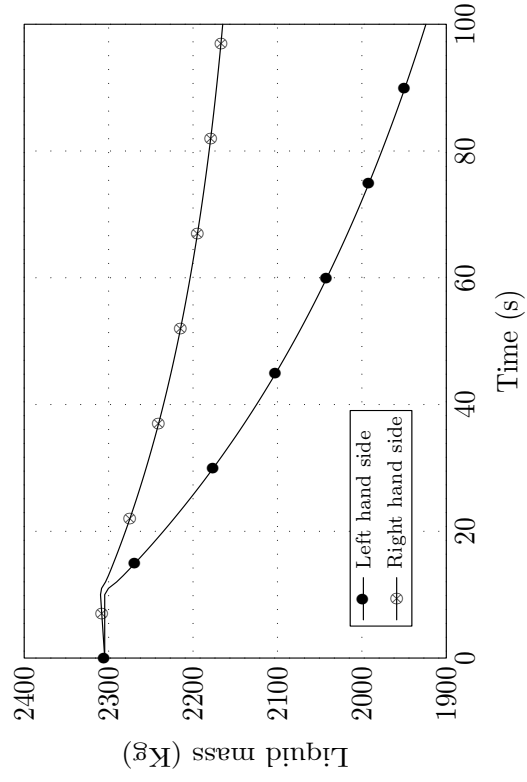


Figure 37: Liquid mass analysis. Break in vapor region.

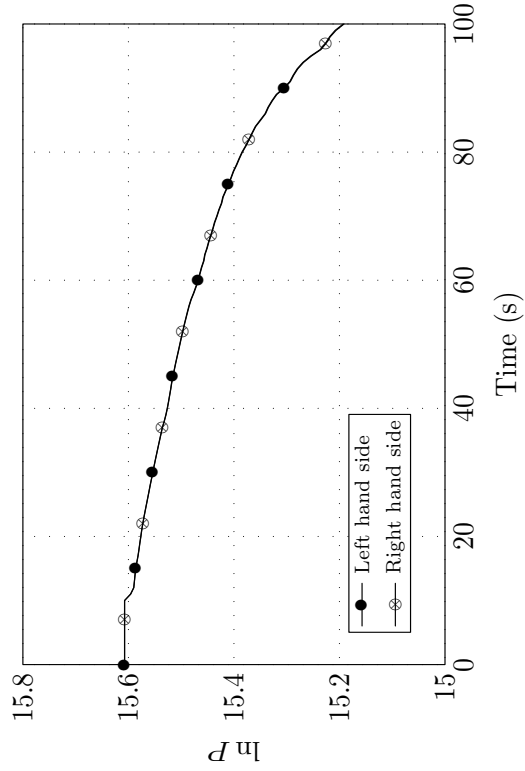


Figure 38: Pressure analysis. Break in liquid region.

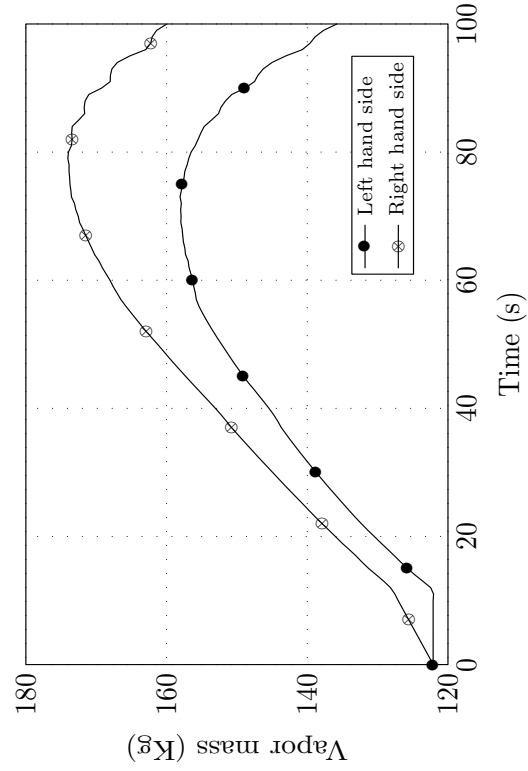


Figure 40: Vapor mass analysis. Break in liquid region.

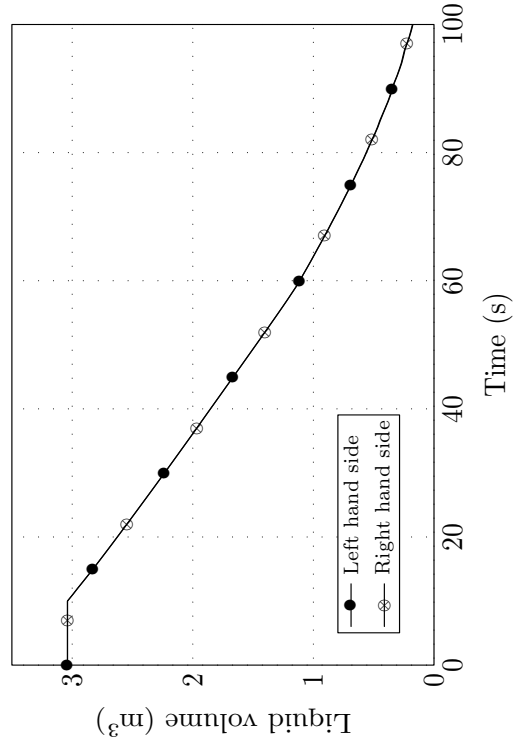


Figure 39: Level analysis. Break in liquid region.

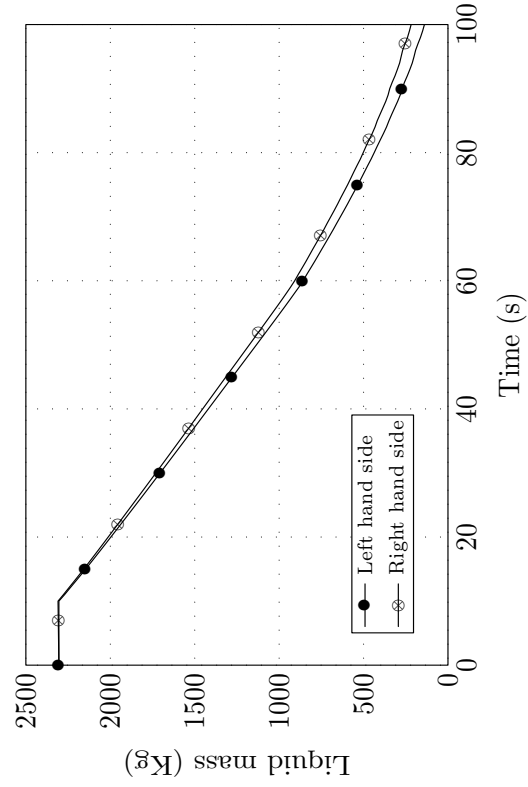


Figure 41: Liquid mass analysis. Break in liquid region.

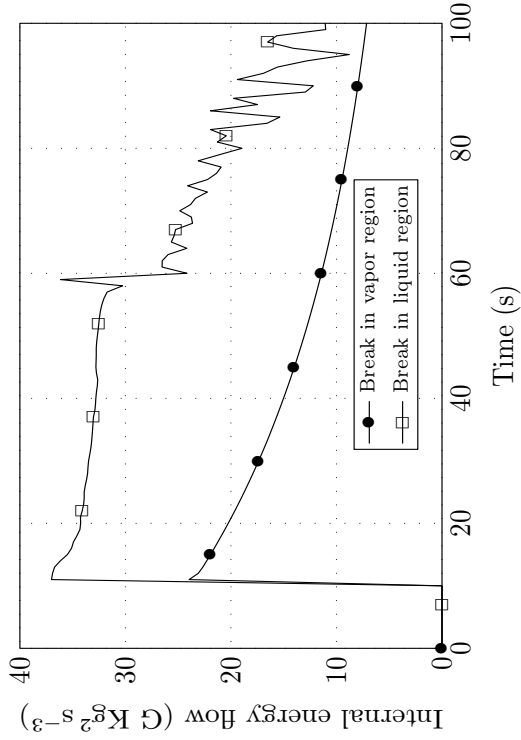


Figure 42: Energy flow released through the breaks.

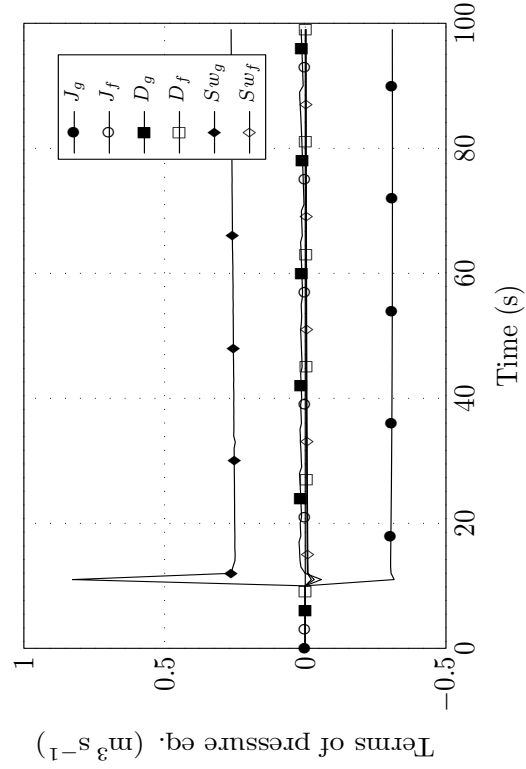


Figure 44: Terms of pressure eq. Break in vapor region.

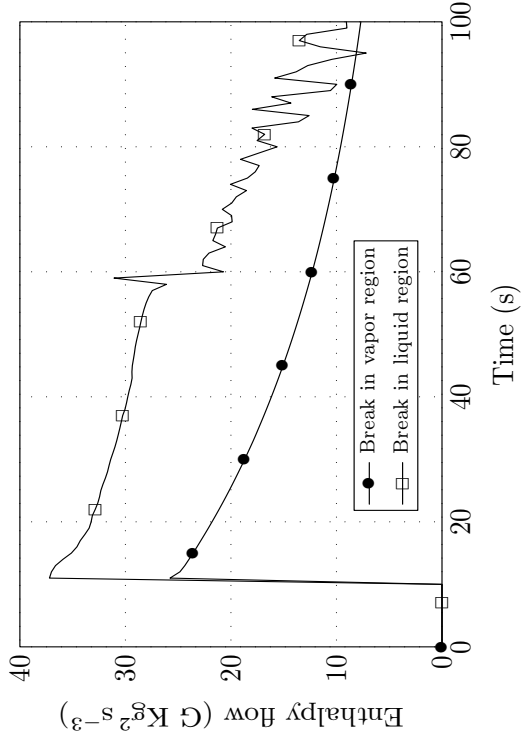


Figure 43: Enthalpy flow released through the breaks.

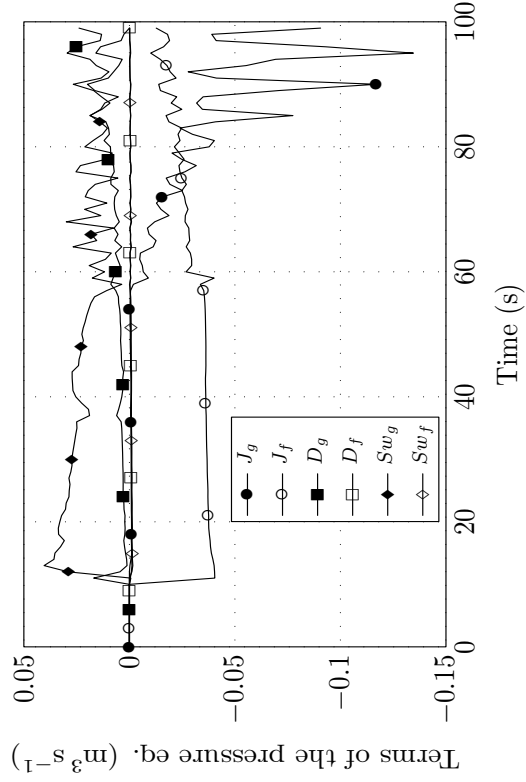


Figure 45: Terms of pressure eq. Break in liquid region.

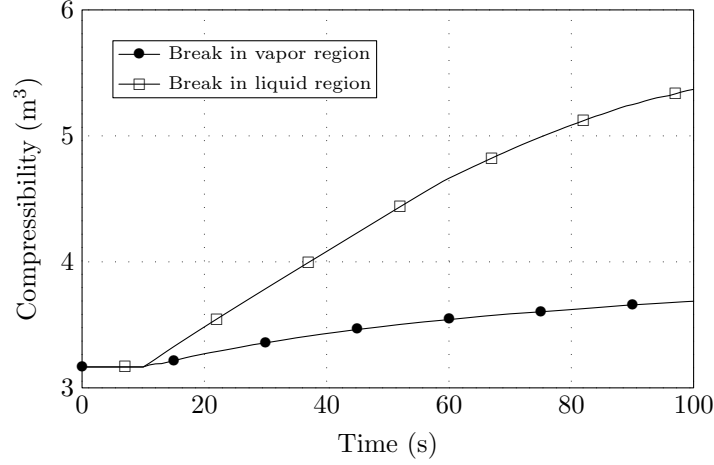


Figure 46: System compressibility.

volumetric term is always the dominant one. The sum of the two terms is greater when the break occurs in the vapor region, leading in this case to a greater numerator in the pressure equation. Moreover, the system compressibility, which acts as the denominator of the pressure equation, is greater when the break takes place in the liquid, as shown in figure 46. The global effect is that the depressurization rate is greater for a break in the vapor region.

4.3.4 Verification of the simplified non-equilibrium model

This section is devoted to the verification of the hypotheses under which the simplified non-equilibrium mass equations described in section 2.5 are valid and the two phases remain decoupled from each other. Fulfillment of equations (68) and (73) will then be checked in the current example. For the sake of brevity, only the control volumes 10001 (in the upper part of the circuit) and 10001 (in the lower part) will be checked. We remind that this is an exercise to illustrate how a code can be used to validate, as if it were an experimental facility, simpler models. We do not intend to validate the above mentioned hypotheses at the sight of this exercise. A true validation would require a huge number of transients and calculations.

If the break occurs in the vapor region, the vapor generation rate in volume 10001 is positive during the whole transient. Hence the verifications must be performed in this phase.

Figure 47 represents the right hand side of equation (49) and the term of that side including the dependency on Γ_{ig} . It is clear that this term contributes very weakly to the right hand side of equation (49).

The terms that depends on Γ_w is zero, since no heat is transferred through the wall.

Figure 48 displays the bulk interfacial heat versus its approximation according to equation (67). The agreement is excellent.

The liquid generation rate is positive in volume 20001 during all the transient, with the exception of the first instants. The right hand side of equation (49) is much larger than the term with Γ_{if} , as shown in figure 49. The bulk interfacial heat for the liquid phase is almost identical to its approximation (see figure 50). During the first instants of the transient the interfacial heat is zero according to the figure. This only means that Γ_{if} is negative during those instants and the simplified model is not applicable for the liquid phase. It must not be understood as a true zero value.

If the break occurs in the liquid region we come out with the similar conclusions, as shown in the figures 51 and 54.

It can be inferred at the sight of these results that the simplifying hypotheses of the non-equilibrium mass equation could be applied in this example at least.

4.4 Over specification of boundary conditions

4.4.1 Motivation and background

This example tries to show how the post-processing techniques can be used to identify potential mistakes committed during the development of a TH computer code, which can lead to unphysical systems even if the problem is well posed from the mathematical point of view. A historical review of the boundary condition over specification problem is pertinent here.

During the 1996 Fall CAMP Meeting [35] the overspecification of boundary conditions in the RELAP5 code when using time dependent junctions was already pointed out. The problem was illustrated with an example based on a simplified pressurizer model. No correcting actions were taken at that time, probably because the root cause of the problem had not been identified yet.

During the 2000 Fall CAMP Meeting, Dr. Jingzao Zhang from Tractebel presented a simple RELAP5 case which led to surprising and unphysical results. It will be demonstrated here that the cause of these results is, like in the pressurizer case presented in [35], the overspecification of boundary conditions. Zhang's case will be finally used to illustrate the problem and explain the root causes, since it is much simpler and leads to more dramatic results than the pressurizer case.

Zhang's case, depicted in figure 55, comprises a time dependent volume (TDV) at the inlet side and another at the outlet side, one single volume, one time dependent junction (TDJ) that connects the inlet TDV to the single volume to specify the inlet flow, and a single junction at the outlet side. The pressure in the inlet TDV is set to 4 MPa and the temperature to 553 K. The pressure of the outlet TDV is set 2.1 MPa, and the temperature, which does not affect the result since the volume is placed downstream, to 523 K. The mass flow in the TDJ is kept constant at a value of 10 Kg s^{-1} . The initial conditions in the single volume are those of the outlet TDV. The initial velocity in the single junctions is zero. In the conditions described above, the system contains superheated steam. Since the inlet TDJ forces the fluid from the inlet TDV to replace the fluid of the single volume, we expect a increase of its

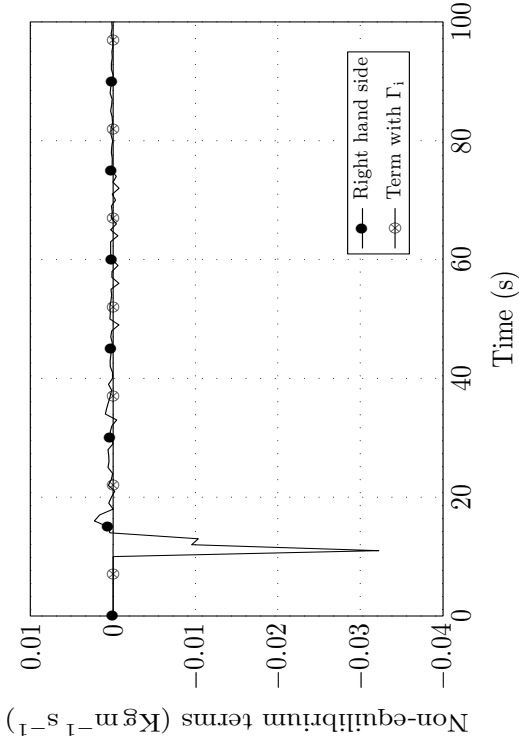


Figure 47: Neglection of term with Γ_i in the non-equilibrium eq. Vapor phase. Volume 10001. Break in vapor region.

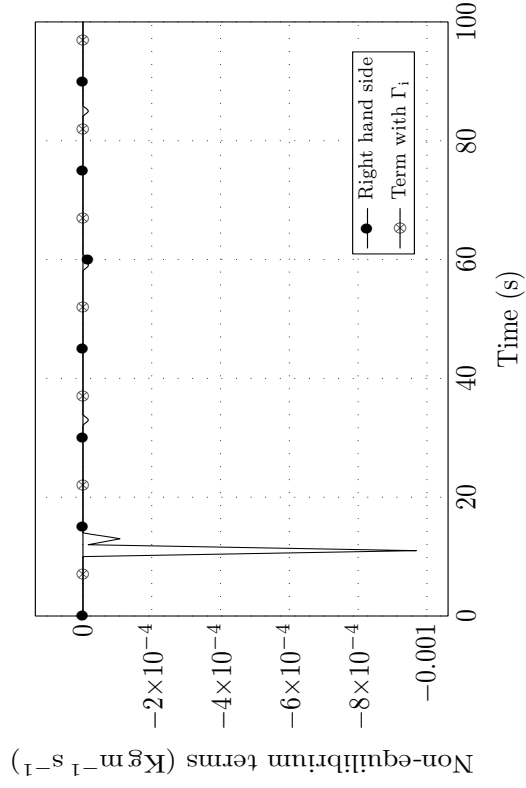


Figure 49: Neglection of term with Γ_i in the non-equilibrium eq. Liquid phase. Volume 20001. Break in vapor region.

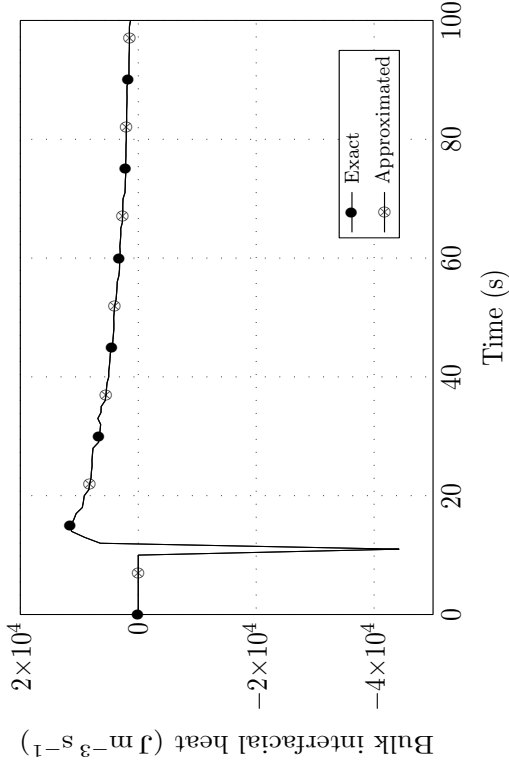


Figure 48: Approximation of the interfacial heat power in the vapor phase. Volume 10001. Break in vapor region.

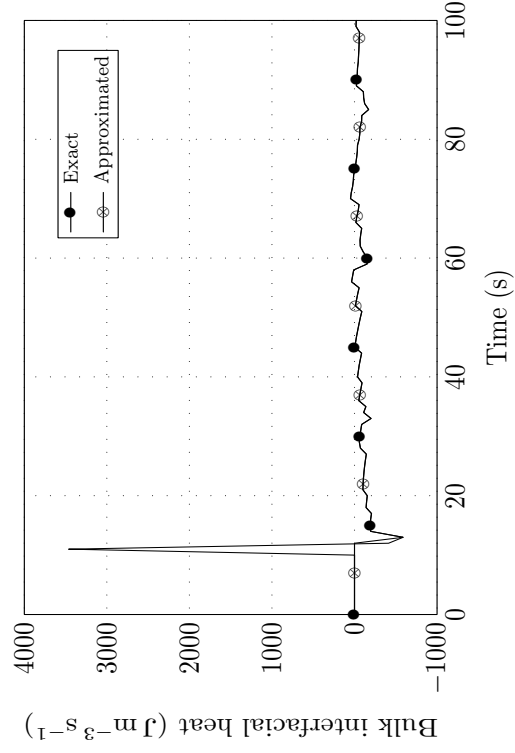


Figure 50: Approximation of the interfacial heat power in the liquid phase. Volume 20001. Break in vapor region.

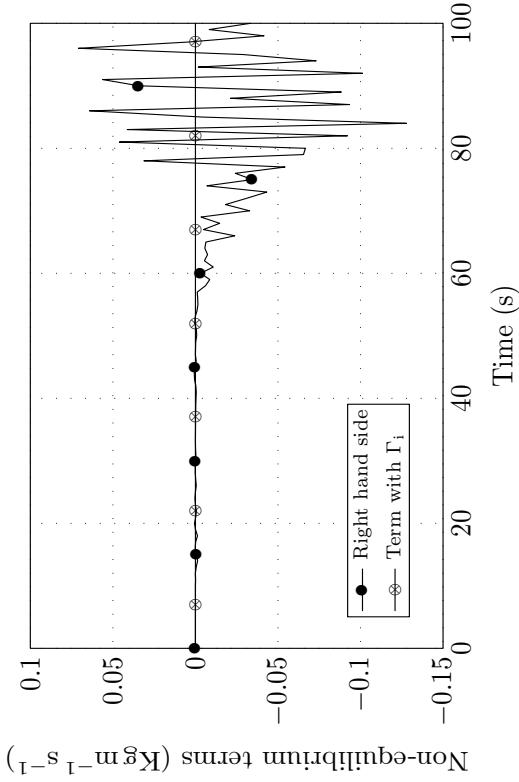


Figure 51: Neglection of term with Γ_i in the non-equilibrium eq. Vapor phase. Volume 10001. Break in liquid region.

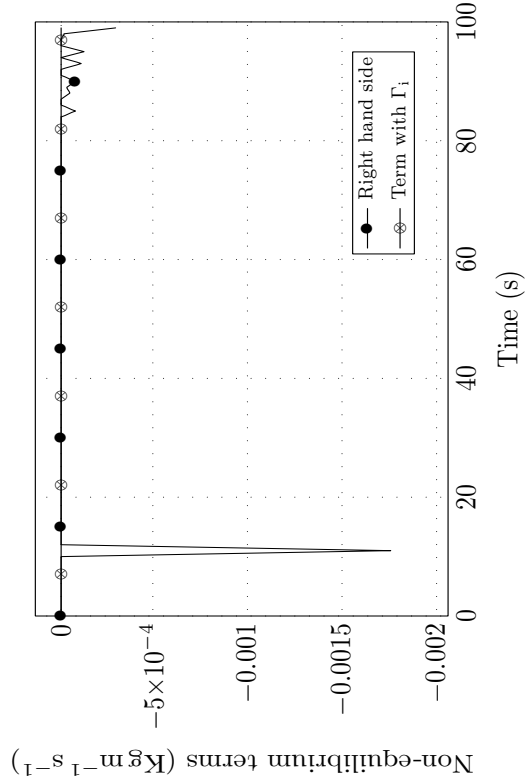


Figure 53: Neglection of term with Γ_i in the non-equilibrium eq. Liquid phase. Volume 20001. Break in liquid region.

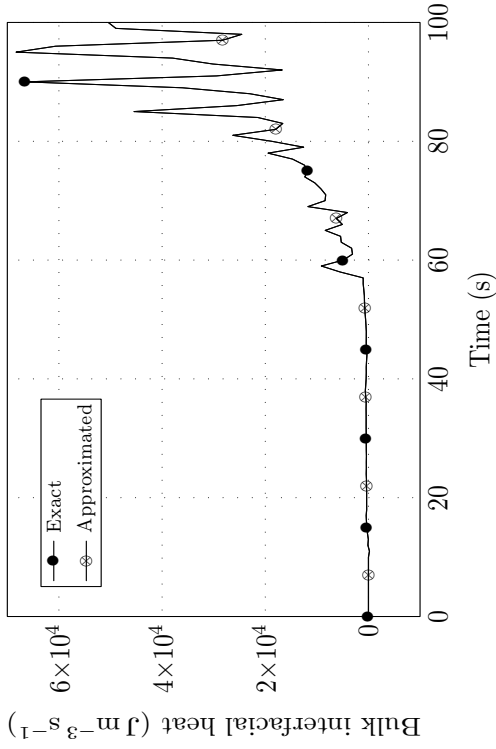


Figure 52: Approximation of the interfacial heat power in the vapor phase. Volume 10001. Break in liquid region.

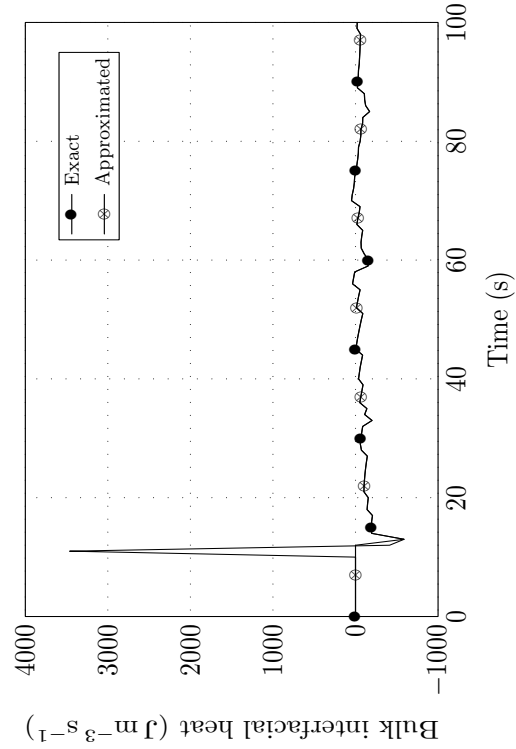


Figure 54: Approximation of the interfacial heat power in the liquid phase. Volume 20001. Break in liquid region.

temperature. In the final steady-state, the thermodynamic properties of the single volume should be very similar to the boundary conditions specified in the inlet TDV.

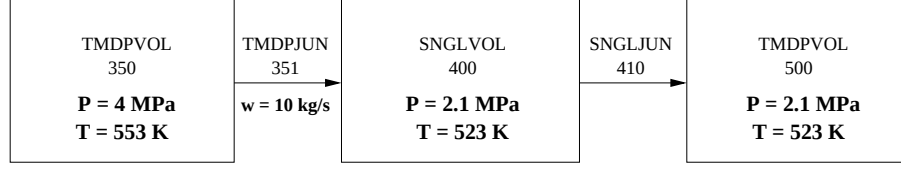


Figure 55: Zhang's problem.

4.4.2 Original case

The postprocessing of Zhang's example reveals some inconsistencies. First of all, the global pressure equation is not fully satisfied, as shown in figure 56. Figure 57 shows that, although a little liquid fraction appears in volume 400 during the transient, that fraction is not accounted for in the right hand side of the level equation. The vapor mass, represented in 58, is properly preserved. The non-equilibrium vapor mass balance in figure 59 is not fulfilled. The pressure drop derived from the momentum equation, and caused mainly by the friction in volume 400, has been represented against the difference $P_{350} - P_{500}$ in figure 60. The mismatch between the two computed variables is evident.

Direct observation of some physical variables provides evidences of unphysical results as well. Figure 61 shows that the temperature in volume 400 decreases during the transient, contrary to expectations. The vapor enthalpy in figure 62 undergoes an unexpected variation too. The temperature and the enthalpy were just the two variables that moved Dr. Zhang to present his results. Other variables present an abnormal behavior, such as the density in figure 63, and the pressure in figure 64.

4.4.3 Zhang's problem with PV term

Reference [107] tries to explain the results obtained in Zhang's problem and to provide a solution that leads to physical results.

That reference states that the internal energy equation (3) is not able to preserve the enthalpy, since its discretization leads to the following convective terms upstream and downstream, in the single phase case:

$$(\dot{\rho}_{j+1}\dot{u}_{j+1} + P_L) v_{j+1}A_{j+1} \quad (74)$$

$$- (\dot{\rho}_j\dot{u}_j + P_L) v_jA_j \quad (75)$$

where L stands for the single volume 400. The term (74) represents a donor cell formulation of the enthalpy equation convective term in the downstream junction, while the term (75) does not represent an equivalent upstream term. For this to occur, the donor cell pressure P_K should appear in the term instead of the downstream pressure P_L .

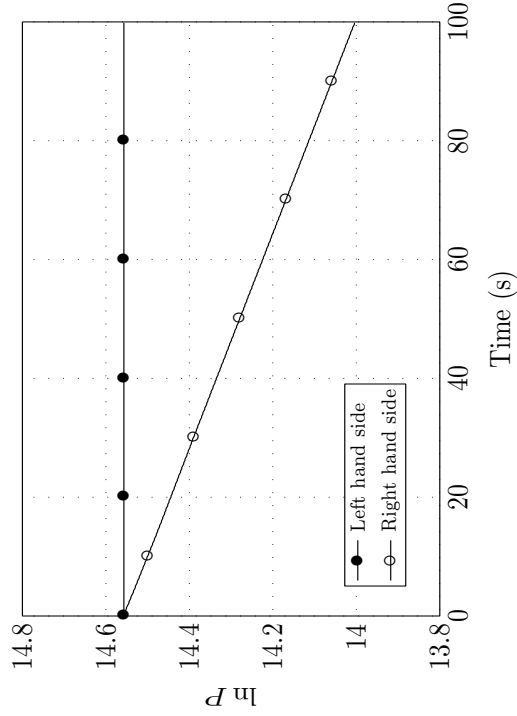


Figure 56: Zhang's problem: pressure postprocessing.

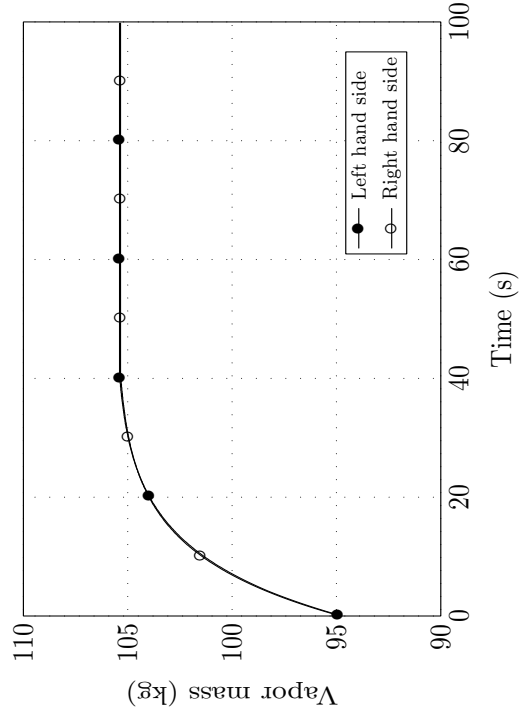


Figure 58: Zhang's problem: vapor mass postprocessing.

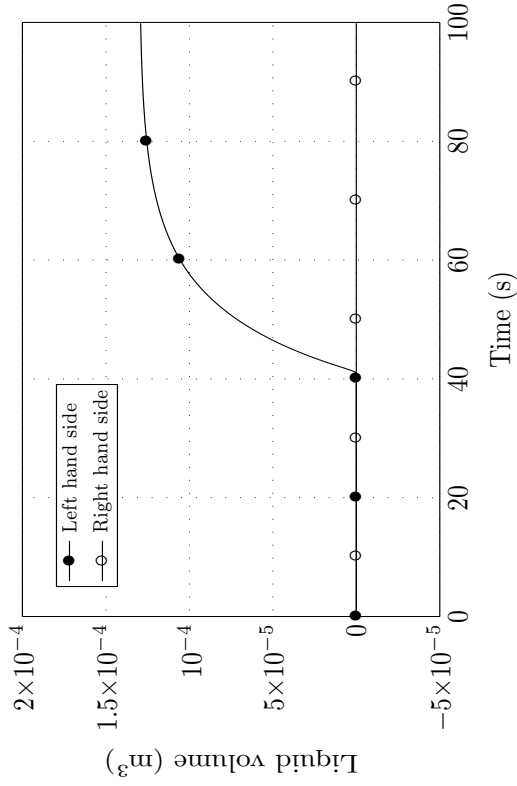


Figure 57: Zhang's problem: level postprocessing.

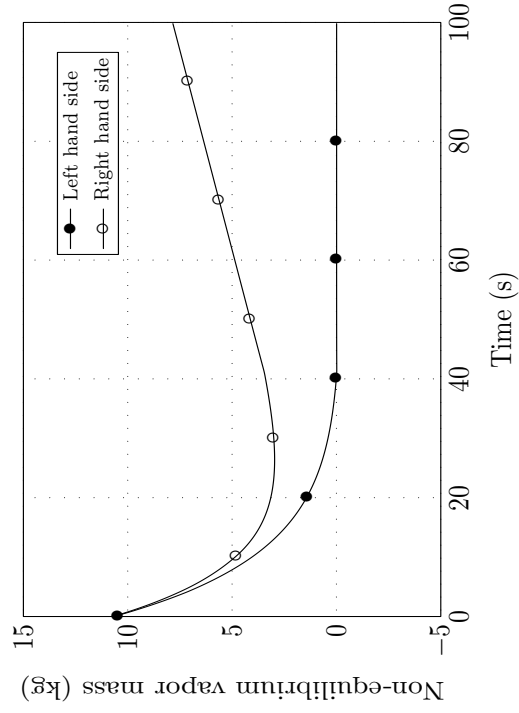


Figure 59: Zhang's problem: non-equilibrium vapor mass postprocessing.

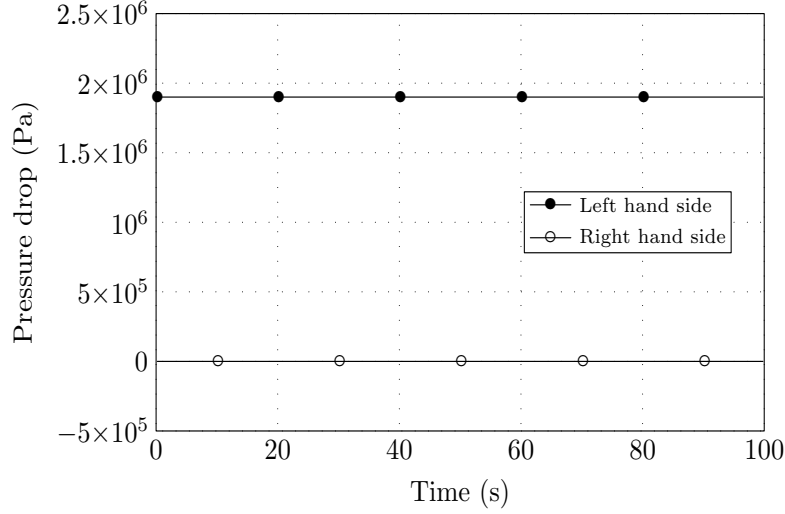


Figure 60: Zhang’s problem: pressure drop postprocessing.

To force this behavior, a new option has been added to the junctions of a developmental version of the RELAP5 code. Upon activation of this option, called the *e-flag*, the code adds the term $-(P_K - P_L) v_j A_j$ to the energy equation in the corresponding junction, where KK stands for the upstream TDV 350. This term is referred to as the *PV term* in the RELAP5 documentation.

The results obtained with the activation of the e-flag in the inlet time dependent junction couldn’t have been postprocessed as the developmental version of RELAP5 including the PV term in the TDJs has not been available. Physical inconsistencies still arise anyway. Although the vapor enthalpy approaches asymptotically the value defined in volume 350 (see figure 66), the temperature does not show the same behavior (see figure 65). The temperature is increased during the transient, but it never reaches the value in volume 350. The density, represented in figure 67, remains constant in its initial value. The pressure, shown in figure 68, is kept in its initial value as well.

4.4.4 Over specification suppression with pressure feedback

Having proved the inefficiency of the PV term, it is necessary to explain the cause of the unphysical results obtained with Zhang’s problem and why the PV term fixes the enthalpy balance even when other variables such as the temperature and the density do not behave as expected. Single phase flow will be assumed in the following, which does not result in a lack of generality.

When using time dependent volumes only, without time dependent junctions, the following boundary conditions are specified:

- the upstream pressure,

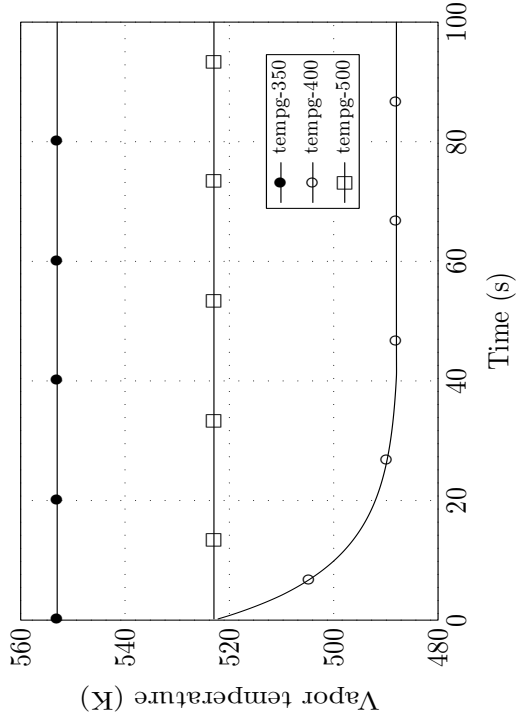


Figure 61: Zhang's problem: vapor temperature.

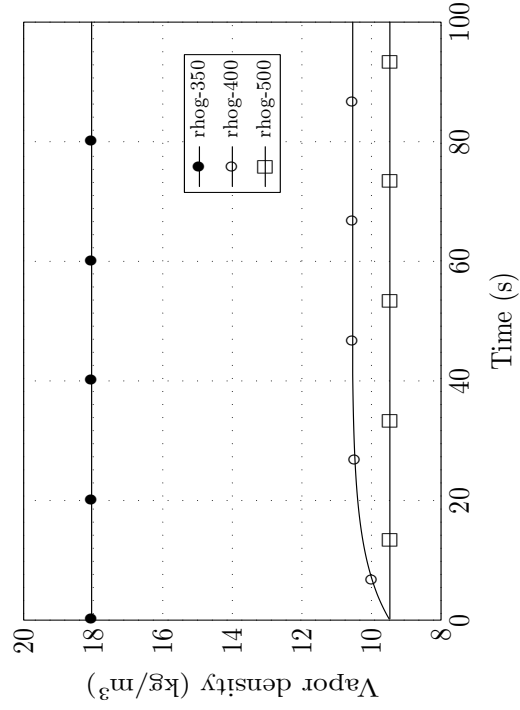


Figure 63: Zhang's problem: vapor density.

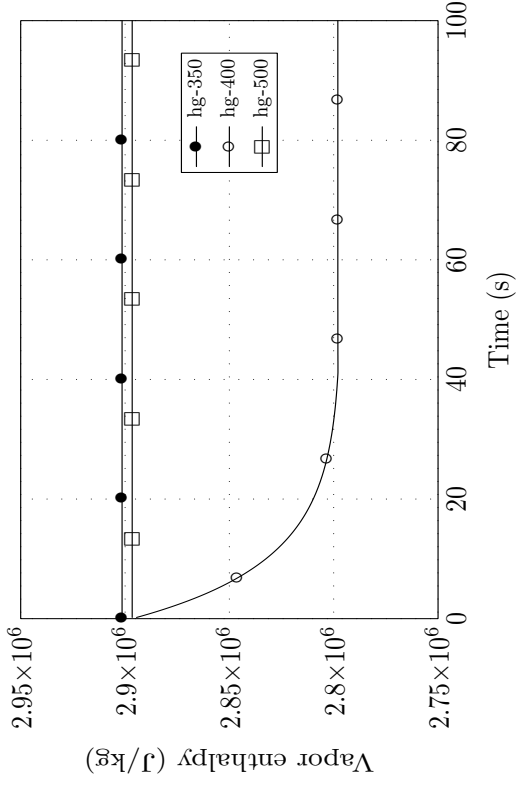


Figure 62: Zhang's problem: vapor enthalpy.

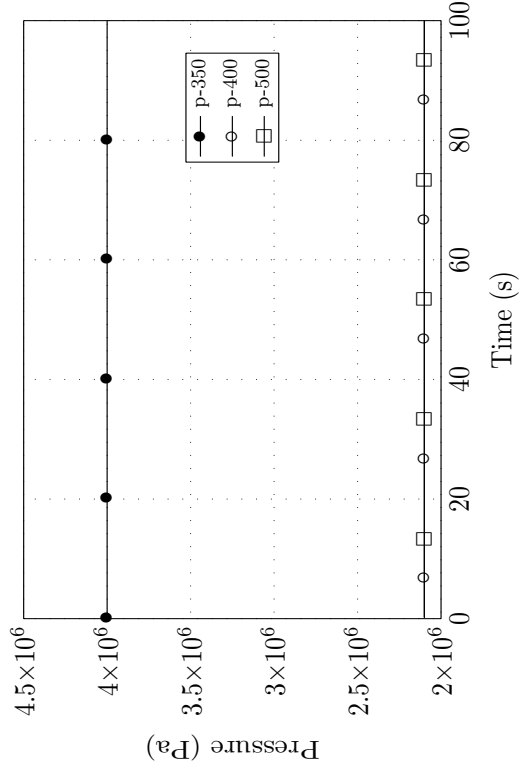


Figure 64: Zhang's problem: pressure.

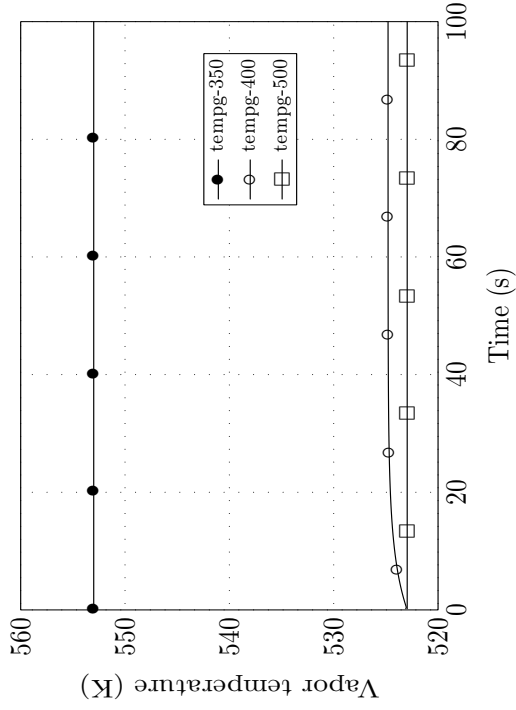


Figure 65: Vapor temperature with PV term.

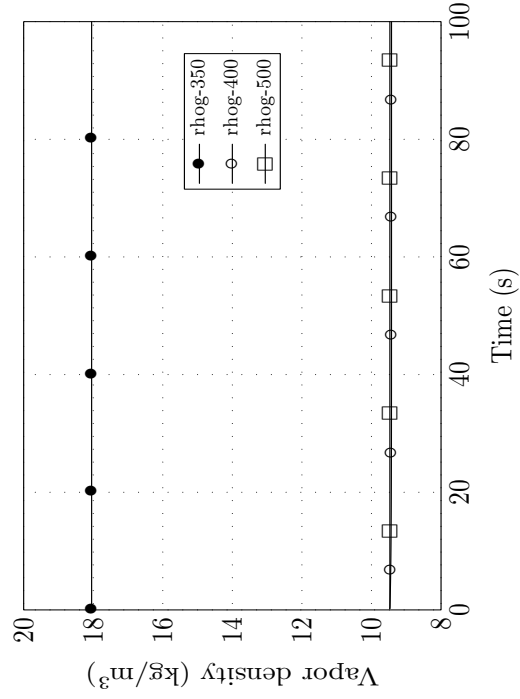


Figure 67: Vapor density with PV term.

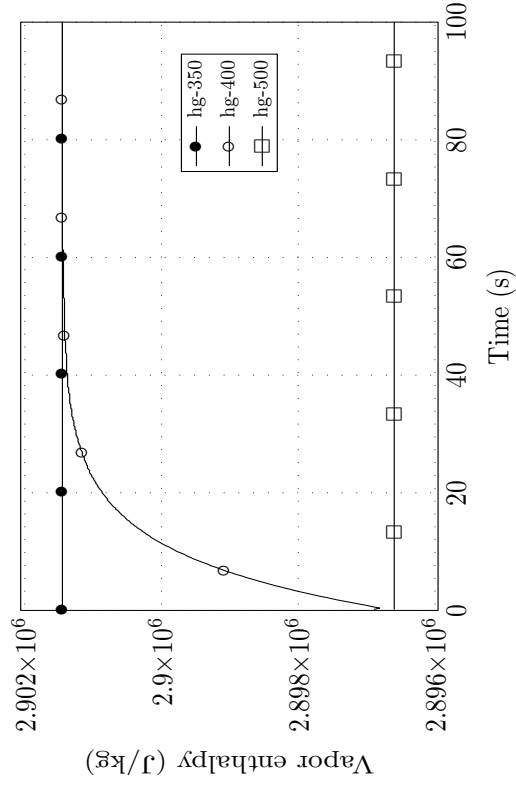


Figure 66: Vapor enthalpy with PV term.

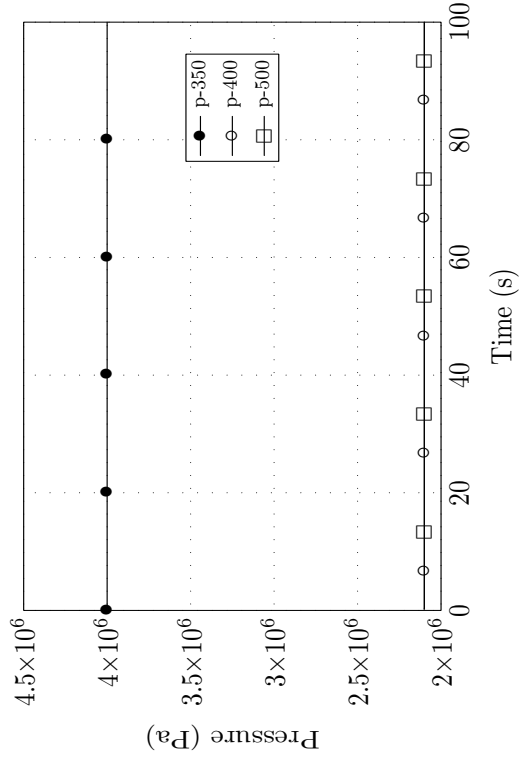


Figure 68: Pressure with PV term.

- a thermodynamic property defined upstream (typically the temperature or the internal energy)
- the downstream pressure (the rest of variables defined in the downstream time dependent volume are not donnored and have no influence in the code results, although the code requests the user to input those values)

The number of boundary conditions is then three. The RELAP5 solves in this case the following set of linearized equations (see [95] for details):

$$\mathbf{A}\mathbf{x} = \mathbf{b} + \mathbf{g}_1 v_{j+1}^{n+1} + \mathbf{g}_2 v_j^{n+1} \quad (76)$$

where the matrices are defined as follows:

$$\mathbf{x} = \begin{bmatrix} \tilde{u}_L^{n+1} - u_L^n \\ P_L^{n+1} - P_L^n \end{bmatrix} \quad (77)$$

$$\mathbf{A} = \begin{bmatrix} \left(\frac{\partial \rho}{\partial u}\right)_L^n & \left(\frac{\partial \rho}{\partial P}\right)_L^n \\ u_L \left(\frac{\partial \rho}{\partial u}\right)_L^n + \rho_L^n & u_L \left(\frac{\partial \rho}{\partial P}\right)_L^n \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0 \\ (Q_{w,L}^n + \text{DISS}_L^n) \Delta t \end{bmatrix} \quad (78)$$

$$\mathbf{g}_1 = \begin{bmatrix} \dot{\rho}_{j+1}^n A_{j+1} \Delta t / V_L \\ -(\dot{\rho}_{j+1}^n \dot{u}_{j+1}^n + P_L^n) A_{j+1} \Delta t / V_L \end{bmatrix} \quad \mathbf{g}_2 = \begin{bmatrix} -\dot{\rho}_j^n A_j \Delta t / V_L \\ (\dot{\rho}_j^n \dot{u}_j^n + P_L^n) A_j \Delta t / V_L \end{bmatrix} \quad (79)$$

It is assumed that the density in matrix $\mathbf{A}$ depends on the pressure P and the internal energy u . When a partial derivative of the density is present in that matrix, with respect to a given variable, it must be understood that the other variable is kept constant.

The two linear equation system (76) is obtained from the mass and energy equations, discretized in the control volume L. The whole system is obtained by adding the two equations obtained from the discretization of the momentum equation in the junctions j and $j + 1$:

$$\begin{aligned} -\Delta t (P_L^{n+1} - P_j^{n+1}) &= -\rho_j^n B_x \frac{\Delta x_L}{2} \Delta t \\ &+ \frac{1}{2} \dot{\rho}_j^n \left[(v^2)_L^n - (v^2)_j^n \right] \Delta t - \rho_j^n \frac{\Delta x_L}{2} v_j^n \\ &+ \left[\rho_j^n \frac{\Delta x_L}{2} (1 + \text{FW}_j^n \Delta t) + \dot{\rho}_j^n \text{HLOSS}_j^n \Delta t \right] v_j^{n+1} \end{aligned} \quad (80)$$

$$\begin{aligned}
-\Delta t (P_{j+1}^{n+1} - P_L^{n+1}) &= -\rho_{j+1}^n B_x \frac{\Delta x_L}{2} \Delta t \\
&+ \frac{1}{2} \dot{\rho}_{j+1}^n \left[(v^2)_{j+1}^n - (v^2)_L^n \right] \Delta t - \rho_{j+1}^n \frac{\Delta x_L}{2} v_{j+1}^n \\
&+ \left[\rho_{j+1}^n \frac{\Delta x_L}{2} (1 + \text{FW}_{j+1}^n \Delta t) + \dot{\rho}_{j+1}^n \text{HLOSS}_{j+1}^n \Delta t \right] v_{j+1}^{n+1}
\end{aligned} \tag{81}$$

The previous four equation set has the following unknowns: P_L^{n+1} , u_L^{n+1} , v_j^{n+1} y v_{j+1}^{n+1} . The variable P_j^{n+1} in equation (80) refers to the pressure in the upstream TDV.

When a time dependent junction is used, a new boundary condition, the inlet velocity, is added. This makes a total of four boundary conditions. This increase of the number of degrees of freedom is a clear index of error, since that number is inherent to the physical system and should not be changed by the choice of the set of variables that will act as boundary conditions. The increase of boundary conditions is caused by the non-solving of the momentum equation in the TDJ. It's mistakenly believed that this equation is not necessary, since inlet velocities are provided. As a consequence, the discrete momentum equation in the inlet junction is suppressed, and the number of equations decreases from four to three. The mathematical problem is still consistent, since the unknown variables are also three (P_L^{n+1} , u_L^{n+1} and v_{j+1}^{n+1}). However, it is not consistent from the physical point of view, since the number of equations cannot be arbitrarily decreased. Once the momentum equation (81) is not solved, the pressures P_L^{n+1} and P_j^{n+1} are no longer linked by any physical principle. This observation modifies all the forms of the energy equation, since the upstream pressure P_j^{n+1} which should be computed by the code defines, along with the upstream temperature, a wrong density and internal energy. These two variables are introduced in the energy equation through the convective terms.

If the momentum equation were solved in the inlet junction, the system of linear equations would be:

$$\mathbf{A}\mathbf{x} = \mathbf{c} + \mathbf{g}_1 v_{j+1}^{n+1} \tag{82}$$

where

$$\mathbf{c} = \begin{bmatrix} -\dot{\rho}_j^n A_j \Delta t / V_L \\ (Q_{w,L}^n + \text{DISS}_L^n) \Delta t + (\dot{\rho}_j^n \dot{u}_j^n + P_L^n) A_j \Delta t / V_L v_j^{n+1} \end{bmatrix} \tag{83}$$

Note that $\mathbf{c}$ is the result of adding $\mathbf{b}$ and $\mathbf{g}_2 v_j^{n+1}$, since the latter term is no longer an unknown. The momentum equations (80) and (81) are also a part of the system. Nevertheless, the unknown variable in equation (80) is now P_j^{n+1} instead of v_j^{n+1} . With this system the number of equations (four) and unknowns (P_L^{n+1} , u_L^{n+1} , v_{j+1}^{n+1} y P_j^{n+1}) is preserved.

Once the root cause of the abnormal results obtained in Zhang's problem has been found, it is necessary to explain why the PV term is enough to preserve the enthalpy, even though the momentum equation is not solved in the inlet junction yet.

As the spatial mesh is finer and finer, the PV term tends to $vA\frac{\partial P}{\partial x}$.

Adding that term to the analytical form of the internal energy equation (3), assuming single phase regime, unheated pipe, and neglecting the energy dissipated by friction, such an equation becomes:

$$\frac{\partial}{\partial t}(A\rho u) + \frac{\partial}{\partial x}(A\rho uv) + vA\frac{\partial P}{\partial x} = -P\frac{\partial Av}{\partial x} \quad (84)$$

Grouping some terms, and taking into account that the enthalpy is expressed as $h = u + \frac{P}{\rho}$, we obtain

$$\frac{\partial}{\partial t}(A\rho h) + \frac{\partial}{\partial x}(A\rho hv) = \frac{\partial}{\partial t}(AP) \quad (85)$$

Integrating (85) in space and time:

$$\begin{aligned} \int_V A\rho h dx(t) &= \int_V A\rho h dx(0) + \int_0^t \left(A\rho hv|_{\text{in}} - A\rho hv|_{\text{out}} \right) dt \\ &+ \int_V AP dx(t) + \int_V AP dx(0) \end{aligned} \quad (86)$$

If the temporal changes of the pressure are negligible as compared to the terms carrying the enthalpy, the previous equation becomes an enthalpy balance with no influence of the upstream and downstream pressures. Even if the pressure gradient is very large, as it is the case in Zhang's problem, the balance (86) will be preserved anyway. This is the reason why the enthalpy balance is fulfilled when the PV term is introduced.

It must be highlighted that equations (84) and (85) do not represent any physical principle, since they are derived from adding a fictitious term to a physically meaningful equation.

It has been proven that the root cause of Zhang's results is the over specification of boundary conditions, which derives in turn from skipping the computation of the momentum equation in the time dependent junctions. The ideal solution would be the modification of the RELAP5 code, to apply a different solution algorithm when TDJs are used. However, a less traumatic solution can be devised. It consists in computing the momentum equation in the inlet junction through the control variables supplied by the code.

The pressure in the upstream time dependent volume is no longer set by the user. It equals the pressure in the centre of volume 400, plus the pressure drop due to the momentum equation. This approach leads to the following equation, which is valid for single phase flow and a pipe with uniform cross-section:

$$\rho \left[\frac{\partial v}{\partial t} + \frac{1}{2} \frac{\partial v^2}{\partial x} \right] = -\frac{\partial P}{\partial x} - \text{FWAL} v \quad (87)$$

Denoting the inlet time dependent junction by j , the volume number 400 by L and the inlet time dependent junction by K , the semi-implicit discretization of equation (87) yields:

$$\rho_j \frac{\partial v_j}{\partial t} + \frac{1}{2} \rho_j \frac{v_L^2 - v_j^2}{\Delta x_j} = -\frac{P_L - P_K}{\Delta x_j} - \text{FWAL}_L v_L \quad (88)$$

Taking into account that $\Delta x_j = \frac{\Delta x_L}{2}$, the upstream pressure can be computed as:

$$P_K = P_L + \rho_j \frac{\partial v_j}{\partial t} \frac{\Delta x_L}{2} + \frac{1}{2} \rho_j (v_L^2 - v_j^2) + \text{FWAL}_L v_L \frac{\Delta x_L}{2} \quad (89)$$

If the boundary conditions in the inlet time dependent junction is specified as a constant velocity, instead of as a constant mass flow, the term with the time derivative in equation (89) is cancelled. The computation of a numerical derivative is then avoided. This little change does not affect the essence of Zhang's problem.

The momentum equation (89) was implemented through RELAP5 control variables to feed-back the pressure into the upstream TDV. The following results were obtained.

Figures 69 to 72 show the results of postprocessing the pressure, the volume, the vapor mass and the vapor non-equilibrium mass respectively. The agreement between both sides of the respective equations is excellent. A discrepancy between both sides of the pressure drop equation, represented in figure 73, is observed during the first time step. The cause is the joint effect of the discretization and the initialization of the velocities to zero values. The system spend this first time step to reach the inlet velocity specified through the TDJ.

The quality of the solution obtained by feeding the pressure back is confirmed by the good behavior of other variables, such as the temperature in volume 400 shown in figure 74. This temperature tends to that in the inlet TDV as expected. This behavior is not observed when using the PV term, as shown in figure 65. The enthalpy in volume 400 also approaches the value in the inlet TDV, according to figure 75. The density in figure 76 behaves in a similar way. It can be realized in figure 77 that the pressure is almost identical in the three volumes, because the frictional pressure drop is negligible as compared to the global system pressure.

4.5 Full plant model. Flow mixing problem

4.5.1 Introduction

In this last example a transient in a full plant model is analyzed. The transient, which is one of the sequences obtained in a Probabilistic Safety Analysis, consists in a steam generator tube rupture with HPIS failure, CVCS failure, accumulator success, steam dump success and RHRS success.

The subsystem to be analyzed is the downcomer in one of the steam generators, since the transient seems to be very demanding in that component. Flow reversal can be observed in figure 78.

Figure 79 shows the analysis of the global pressure equation in the *downcomer*. The comparison between both sides of the equation is completely unsatisfactory. On the other hand, the analysis of the liquid volume equation (see figure 80), is acceptable. The analysis of the vapor and liquid mass equations (figures 81 and 82) is also satisfactory except for a certain offset in the vapor, which can be attributed to a certain spike in the derivative of that quantitie. A

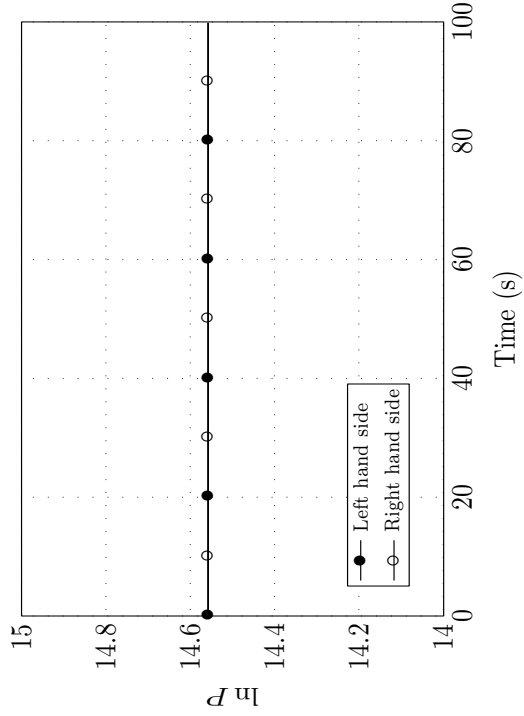


Figure 69: Pressure equation with momentum calculation.

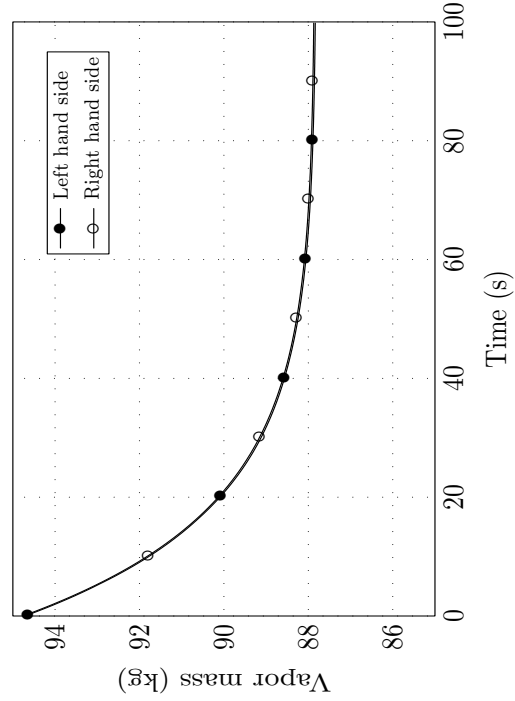


Figure 71: Mass equation with momentum calculation.

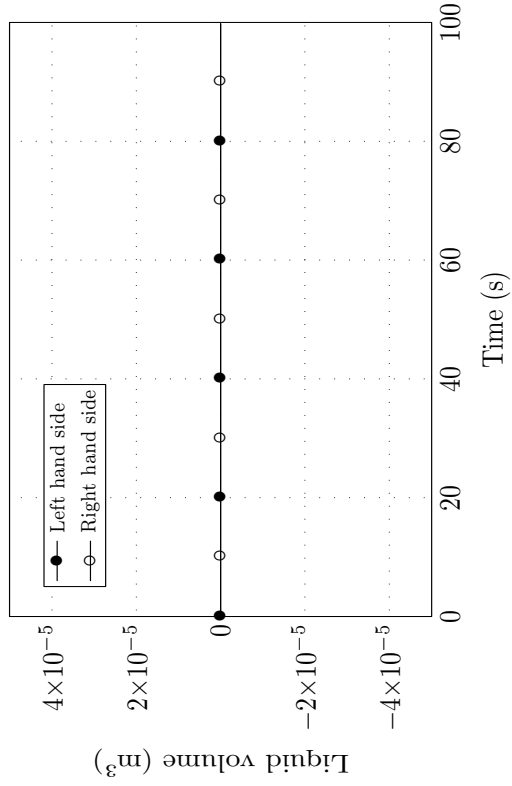


Figure 70: Liquid volume equation with momentum calculation.

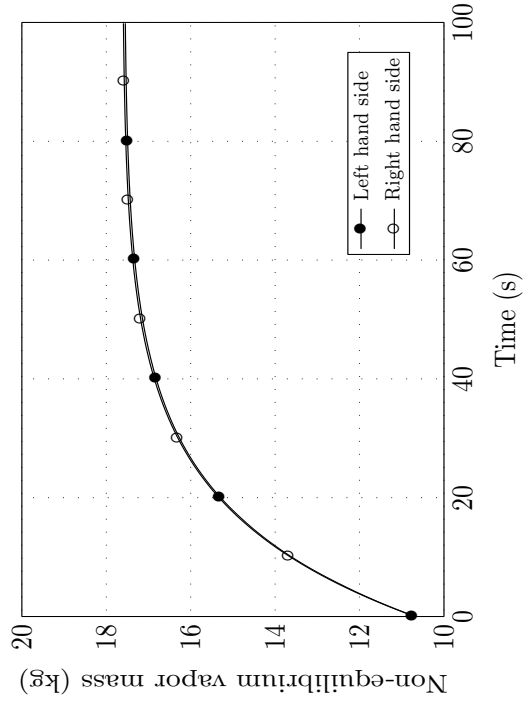


Figure 72: Non-equilibrium mass equation with momentum calculation.

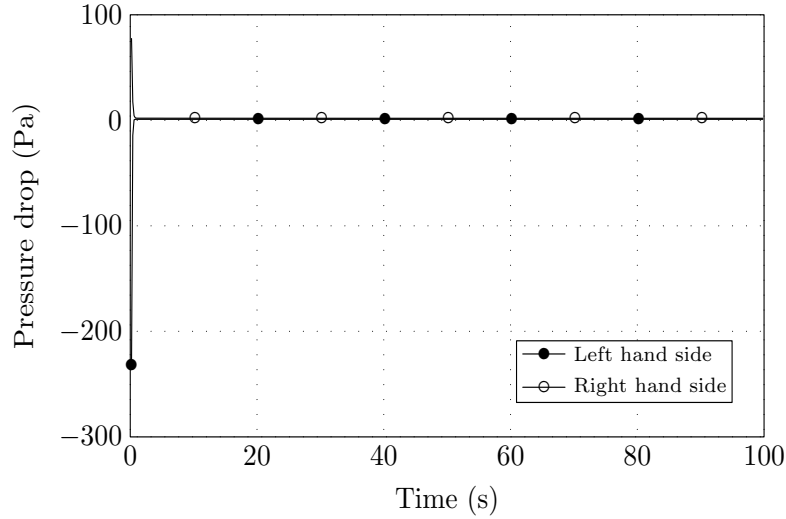


Figure 73: Pressure drop equation with momentum calculation.

direct comparison of the derivatives of both sides of the equation would be more convincing. The analyses of the non-equilibrium vapor and liquid mass equations (figures 83 and 84) is not satisfactory either, as they are affected by the pressure equation. Figure 85 shows the terms of the pressure and liquid volume equations. The inlet vapor volumetric flow and the vapor swelling are the dominant terms. Both terms tend to compensate each other.

A possible explanation for the results obtained in the analysis of the pressure equation is the flow mixing that takes place at the downcomer inlet. The feedwater, separator and recirculation flows are mixed at that point. The three flows have very different thermodynamic properties. To prove this hypothesis, the same analysis have been undertaken again excluding the first control volume in the downcomer. The resulting subsystem is strictly one-dimensional, with one inlet junction and one outlet junction.

Figure 86 shows that the pressure analysis is more satisfactory when removing the first control volume. Similar comments apply to the liquid volume and vapor and liquid mass equations (figures 87 to 89). However, fulfillment of the non-equilibrium mass equations (figures 90 and 91) is not properly achieved yet. The pressure equation terms (figure 92) resemble those of the analysis with the whole downcomer.

The results seem to confirm that flow mixing is not properly treated by the RELAP5 code. The following lines will try to explain why the RELAP5 treatment of flow mixing is not satisfactory.

The one-dimensional formulation of two-phase flow dynamics allows to solve problems in pipes with two well defined ends (i.e., inlet and outlet). In this kind of pipes the direction along the pipe plays the role of the independent variable in a parametric representation, ranging from 0 to L , where the latter is the whole length of the pipe. In the case of flow mixing, where n pipes converge into a single one as described in figure 93, the overall problem

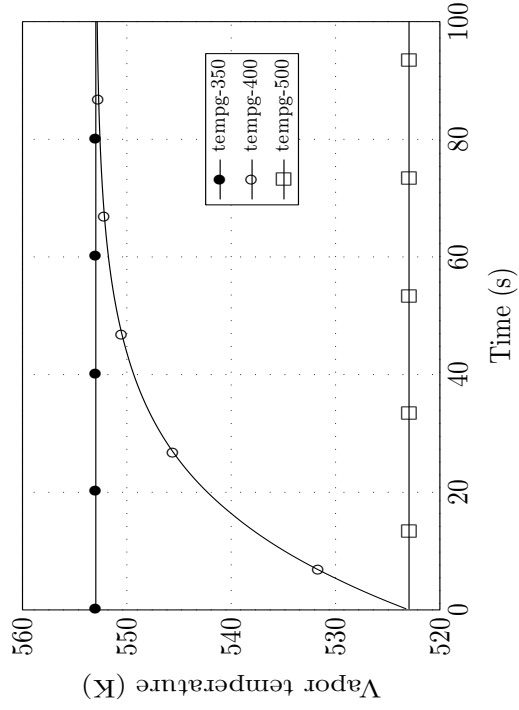


Figure 74: Vapor temperature with momentum calculation.

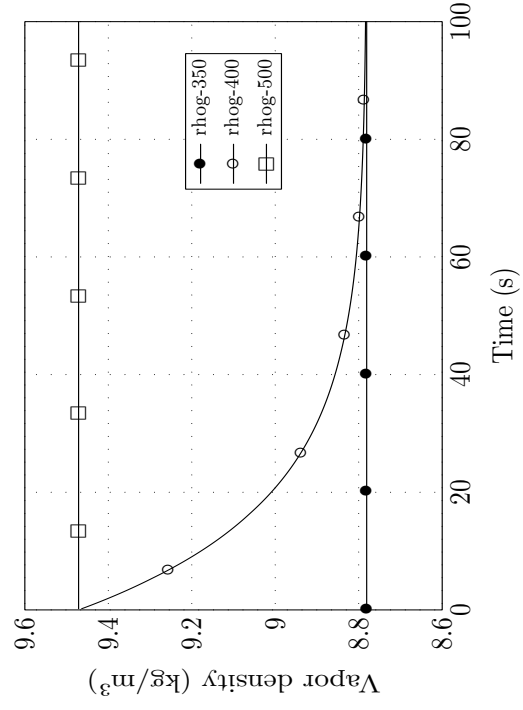


Figure 76: Vapor density with momentum calculation.

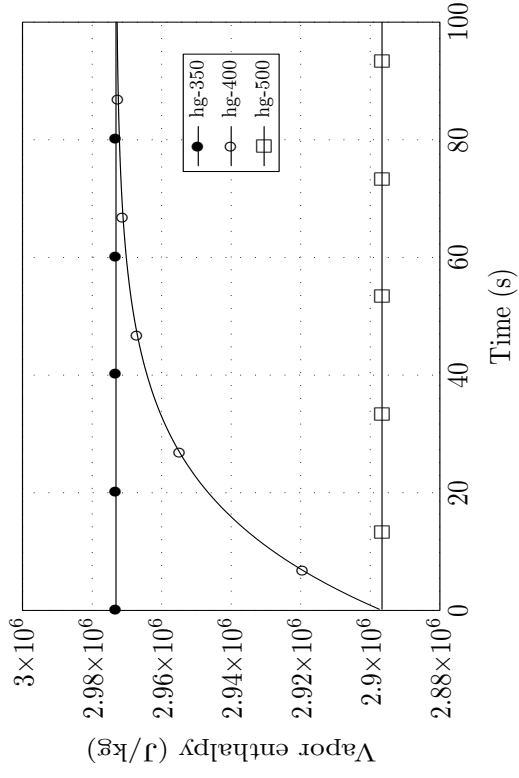


Figure 75: Enthalpy with momentum calculation.

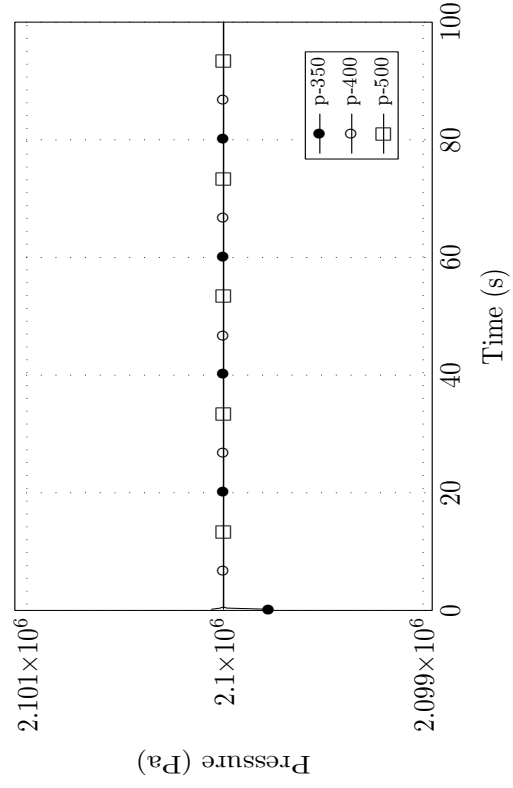


Figure 77: Pressure with momentum calculation.

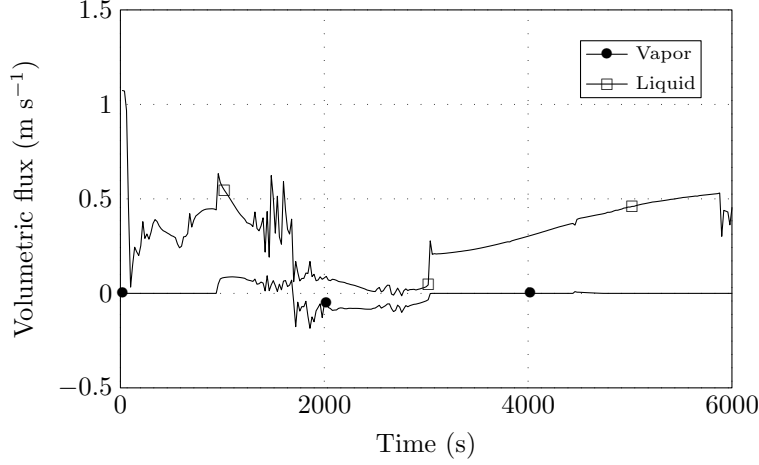


Figure 78: Volumetric flow in junction 5 of the downcomer.

comprises actually $n + 1$ one-dimensional subproblems, each one defined in a range $[0, L_i]$. The $n + 1$ are coupled among them. They are linked in such a way that the boundary conditions supplied to the pipe on the right hand side in figure 93 fulfil the conservation equations. Those boundary conditions must be obtained by integrating on a null volume (i.e., with zero length) the conservation equations (1), (2) and (3). Since the integral of the volumetric sources is considered to be zero in such a null volume, only the convective terms, i.e. those affected by partial derivatives with respect to the spatial variable, contribute to the equations. The integration of a convective term leads to a difference between the sums of inlet and outlet flows. If a convective term has the form $\frac{\partial \Psi v}{\partial x}$, where Ψ is a generic variable and v is the flow velocity, the integral on a zero volume results in $\Psi v|^{out} - \Psi v|^{in}$. Note that integration on a zero volume requires that the partial derivative with respect to time affect all the factors, because terms such as $\phi \frac{\partial \Psi v}{\partial x}$ would not have a well defined integral. As a conclusion, integration on a null volume must only be applied to the so called *conservative form* of the conservation equations.

For the mass equation, the convective terms meet that requirement to yield

$$\sum \alpha_k A \rho_k v_k|^{in} = \alpha_k A \rho_k v_k|^{out} \quad (90)$$

The internal energy equation includes a term of the form $-P \frac{\partial}{\partial x} (\alpha_k A v_k)$, which cannot be integrated directly. I.e. the internal energy equation is not a conservative form. In order to introduce the pressure into the gradient, the kinetic energy equation is added to the internal energy equation, which results from multiplying the momentum equation by the velocity.

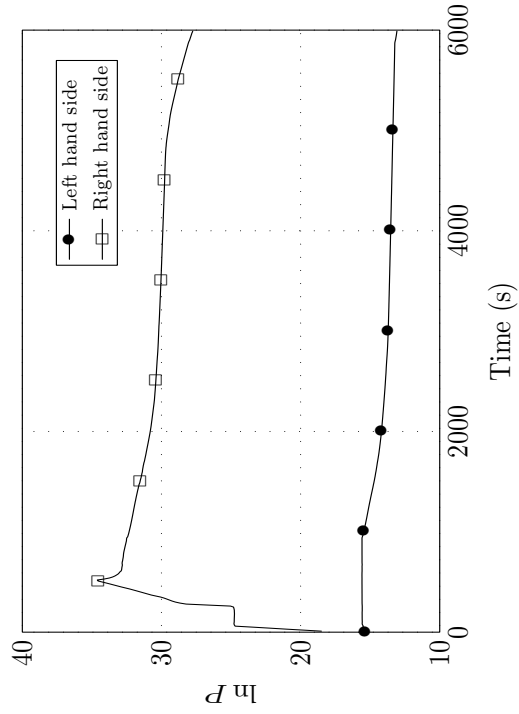


Figure 79: Analysis of the pressure equation in the downcomer.

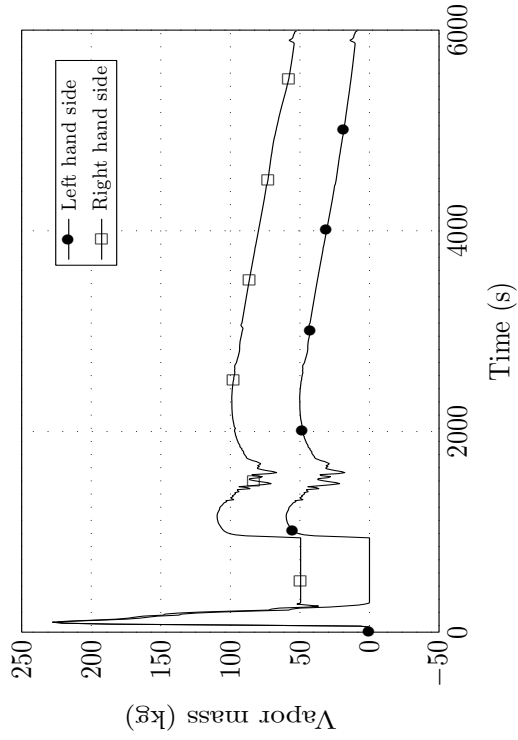


Figure 81: Analysis of the vapor mass equation.

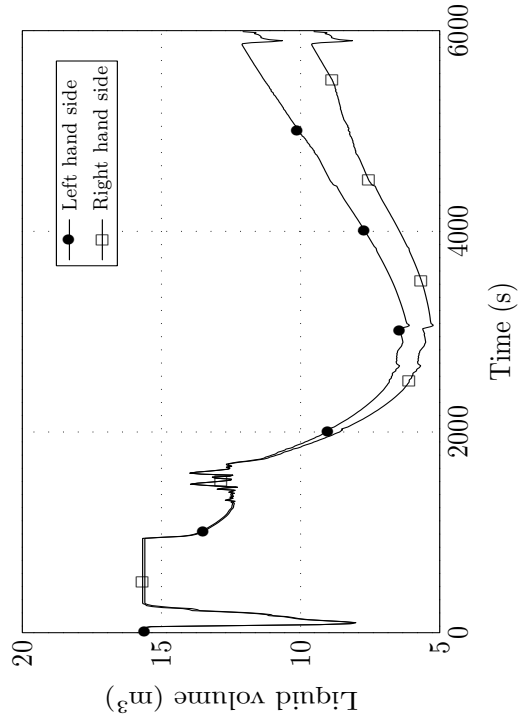


Figure 80: Analysis of the liquid volume equation in the downcomer.

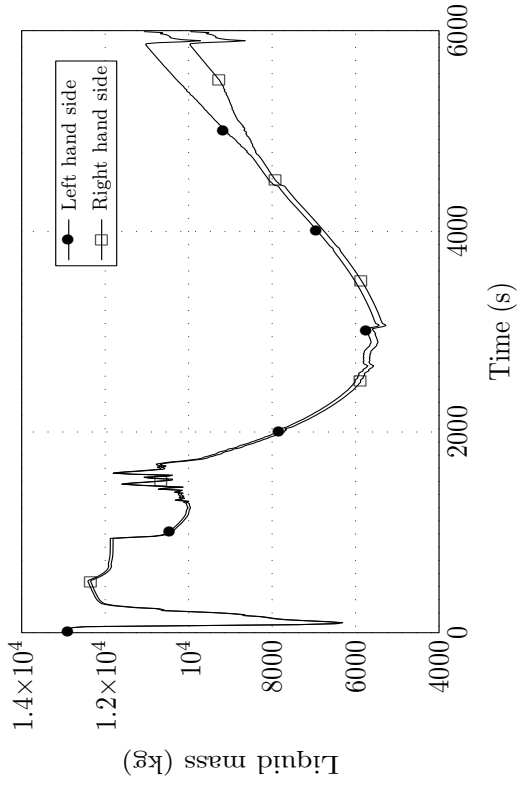


Figure 82: Analysis of the liquid mass equation in the downcomer.

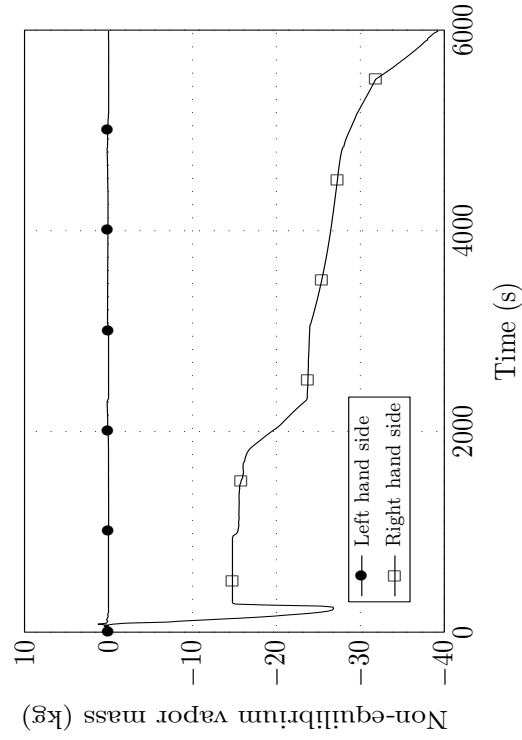


Figure 83: Analysis of the non-equilibrium vapor mass equation in the downcomer.

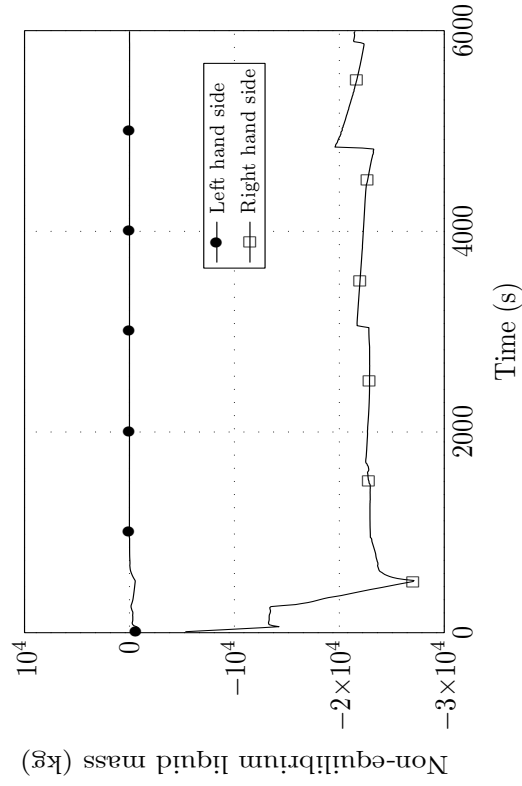


Figure 84: Analysis of the non-equilibrium liquid mass equation in the downcomer.

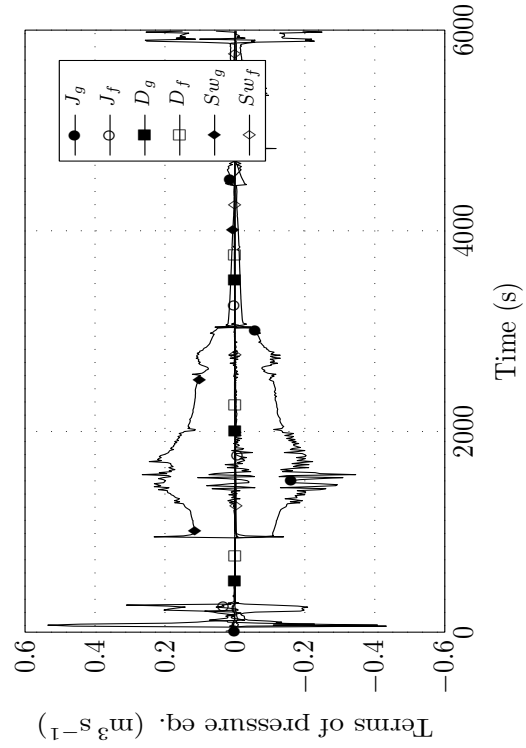


Figure 85: Terms of the pressure equation in the downcomer.

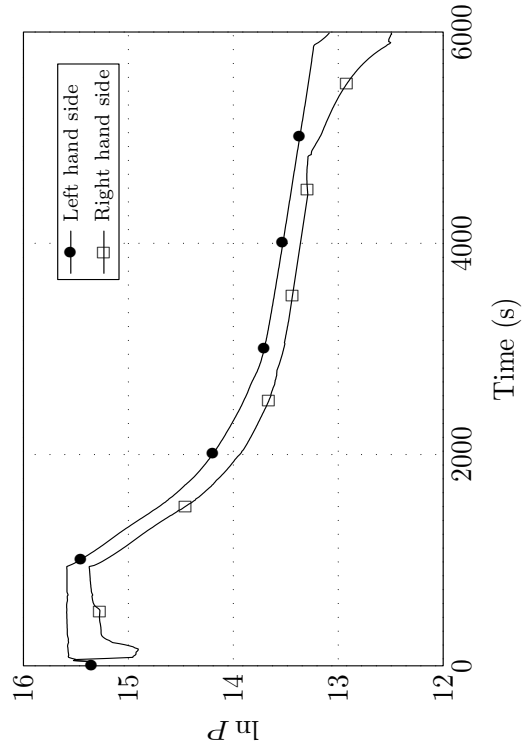


Figure 86: Analysis of the pressure equation in downcomer subsystem.

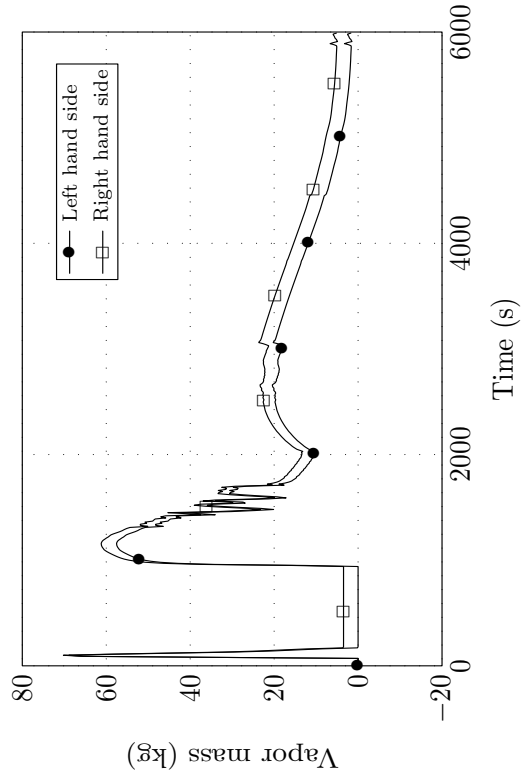


Figure 88: Analysis of the vapor mass equation in downcomer subsystem.

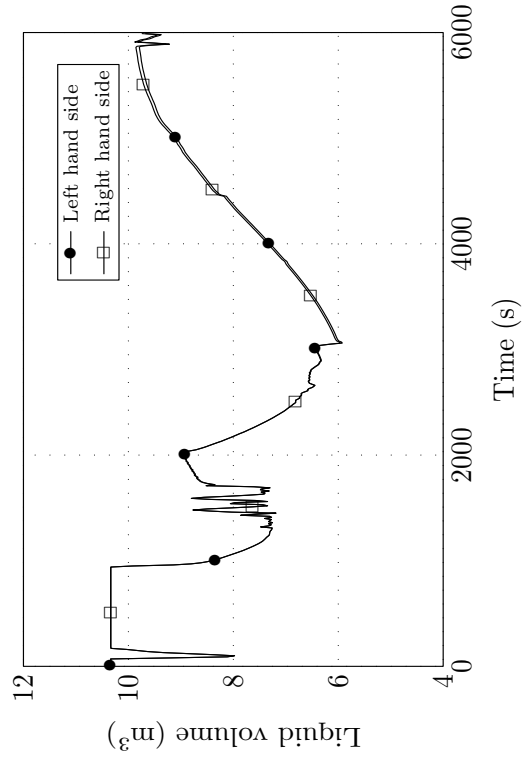


Figure 87: Analysis of the level equation in downcomer subsystem.

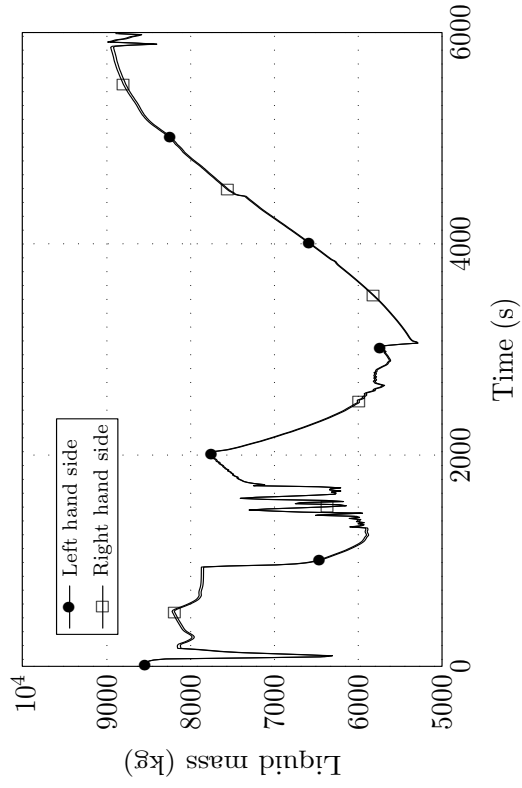


Figure 89: Analysis of the liquid mass equation in downcomer subsystem.

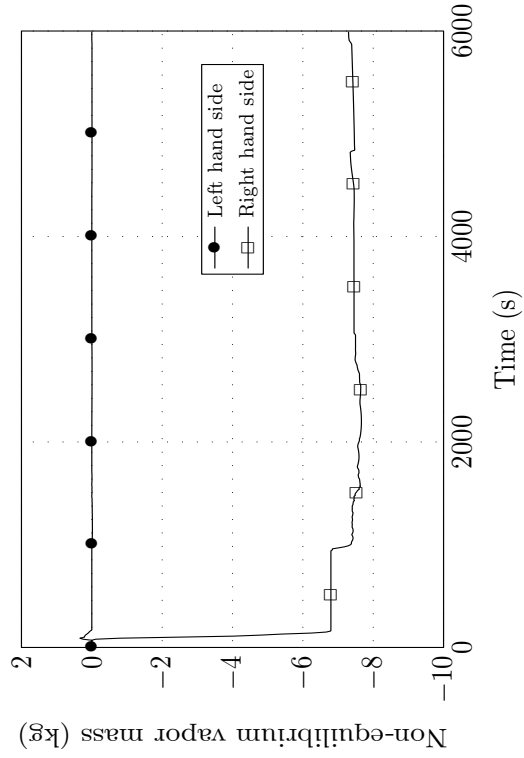


Figure 90: Analysis of the non-equilibrium vapor mass equation in downcomer subsystem.

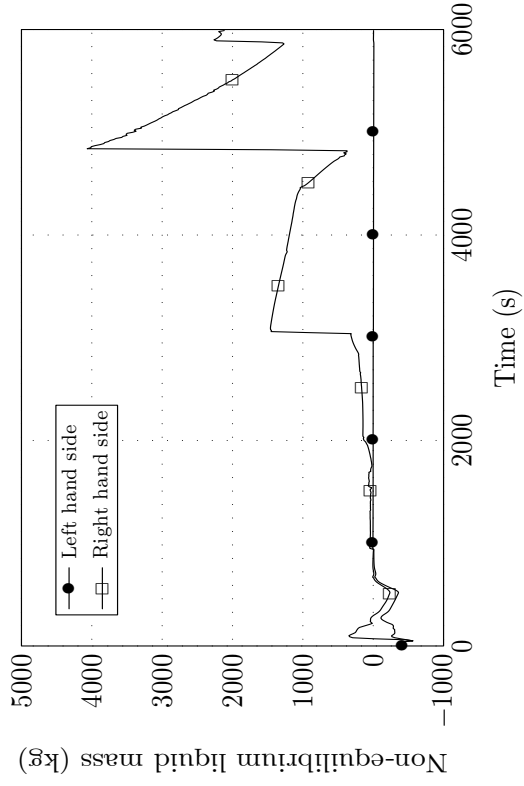


Figure 91: Analysis of the non-equilibrium liquid mass equation in downcomer subsystem.

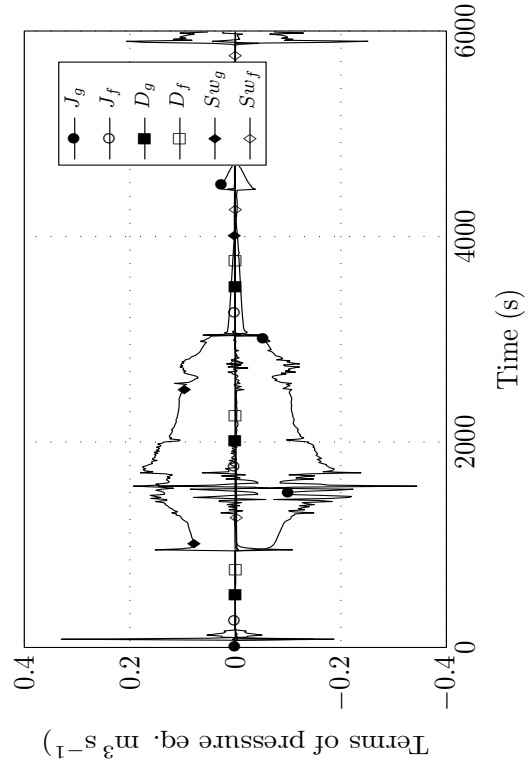


Figure 92: Pressure equation terms in downcomer subsystem.

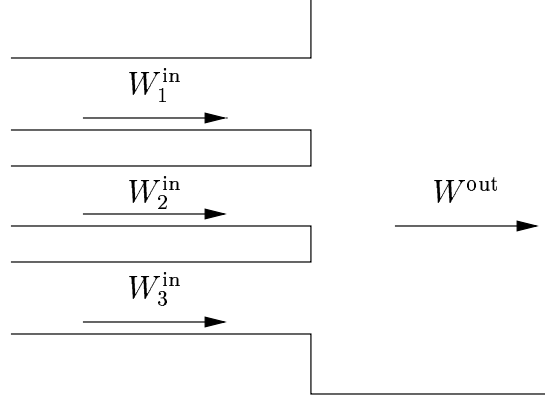


Figure 93: One-dimensional flow mixing.

The kinetic energy equation is as follows:

$$\begin{aligned}
\frac{1}{2}\alpha_k A \rho_k \frac{\partial v_k}{\partial t} + \frac{1}{2}\alpha_k A \rho_k \frac{\partial v_k^2}{\partial x} = & -\alpha_k A v_k \frac{\partial P}{\partial x} + \alpha_k \rho_k B_x A v_k \\
& - \alpha_k \rho_k A F W_k v_k^2 + A \Gamma_k v_i v_k - \alpha_k \rho_k A F I_k (v_k - v_{k'}) v_k \\
& - C \alpha_k \alpha_{k'} \rho_m A \left[\frac{\partial}{\partial t} (v_k - v_{k'}) + v_{k'} \frac{\partial v_k}{\partial x} - v_k \frac{\partial v_{k'}}{\partial x} \right] v_k \\
& - \frac{1}{2} K \alpha_k \rho_k v_k^3 \delta(x - x_0)
\end{aligned} \tag{91}$$

where the continuity equation (1) has been taken into account.

Adding (91) to (3), the term with the pressure takes the integrable form $-\frac{\partial}{\partial x}(\alpha_k P A v_k)$. The mixing condition is then:

$$\begin{aligned}
\sum \alpha_k A \rho_k \left(u_k + \frac{1}{2} v_k^2 \right) v_k \Big|_{\text{in}} - \alpha_k A \rho_k \left(u_k + \frac{1}{2} v_k^2 \right) v_k \Big|_{\text{out}} \\
= - \sum \alpha_k P A v_k \Big|_{\text{in}} + \alpha_k P A v_k \Big|_{\text{out}}
\end{aligned} \tag{92}$$

Taking into account that $h = u + P\vartheta$, the previous equation can be written as

$$\sum \alpha_k A \rho_k \left(h_k + \frac{1}{2} v_k^2 \right) v_k \Big|_{\text{in}} = \alpha_k A \rho_k \left(h_k + \frac{1}{2} v_k^2 \right) v_k \Big|_{\text{out}} \tag{93}$$

The momentum equation (2) cannot be directly integrated on a null volume, since a term of the form $-\alpha_k A \frac{\partial P}{\partial x}$ is found on the right hand side. The factors α_k and A , which lie inside the partial derivative in the original three-dimensional equation, are factored out in the one-dimensional form by using the Gauss divergence theorem:

$$\alpha_k A \frac{\partial P}{\partial x} = \frac{\partial}{\partial x}(\alpha_k A P) + \lim_{\Delta x \rightarrow 0} \frac{1}{\Delta x} \int_{A_I} P n_x dA + \lim_{\Delta x \rightarrow 0} \frac{1}{\Delta x} \int_{A_W} P n_x dA \tag{94}$$

where A_I is the interfacial area within the pipe interval Δx , A_W is the wall area in the same interval, and n_x is the projection of the unit normal vector to the surface (either interface or wall) onto the x axis. Even assuming that the interfacial term is zero, the wall term would still remain. In case of flow mixing, or discontinuous area change, the computation of that term would remain undetermined. If the discontinuity is seen as the limit case of a series of problems with ever increasing slope in the representation of the cross-sectional area with respect to the x direction, the final result would depend on the infinite possible ways of approximation. A relationship between inlet and outlet properties can be found for the mass and total energy equations because the terms with spatial derivative represent fluxes (i.e. they include the factor Av_k). Since the velocity is parallel to the wall, the flux through the wall is always zero. On the other hand, the term with the spatial derivative in the momentum equation is not a flux and thus the corresponding wall integral is not zero.

As a conclusion, it is impossible to rigorously describe flow mixing or abrupt area change problems with the one-dimensional formulation. The momentum equation has not been used in the derivation of the global pressure equation anyway, so it does not justify the results obtained in the analysis of the downcomer. The previous study just tries to highlight that extreme care must be taken with flow mixing in order to achieve proper conservation of the thermodynamic variables. In the case of the momentum equation, the hypotheses to calculate the local pressure drop in a flow mixing or abrupt area change must be clearly specified. Current codes do not pay much attention to this issue, since the local pressure drops are used to tune the circuit flow.

It is time to explain now why the pressure analysis is not satisfactory when flow mixing is present. The RELAP5 manual does not refer anywhere to flow mixing. The discontinuities caused by merging flows with different thermodynamic properties at a sharp point, or by abrupt area changes, are not taken into account. When a control volume is attached to several inlet junctions, RELAP5 computes the internal energy convective term by adding the energy fluxes in each junction. Each of these fluxes has the form $\dot{\alpha}_{k,j}^n (\rho_{k,j}^n u_{k,j}^n + P_L^n) v_{k,j}^{n+1}$. The donor cell quantities are referred to the upstream side of the junction. In order to avoid the discontinuities in the junctions, they should be referred to the downstream side. Of course, the conservation equations should be used to compute the downstream variables as a function of the upstream values. This treatment of the convective terms does not preserve the energy balance rigorously, what justifies the results obtained in the pressure analysis.

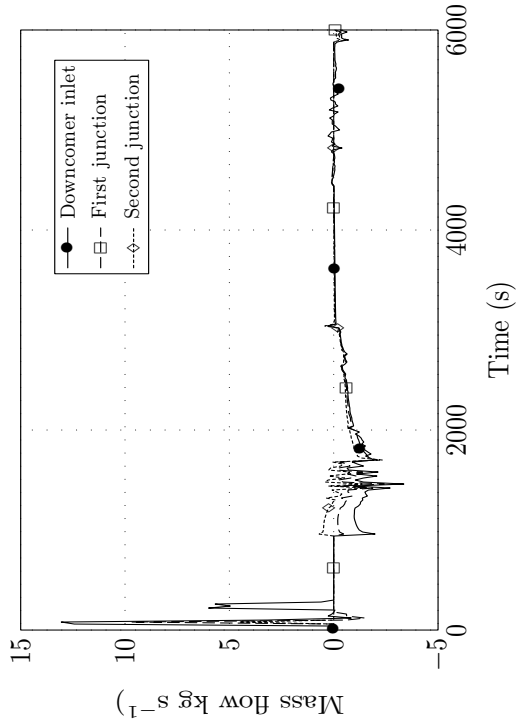


Figure 94: Vapor mass flows in the downcomer.

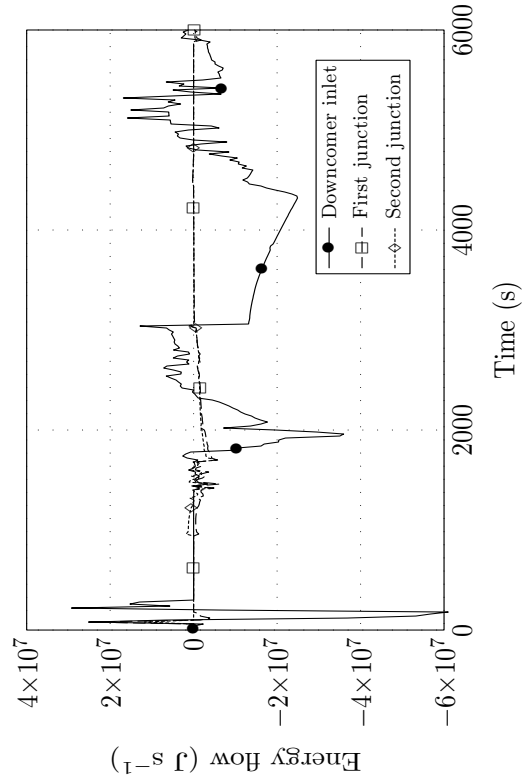


Figure 96: Vapor energy flows in the downcomer.

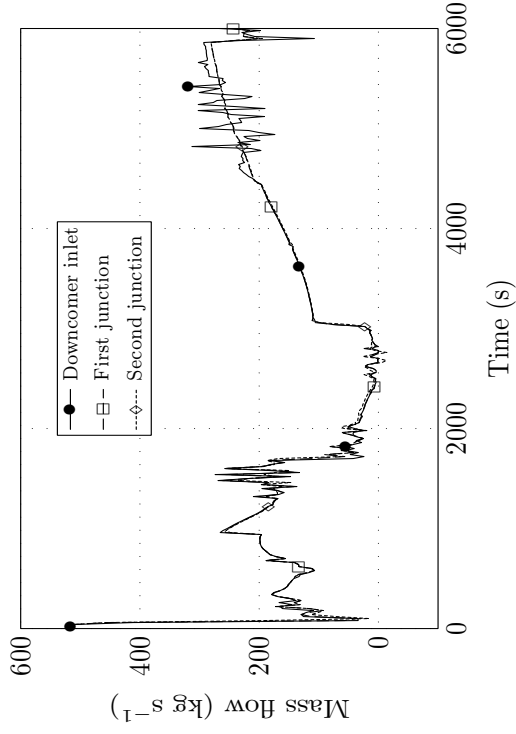


Figure 95: Liquid mass flows in the downcomer.

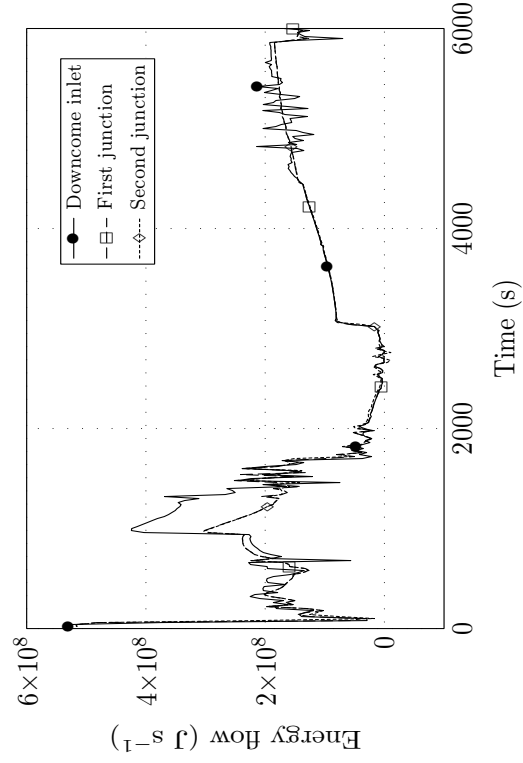


Figure 97: Liquid energy flows in the downcomer.

5 CONCLUSIONS

This report demonstrates the convenience of utilities for analysis and postprocess of computer codes. The utility has been particularly successful in the following areas:

- Correct interpretation of the results in terms of models more suitable to explain the evolution of certain variables, such as the global pressure (29) and the liquid volume (31) equations. These equations have been used in application example 4.3 to refute a classical argument to explain the results obtained when the secondary circuit of a PWR is broken.
- Diagnosis of errors, which may be due to several causes. Example 4.2 shows how the analysis tool has been used to identify a user error in the input file. Case 4.4 illustrates how the tool can reveal a code misconception that affects not only RELAP5 but also TRAC-P [93] and TRAC-B [84]. RETRAN03 [13] developers seem to have solved that error successfully, since the input file only requires the velocity and the enthalpy of the fluid as inlet boundary conditions. Furthermore, the tools pointed out in example 4.5 the problems that may arise when mixing flows with different thermodynamic properties.
- Validation of alternative models. Those models may be more simplified, with the aim of speeding up the overall calculation or simplifying the resolution scheme, as demonstrated in example 4.3.4 with the non-equilibrium mass model derived in section 2.5.

As a conclusion, the development of a new generation of computer codes with estimation of error upper bounds is highly recommended. This is the only way to scientifically ensure the quality of the results.

6 RECOMMENDATIONS FOR FUTURE DEVELOPMENTS

6.1 Analysis of other codes and models

Verification of the solutions computed by simulation codes by means of alternative balance equations is a subject open to research. Any other equation rigorously derived from the original equations can be added to enhance the tool. Balance equations derived in section 2 are just a sample of the large number of possibilities offered by this technique.

The heat structures and the power generation section seem to be natural candidates for the extension of the postprocessing ideas.

6.2 Identification of lower order models

If transient dynamics is very involved, a previous simplification of the models may be necessary. Section 2.5.3 is a good example of this kind of simplifications. It shows how simpler models, with an uncoupled phase, can be derived from the non-equilibrium mass equations. Many other simplified models, using a number of equations between 3 and 6, are found in the history of this discipline [77]. Most of them can be demonstrated to be valid under some assumptions. Validation of those models in certain regions of operation can be performed with techniques similar to those shown in this work.

A typical application of simplified models is the Probabilistic Safety Assessment (PSA) of nuclear power plants. The multifarious sequences that can stem from an initiating event in a level 1 PSA demand the use of simplified, conservative models, in order to decrease the overall computation time [46].

However, some other applications require even more simplified thermal hydraulic dynamics, so it can be integrated with other type of phenomena. For instance, new phenomena arise in level 2 PSA, derived from the degraded core as a consequence of level 1 PSA. Those phenomena are very often of stochastic, barely known nature, subject to a high degree of uncertainty, resulting in a wide variety of possibilities for further evolution of the sequence. That kind of dynamics can hardly be incorporated into the risk calculations, and very specially to the delineation of the so called Accident Progression Event Trees (APET) [21, 10].

Techniques to be used in those cases consist mainly in model identification. A general mathematical form, dependent on some coefficients, is assumed for the dynamic evolutions. The coefficients are tuned to fit a sufficient number of transients precalculated with *best estimate* codes such as RELAP5 or MELCOR. This process can be termed as identification or training of the system. Once the system is trained, it is used to compute non precalculated transient evolutions. Choice of format as physically meaningful as possible is essential. At this point, Dynamic Systems Theory may be helpful.

For example, system dynamics can be regarded as comprising several time intervals, within each one the system can be linearized (piecewise linearization). It can be demonstrated that the behavior of any system can be approximated this way, with the possible exceptions of chaotic systems or presence of discontinuities caused by explosive phenomena. The main drawback of this approach is that the order of the linearized system may eventually be too high and the range of validation very narrow. The main advantage is that the previous history of the system is collected through the state vector. Some integral representations may obviate the high order problem, at the expense of dealing with the previous history explicitly. Some examples can be found in reference [75].

Rigorous inclusion of those generic PSA models can be done with the help of the latest trends in dynamic reliability [58, 59].

Model identification, in the terms explained above, can be another application of the post-processing techniques shown in this work.

6.3 Development of codes with error control

The postprocessing tools has proven to be more effective in the interpretation of results and identification of programming errors than in the diagnosis of errors due to an inappropriate nodalization or time step choice. Identification of such a type of errors would imply a comparison of the discrete solution provided by the code with an hypothetical analytical solution. The latter is of course unknown. Verification of the discrete solution by replacement into a discretized set of equations seems to be close to a tautology. It would mean that the solution is checked versus the same premises with which it has been obtained. It is not surprising that the solution fulfills the balance equations in every node, although very probably the global analytical equations would not be satisfied.

The discretization, which depends strongly on the code user, is still one of the main contributors to the overall code uncertainty. Moreover, discretization errors can make the *verification* and *validation* processes merge inappropriately. Figures 98 and 99 illustrate this statement.

Verification is the process which ensures that the discrete solution approaches the true analytical solution within a certain tolerance margin. Verification implies matching discrete with continuous mathematical forms. It must be noted that the verification process does not have to do with the capability of the analytical models to represent the real world. Verification of the solution is attempted by very few codes.

The *validation* process can be launched only after a proper verification of the solution. It implies a checking of the capability of the models included in the code to reproduce a set of experimental data. The process consists in matching continuous domain maths with the physical reality. Moreover, the term validation makes reference to a certain experiment for set of experiments. It is not an absolute term as it is often understood. Figure 98 emphasizes that the verification and validation processes must be clearly separated from each other.

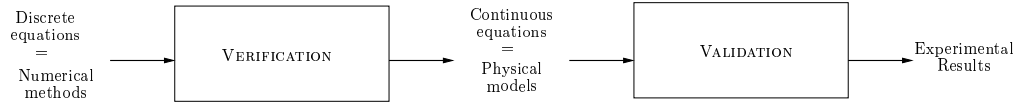


Figure 98: Verification and validation processes.

Since the verification process is seldom undertaken, it is usually merged with validation as shown in figure 99. This means that the solutions computed by the code are directly compared with the experimental data, which is a risky although common practice. If the comparison between code results and experimental data is not satisfactory, a good model could be rejected even if the cause of discrepancies lies in an inadequate nodalization. On the contrary, the analyst may try to improve the solution by refining the noding mesh, although the analytical model might be the cause of the insatisfactory results. It is difficult to accurately determine the source of errors if the verification and validation processes are merged.



Figure 99: Collapsed verification and validation processes.

The verification task can be undertaken by means of the so called precise numerical methods [2,1]. These methods take into account the intrinsic discrete nature of both the numerical methods and the computing machine.

The problem to be solved no longer consists in solving the stages of the numerical method in the user defined computational mesh. On the contrary, the user must specify the discrete points at which the solution is to be observed, as well as the required precision. Note that the observation mesh may not be coincident with the computational mesh necessary to find a solution within the desired tolerance margin. Moreover, the latter depends strongly on the former. The precision of the solution can be known only if an upper bound of the truncation error is estimated. If the error associated to any of the output variables under observation is greater than the desired precision, the transient must be recomputed with a finer spation-temporal nodalization. Although current codes usually consider just one kind of computational mesh, The following types must be discriminated:

- The spatial mesh used to define certain input parameters. For instance, the cross-sectional areas of the pipes. The greater the spatial variations of those parameters are, the finer this mesh must be.
- The spatial mesh used to define the initial conditions at discrete points, which must be finer as long as the initial conditions contain high frequency spatial components.

- The temporal mesh used to define the boundary conditions. The higher the time frequency of the boundary conditions is, the finer the mesh must be.
- The spatio-temporal mesh formed by the points at which the user desires to observe certain output variables.
- The calculational mesh used to solve the numerical method.

For most current codes all the types of discretization described above are the same, even though each of them accounts for very different aspects. If the error associated to any of the desired output variables is larger than the required precision a new calculation must be attempted with a finer spatio-temporal nodalization. If the numerical method is proven to be both consistent and convergent, the new noding scheme should yield smaller truncation errors. The proposed approach needs of course a certain method for automatic mesh generation. The numerical discretization recovers then its strict algorithmical meaning. It does not depend on the user's choice but on the desired output variables and the required precision. The plant nodalization is no longer a contributor to the so called *user effect*. This is restricted just to the choice of optional models.

The estimation of the truncation errors associated to a certain numerical method usually requires the use of higher order derivatives of mathematical functions. An analytical expression of those derivatives is usually unknown when the code is programmed. Derivation of the analytical expressions demands the use of symbolic calculus, which adds an extra degree of complexity into the overall problem.

Besides the truncation error associated to the numerical methods, round-off errors must be controlled as well. Digital computers, being discrete machines, can represent only a subset of the rational numbers and none of the irrational numbers with their native floating-point arithmetic. Although some authors such as [110] state that round-off errors are negligible as compared to truncation errors, several evidences on the contrary have been found. Another evidence of the importance of round-off errors is provided in [69], which shows that different results can be obtained, even with the same machine, when the thermal conductivity of water in presence of non-condensable gases is calculated by linear interpolation based either on the leftmost point,

$$y(x) = y_0 + \frac{y(x_1) - y(x_0)}{x_1 - x_0}(x - x_0)$$

or the rightmost point,

$$y(x) = y_1 + \frac{y(x_1) - y(x_0)}{x_1 - x_0}(x - x_1)$$

Both formulations are completely equivalent. Any discrepancy in the results can only be explained by the different effects of each formulation in the round-off errors.

The *precise numerical methods* focus on the control of any source of error. They require the use of a special type of floating-point arithmetic called *range arithmetic*. Real numbers are represented by data structures such as `struct` in C language or `TYPE` in FORTRAN 90.

Every digit in the mantissa is represented by a native integer type. Another native integer data type accounts for the exponent. The number of digits in the mantissa is not fixed, as it is in ordinary native floating-point arithmetics. An extra integer digit represents an upper bound of the error associated to the floating-point number. For instance, the number $1.5984336\sim 2\text{E-}2$ means that its value lies in the interval $1.5984336\times 10^{-2} \pm 2 \times 10^{-9}$.

Whenever an operation between numbers is performed, the round-off error is accounted for either incrementing the error digit or, if the latter is greater than 9, decrementing the number of digits in the mantissa. It is clear than the greater the number of operations used to compute a number, the smaller the precision is. If the error associated to any of the user required variables is greater than the maximum allowed, the transient must be computed from the beginning, starting with a larger initial precision. The number of digits in the mantissa can be changed by using the dynamic memory allocation capabilities provided by advanced programming languages. With range arithmetic the precision is no longer limited by any machine word length (usually 64 bits for double precision) but by the memory available in the computer. Note that, since the precision is progressively decreased during the course of the transient, a very large initial precision might be required to obtain an output precision of just a few digits.

The use of precise numerical methods yields the following cascade effects:

- Since range arithmetic is implemented through integer data types, and the result of operations between integer numbers is always exact, the results are machine independent. The so called *machine-compiler effect* dissapears.
- Since the working precision is unlimited (although it is always finite), the round-off errors can be as small as desired.
- As a consequence of the item above, nodalizations can be refined as much as necessary to obtain the required precision in the output variables. The refinement implies a greater number of operations and thus greater round-off errors, but they can be compensated by a larger initial precision. The nodalization needed to obtain the desired final precision is automatically chosen by the code, not by the user. The nodalization is not then a contributor to the *user effect*.

7 REFERENCES

- [1] Oliver Aberth. *Precise Numerical Methods Using C++*. Academic Press, 1998.
- [2] Oliver Aberth and Mark J. Schaefer. Precise computation using range arithmetic, via C++. *ACM Transactions on Mathematical Software*, 18(4):481–491, December 1992.
- [3] P.D. Abramson. *Guidebook to Light Water Reactor Safety Analysis*. Hemisphere Publishing Corporation.
- [4] S.N. Aksan, F. D’Auria, and Städtke. User effects on the thermal-hydraulic system code calculations. *Nuclear Engineering and Design*, 145:159–174, 1993.
- [5] H. Asaka, V.G. Zimin, T. Iguchi, and Y. Anoda. Coupling of the thermal-hydraulic code TRAC with 3D neutron kinetics code SKETCH-N. In *OECD-CSNI Workshop on Advanced ‘Thermal-Hydraulic and Neutronic Codes: Current and Future Applications’*. Barcelona, April 2000.
- [6] R. Ashley, M. El-Shanawany, F. Eltawila, and F. D’Auria. Good practices for user effect reduction. Status report. Working draft 2. Technical Report OECD/NEA/CSNI/PWG2 TG on THA, OECD/NEA, April 1998.
- [7] Anthony Baratta. Interface requirements for coupling a containment code to a reactor system thermal hydraulic code. In *OECD/CSNI Workshop on ‘Transient Thermal-Hydraulic and Neutronic Requirements’*. Annapolis, November 1996.
- [8] A. L. Beguelin et al. *PVM-3.3: Parallel Virtual Machine System*. Univ. of Tennessee-Knoxville; Oak Ridge National Laboratory; Amory University, Atlanta, Georgia, 1992.
- [9] A. Bengaouer and G. Geffraye. Three dimensional kinetics coupling to thermal hydraulics. In *OECD/CSNI Seminar on ‘Best Estimate Methods in Thermal-Hydraulics Safety Analysis’*, Ankara, June 1998.
- [10] A. S. Benjamin, V. Behr, D. M. Kunsman, and W. B. Murfin. Containment event analysis for postulated severe accidents: Surry power station, Unit 1. Technical Report NUREG/CR-4700, SAND 86-1135, US Nuclear Regulatory Commission, February 1987.
- [11] B. Brun. Current implementation and future plans on new code architecture, programming language and user interface. In *OECD/CSNI Workshop on ‘Transient Thermal-Hydraulic and Neutronic Requirements’*. Annapolis, November 1996.

- [12] F. Camous, F. Jacq, P. Chatelard, and J.M. Flores. Interface requirements to couple thermal hydraulics codes to severe accident codes: ICARE/CATHARE. In *OECD/CSNI Workshop on 'Transient Thermal-Hydraulic and Neutronic Requirements'*. Annapolis, November 1996.
- [13] Computer Simulation Flow Systems. RETRAN-03: A program for transient thermal-hydraulic analysis of complex fluid flow systems. volume 1: Theory and numerics. Technical Report NP-7450, Volume 1. Research Project 889-10, Electric Power Research Institute, May 1992.
- [14] Michael Corradini. Fundamentals of multiphase flow. Technical report, Department of Engineering Physics. University of Wisconsin, August 1997.
<http://loca.neep.wisc.edu/~sam/2phase/ALL/main/main.html>.
- [15] James F. Cremer, Richard S. Palmer, and Richard E. Zippel. Creating scientific software. *Transactions of the Society for Computer Simulation*, 14(1):37–39, 1997.
- [16] F. D'Auria, E. Cojnick, H. Glaeser, C. Lage, and T. Wickett. Overview of uncertainty issues and methodologies. In *OECD/CSNI Seminar on 'Best Estimate Methods in Thermal-Hydraulics Safety Analysis'*, Ankara, June 1998.
- [17] X. Delhay, Callaert M., C. Schneidesch, Ph. Damas, and L. Vanhoenacker. Coupling of the neutronics code PANTHER with RELAP5. The BELGATOM approach to simulate the core response of non-symmetric reactivity transients. In *Spring 98 CAMP meeting*. Ankara, 1998.
- [18] Tom Downar, Barber Doug, Vince Mousseau, Dave Ebert, and Joe M. Kelly. RELAP5/PARCS. Generalized thermal-hydraulics/neutronic interface. In *Spring 98 CAMP meeting*. Ankara, 1998.
- [19] D.A. Drew. Mathematical modeling of two-phase flow. *Annual Review of Fluid Mechanics*, 15:261–291, 1983.
- [20] S. Mittag et al. Validation of coupled codes for vvers by analysis of plant transients. In *OECD-CSNI Workshop on 'Advanced Thermal-Hydraulic and Neutronic Codes: Current and Future Applications'*. Barcelona, April 2000.
- [21] J.M. Evrard, J.P. Mireau, P. Lardjane, and B. Lucas. KANT: a probabilistic software for level 2 PSA. In *Proceedings of the Fourth International Conference on Probabilistic Safety Assessment and Management*, New York, September 1998.
- [22] Graham E. Fagg and Jack J. Dongarra. PVMPI: An integration of the PVM and MPI systems, April 1996.
<http://www.netlib.org/utk/papers/pvmpi/paper.html>.

- [23] Graham E. Fagg, Jack J. Dongarra, and Al Geist. Heterogeneous MPI application interoperation and process management under PVMPI. In *Euro PVM-MPI conference, Cracow, Poland*, November 1997.
<http://www.netlib.org/utk/papers/pvmmpi97.ps>.
- [24] Fauske & Associates Inc. *MAAP 3.0B, User's Manual*, 1995.
- [25] Lynn R. Feinauer. Compiler issues associated with safety-related software. *Nuclear Technology*, 93:116–122, January 1991.
- [26] T.L. Freeman and C. Phillips. *Parallel Numerical Algorithms*. International Series in Computer Science. Prentice Hall, 1992.
- [27] Al Geist et al. *PVM, Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing*. The MIT press; Scientific and Engineering Computation Series, 1994.
<ftp://netlib2.cs.utk.edu/pvm3/book/pvm-book.ps>.
- [28] Al Geist et al. *PVM3 Users Guide and Reference Manual*. Oak Ridge National Laboratory, September 1994.
<ftp://netlib2.cs.utk.edu/pvm3/ug.ps>.
- [29] G.A. Geist, J.A. Kohl, and P.M. Papadopoulos. PVM and MPI: a comparison of features. *Calculateurs Parallels*, 8(2), May 1996.
<http://www.epm.ornl.gov/pvm/PVMvsMPI.ps>.
- [30] Jean J. Ginoux. *Two-Phase Flows and Heat Transfer*. McGraw Hill, 1978.
- [31] Torbjörn Granlund. *The GNU Multiple Precision Arithmetic Library*, 2.0.2 edition, June 1996.
<ftp://ftp.rediris.es/mirror/gnu/gmp/gmp-2.0.2.tar.gz>.
- [32] B. Gropp, R. Lusk, and A. Skjellum. *Using MPI*. MIT press, 1994.
- [33] A. Hämäläinen, T. Vanttola, and P. Siltanen. Advanced analysis of steam line break with the codes HEXTRAN and SMABRE for Loviisa NPP. In *OECD-CSNI Workshop on 'Advanced Thermal-Hydraulic and Neutronic Codes: Current and Future Applications'*. Barcelona, April 2000.
- [34] V. Hernández, A. Vidal, I. Blanquer, J. Román, Flich S., G Verdú, J. L. Muñoz Cobo, A. Escrivá, X. Sancho, J. Serra, and A. Gómez. Reducción del tiempo de respuesta del código de análisis de transitorios TRAC-BF1 mediante computación de altas prestaciones. In *Proc. of the XXII meeting of the Spanish Nuclear Society, Santander*, October 1996.
- [35] R. Herrero. Specification of initial and boundary conditions in RELAP5/Mod3.2. In *9th CAMP Fall Meeting. Santa Fé*, October 1996.

- [36] Roberto Herrero. A standardized methodology for the linkage of computer codes. application to RELAP5/Mod3.2. Technical Report NUREG/IA-0179, US Nuclear Regulatory Commission. Office of Nuclear Regulatory Research, March 2000.
- [37] Roberto Herrero. Development of a standardised protocol for connection of nuclear safety codes. application to TRAC-M. Technical report, Universidad Politécnica de Madrid, November 2001.
- [38] Roberto Herrero and Enrique Meléndez. Linkage and parallelisation of scientific programs with a modular simulation language. Submitted to Transactions of the Society for Computer Simulation.
- [39] Gad Hetsroni, editor. *Handbook of Multiphase Systems*. Hemisphere Publishing Corporation, 1982.
- [40] B. Holmes, G.R. Kimber, J.N. Lillington, M. Parkes, and R. Page. RELAP5-PANTHER coupled code transient analysis. In *OECD-CSNI Workshop on ‘Advanced Thermal-Hydraulic and Neutronic Codes: Current and Future Applications’*. Barcelona, April 2000.
- [41] E.D. Hughes. An evaluation of state-of-the-art two-velocity two-phase flow models and their applicability to nuclear reactor transient analysis. Technical Report NP-143, EPRI, 1976.
- [42] I.E. Idelchik. *Handbook of Hydraulic Resistance*. Hemisphere Publishing Corporation, 2nd edition, 1986.
- [43] M. Ishii. *Thermo-Fluid Dynamic Theory of Two-Phase Flows*. Eyrolles, 1975.
- [44] K. Ivanov, A. Baratta, and E. Sartori. OECD/NRC MSLB benchmark – a systematic approach to validate best-estimate coupled codes using a multi-level methodology. In *OECD-CSNI Workshop on ‘Advanced Thermal-Hydraulic and Neutronic Codes: Current and Future Applications’*. Barcelona, April 2000.
- [45] J.M. Izquierdo, R. Mendizábal, and J.M. Rey. Fundamentos de ecuaciones de conservación para caudales bifásicos. Technical Report ICAP, Consejo de Seguridad Nuclear, January 1989.
- [46] J.M. Izquierdo, C. Queral, R. Herrero, J. Hortal, M. Sánchez, and R. Muñoz. Role of fast running codes and their coupling with PSA tools. In *OECD-CSNI Workshop on ‘Advanced Thermal-Hydraulic and Neutronic Codes: Current and Future Applications’*. Barcelona, April 2000.
- [47] José María Izquierdo et al. TRETA and TIZONA fast running thermal hydraulic codes. *Annals of Nuclear Energy*, 34:533–549, 2007.

- [48] José María Izquierdo, Javier Hortal, and L. Vanhoenacker. Merits and limits of thermohydraulic plant simulations — towards a unified approach to qualify plant models. *Nuclear Engineering and Design*, 145:175–205, 1993.
- [49] José María Izquierdo and Miguel Sánchez Perea. CAMPEADOR: An implicit simulation language for continuous systems combined with discrete events able to apply protection theory. In *1994 European Simulation Multiconference (ESM'94)*. Barcelona, Spain, June 1994.
- [50] Claes Johnson, Rolf Rannacher, and Mats Boman. Numerics and hydrodynamic stability: Toward error control in computational fluid dynamics. *SIAM J. Numer. Anal.*, 32(4):1058–1079, August 1995.
- [51] A. Jones et al. Experience from the development of the severe accident code system ESTER. In *Proc of the ANS Winter Meeting, Washington*, November 1994.
- [52] Ken Jones. *XMGR5 extensions*. Idaho National Engineering Laboratory, 1995.
- [53] Walter J. Karplus and H. Jürgen Halin. Mathematical modeling and digital computer simulation of engineering and scientific systems. Seminar 94.6, ETH Zurich, June 1997.
- [54] Richard Kesley, William Clinger, and Jonathan Rees. *Revised⁵ Report on the Algorithmic Language Scheme*, February 1998.
<http://www-swiss.ai.mit.edu/scheme-reports/r5rs.ps.gz>.
- [55] N.A. Kheir. *Systems Modeling and Computer Simulation*. Marcel Dekker Inc., New York, 1988.
- [56] G. Kocamustafaogullari and M. Ishii. Foundation of the interfacial area transport equation and its closure relations. *Int. J. Heat Mass Transfer*, 38(3):481–493, 1995.
- [57] Kyrki-Rajamäki. The need of coupled 3D neutronics in DBA and BDBA analyses using conservative or best-estimate approach. In *OECD/CSNI Seminar on 'Best Estimate Methods in Thermal-Hydraulics Safety Analysis'*. Ankara, June 1998.
- [58] Pierre Etienne Labeau and José María Izquierdo. Modelling PSA problems I: The stimulus-driven theory of probabilistic dynamics. *NSE Nuclear Science and Engineering*, 150:115–139, 2005.
- [59] Pierre Etienne Labeau and José María Izquierdo. Modelling PSA problems II: A cell-to-cell transport theory approach. *NSE Nuclear Science and Engineering*, 150:140–154, 2005.
- [60] S. Langenbuch, H. Austregesilo, P. Fomitchenko, U. Rohde, and K. Velkov. Interface requirements to couple thermal-hydraulic codes to 3D neutronic codes. In *OECD/CSNI Workshop on 'Transient Thermal-Hydraulic and Neutronic Requirements'*. Annapolis, November 1996.

- [61] G.S. Lellouche and B. Chexal. A full-range drift flux correlation for vertical flows, rev. 1. Technical Report NP-3969-SR, EPRI, 1986.
- [62] Amable Liñán. *Mecánica de Fluidos*. Sección de Publicaciones de la Escuela Técnica Superior de Ingenieros Aeronáuticos, October 1967.
- [63] Lorie M. Liebrock. Parallelisation and automatic data distribution for nuclear reactor simulations. In *Proc of the OECD/CSNI Workshop on transient thermal-hydraulic and neutronic code requirements; Annapolis*, 5–8 November 1996.
- [64] Sandra Loosemore, Roland McGrath, Andrew Oram, and Richard M. Stallman. *The GNU C Library Reference Manual*, 0.05 edition, August 1993.
<ftp://ftp.rediris.es/pub/gnu/gcc>.
- [65] Alfredo López and José María Sierra. Contraste de resultados obtenidos con RELAP5/Mod3.2 en diferentes entornos de computación. In *First Technical Session CAMP Spain. Barcelona*, April 1997.
- [66] J. Mahaffy and C. Murray. An exterior communications interface for the USNRC consolidated code. In *OECD-CSNI Workshop on ‘Advanced Thermal-Hydraulic and Neutronic Codes: Current and Future Applications’*. Barcelona, April 2000.
- [67] Fernando Merino, Carol Ahnert, and José María Aragonés. Development and validation of the 3-D PWR core dynamics SIMTRAN code. In *Joint Intl Conf on math methods and supercomputing in nuclear applications, Karlsruhe*, April 1993.
- [68] R. L. Moore, C. S. Miller, and C. D. Fletcher. Linking external models with RELAP5 using PVM. Technical report, Idaho National Engineering Laboratory.
- [69] Glen A. Mortensen. One root cause of numerical problems that occur in large codes. In *Reunión CAMP de otoño*, October 1999.
- [70] William L. Oberkampf and Frederick G. Blotner. Issues in Computational Fluid Dynamics. Code verification and validation. Technical Report SAND95-1352, Sandia National Laboratory, 1997.
- [71] OECD Halden Reactor Project. *COPMA-II User’s Manual*, 1996.
- [72] Ohio Supercomputer Center. The Ohio State University. *MPI Primer / Developing with LAM*, November 1996.
<http://www.itl.nist.gov/div895/sasg/LAM/lam61.doc.ps>.
- [73] J. Olive. Thermo-hydraulique diphasique monodimensionnelle. Technical Report 96NB00116, Electricité de France. Direction des Etudes et Recherches, November 1995.

- [74] R. Page and J.R. Jones. Development of an integrated thermal-hydraulics capability incorporating RELAP5 and PANTHER neutronics code. In *OECD/CSNI Workshop on 'Transient Thermal-Hydraulic and Neutronic Requirements'*. Annapolis, November 1996.
- [75] Miguel Sánchez Perea. *Análisis y simulación de redes de subsistemas lineales mediante desarrollos en serie de funciones de Laguerre*. PhD thesis, Universidad Pontificia de Comillas. Escuela Técnica Superior de Ingenieros Industriales (ICAI). Dpto. de Electrónica y Automática, 1996.
- [76] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C. The Art of Scientific Computing*. Cambridge University Press, 2nd edition, 1995.
<http://www.nr.com>.
- [77] César Queral. *Algoritmo para la resolución de las ecuaciones de conservación de fluidos bifásicos. Aplicación a centrales nucleares*. PhD thesis, Universidad Nacional de Educación a Distancia. Ftad. de Ciencias. Dpto. de Física Fundamental, 1995.
- [78] César Queral and M^a. Amparo García Zamorano. 3^{er} informe de avance sobre el acuerdo específico CSN-UPM para el desarrollo de herramientas de evaluación de guías de accidente. Technical report, Departamento de Sistemas Energéticos, UPM, junio 1997.
- [79] César Queral and M^a. Amparo García Zamorano. 3rd Progress report on the specific agreement CSN-UPM for the development of accident management guide evaluation tools. Technical report, Departamento de Sistemas Energéticos, UPM, June 1997.
- [80] César Queral and M^a. Amparo García Zamorano. Final report on the specific agreement CSN-UPM for the development of accident management guide evaluation tools. Technical report, Departamento de Sistemas Energéticos, UPM, December 1997.
- [81] Paul Raines. *Tcl/Tl Pocket Reference*. O'Reilly & Associates, Inc., 1st edition, October 1998.
- [82] V. H. Ransom. A numerical model of two-phase flows. In *Ecole d'Ete d'Analyse Numerique*, May 1989. EGG-EAST-8546.
- [83] V.H. Ransom. Faucet flow, oscillating manometer. In G.F. Hewitt, J.M. Delhay, and N. Zuber, editors, *Multiphase Science and Technology*, pages 465–470. Hemisphere Publishing Corporation, 1987.
- [84] W. H. Rettig and N. L. Wade. TRAC-BF1/MOD1: An advanced best-estimate computer program for BWR accident analysis. User's guide. Technical Report NUREG/CR-4356. EGG-2626. Vol. 2 R4, Idaho National Laboratory, June 1992.

- [85] P.J. Roache. Quantification of uncertainty in computational fluid dynamics. *Annual Review of Fluid Mechanics*, 29:123–160, 1997.
- [86] Arnold D. Robbins. *Effective AWK Programming*, 1.0.3 edition, February 1997.
`ftp://ftp.rediris.es/pub/gnu/gawk/gawk-3.0.3-ps.tar.gz`.
- [87] Kenneth H. Rosen, Richard R. Rosinski, and James M. Farber. *UNIX Sistema V Versión 4*. Osborne McGraw-Hill, 1st edition, 1993.
- [88] T. Rothe. Linking external models to RELAP5: A new era of RELAP5 applications. In *Proc. of the Intl. Conf. on New trends in Nuclear System Thermohydraulics, Pisa*, 1994.
- [89] Joachim Schrod. *The CWEB Class*, November 1995.
`CTAN/macros/latex/contrib/supported/cweb/cweb-user.dvi`.
- [90] Lee A. Segel. *Mathematics Applied to Continuum Mechanics*. Dover Publications Inc.
- [91] R.W Shumway. Athena: Advanced thermal-hydraulic energy network analyzer. Technical Report EG&G-TFM-7925, Idaho National Engineering Laboratory, October 1987.
- [92] J.C. Slattery. *Momentum, Energy and Mass Transfer*. McGraw-Hill, 1972.
- [93] Jay W. et al Spore. TRAC- PF1/MOD2. Volume I . Theory manual. Technical Report LA-12031-M, Vol. I. NUREG/CR-5673, Los Alamos National Laboratory, 1993.
- [94] Zoran Stosic. Enhancing the scope of applications of standard thermal hydraulic codes by linking with others. In *Proc of the ASME International Conference on Nuclear Engineering, New orleans*, volume 1, part. B, 1996.
- [95] The RELAP5 Code Development Team. RELAP5/MOD3 code manual, volume I: Code structure, system models and solution methods. Technical Report NUREG/CR-5535, INEL-95/0174, Idaho National Engineering Laboratory, June 1995.
- [96] The RELAP5 Code Development Team. RELAP5/MOD3 code manual, volume II: User’s guide and input requirements. Technical Report NUREG/CR-5535, INEL-95/0174, Idaho National Engineering Laboratory, June 1995.
- [97] The RELAP5 Code Development Team. RELAP5/MOD3 code manual, volume IV: Models and correlations. Technical Report NUREG/CR-5535, INEL-95/0174, Idaho National Engineering Laboratory, June 1995.
- [98] L.S. Tong and Y.S. Tang. *Boiling Heat Transfer and Two-Phase Flow*. Taylor & Francis, 2nd edition, 1997.
- [99] K. Trambauer. Computer and compiler effects on code results. Technical Report (97)6, OECD/CSNI, 1997.

- [100] Klaus Trambauer. Interface requirements to couple thermal-hydraulic codes to severe accident codes: ATHLET-CD. In *OECD/CSNI Workshop on 'Transient Thermal-Hydraulic and Neutronic Requirements'*. Annapolis, November 1996.
- [101] Paul J. Turner. *ACE/gr Users Manual. Graphics for exploratory data analysis. Software Documentation Series, SDS3, 91-3*. Center for Coastal and Land-Margin Research. Oregon Graduate Institute of Science and Technology, 19600 NW von Neumann Dr., Beaverton, Oregon, 97006-1999, 1993.
- [102] P.J. Turner. *ACE/gr Users Manual. Graphics for exploratory data analysis*. Center for Coastal and Land-Margin Research. Oregon Graduate Institute of Technology, 1993.
- [103] J. Uhle and B. Aktas. USNRC code consolidation and development program. In *OECD-CSNI Workshop on 'Advanced Thermal-Hydraulic and Neutronic Codes: Current and Future Applications'*. Barcelona, April 2000.
- [104] Guido van Rossum. *Python Reference Manual, version 1.5.2*. Corporation for National Research Initiatives (CNRI), October 1998.
<http://www.python.org/ftp/python/doc/postscript-a4-1.5.2a2.tgz>.
- [105] Larry Wall, Tom Christiansen, and Randal L. Schwartz. *Programming Perl*. O'Reilly & Associates, Inc., 2nd edition, September 1996.
- [106] G.B. Wallis. *One-Dimensional Two-Phase Flow*. McGraw Hill, 1969.
- [107] Dan Prelewicz Weidong Wang, Glen Mortensen. Energy conservation in RELAP5. Technical Report SCIE-NRC-392-99, SCIENTECH, Inc., November 1999.
- [108] W. Wulff, H.S. Cheng, S.V. Lekach, and A.N. Mallen. The BWR plant analyzer. Technical Report NUREG/CR-3943. BNL-NUREG-51812, Brookhaven National Laboratory, August 1984.
- [109] W. Wulff et al. The BWR plant analyzer. Final report. Technical Report NUREG CR-3943, 1988.
- [110] Wolfgang Wulff. Computational methods for multiphase flow. In *Second International Workshop on Two-Phase Flow Fundamentals*. Rensselaer Polytechnic Institute, March 1987.

Appendix A. Nomenclature

Latin letters

t	time (s).
x	spatial variable (m).
v	fluid velocity (m s^{-1}).
v_i	interphase velocity (m s^{-1}).
A	cross-sectional area (m^2).
V	volume (m^3).
m	mass (Kg).
W	mass flow (Kg s^{-1}).
P	pressure (Pa).
B_x	projection of gravity onto the pipe direction (m s^{-2}).
FW	wall friction coefficient (s^{-1}).
FI	interphase friction coefficient (s^{-1}).
C	virtual mass coefficient (non-dimensional).
	compressibility (m^3).
K	local pressure drop coefficient (non-dimensional).
HLOSS	form loss coefficient (m s^{-1}). $\text{HLOSS} = \frac{1}{2}K v $
u	specific internal energy (J Kg^{-1}).
H	heat transmission coefficient per unit volumen ($\text{J s}^{-1} \text{m}^{-3} \text{K}^{-1}$),
	control volume height (m).
Q	heat power per unit volume ($\text{J s}^{-1} \text{m}^{-3}$).
h	specific enthalpy (J Kg^{-1}).
T	temperature (K).
s	specific entropy ($\text{J Kg}^{-1} \text{K}^{-1}$).
DISS	dissipation power per unit volume ($\text{J s}^{-1} \text{m}^{-3}$).
L	length of a one-dimensional TH system (m).
N	number of control volumes in a TH system.
M	number of inlet or outlet junctions in a TH system.
J	difference between inlet and outlet volumetric flows ($\text{m}^3 \text{s}^{-1}$).
Sw	expansion/contraction due to appearance/disappearance of phase ($\text{m}^3 \text{s}^{-1}$).
D	dilation of phase, caused by energy interchanges ($\text{m}^3 \text{s}^{-1}$).
Ac	term dominated by pressure gradients ($\text{m}^3 \text{s}^{-1}$).
C_P	constant pressure specific heat capacity ($\text{J Kg}^{-1} \text{K}^{-1}$).
C_ϑ	constant volume specific heat capacity ($\text{J Kg}^{-1} \text{K}^{-1}$).
g	gravity acceleration (m s^{-2}).

Greek letters

α	fraction of phase (non-dimensional).
ρ	density (Kg m^{-3}).
Γ	phase generation rate per unit volume ($\text{Kg m}^{-3} \text{s}^{-1}$).
$\delta(x - x_0)$	Dirac's delta function, centered around the point x_0 (m^{-1}).
ϑ	specific volume ($\text{m}^3 \text{Kg}^{-1}$).
π	thermal expansion coefficient ($\text{m}^3 \text{J}^{-1}$).
γ	isentropic compressibility coefficient (non-dimensional).
λ	compressibility coefficient along the saturation line (non-dimensional).
β	isobaric thermal expansion coefficient (K^{-1}).
η	$\left(\frac{\partial \rho}{\partial P}\right)_u$ ($\text{m}^{-2} \text{s}^2$).
θ	angle between a control volume and the vertical direction (non-dimensional).

Subindices

k	phase (liquid or vapor).
k'	phase opposite to phase k .
i	relative to the interphase.
w	relative to the wall.
b	relative to the fluid bulk.
m	relative to the mixture.
g	relative to vapor.
f	relative to liquid.

Superindices

in	relative to the inlet face.
out	relative to the outlet face.
nq	non- equilibrium.
eq	equilibrium.
n	time step.
sat	relative to the saturation state.

RELAP5 output variables: control volumes

csubpg	$C_{P,g}$ ($\text{J Kg}^{-1} \text{K}^{-1}$).
csubpf	$C_{P,f}$ ($\text{J Kg}^{-1} \text{K}^{-1}$).
betagg	β_g (K^{-1}).
betaff	β_f (K^{-1}).
drgdp	η_g ($\text{m}^{-2} \text{s}^2$).

drfdp	η_f ($\text{m}^{-2} \text{s}^2$).
vvol	volume (m^3).
avol	cross-sectional area (m^2).
p	control volume pressure (Pa).
voidg	vapor fraction (non-dimensional).
voidf	liquid fraction (non-dimensional).
rhog	vapor density (Kg m^{-3}).
rhof	liquid density (Kg m^{-3}).
ug	vapor specific internal energy (J Kg^{-1}).
uf	liquid specific internal energy (J Kg^{-1}).
gammai	vapor generation rate per unit volume in the fluid bulk ($\text{Kg m}^{-3} \text{s}^{-1}$).
gammaw	vapor generation rate per unit volume in the wall
pared	($\text{Kg m}^{-3} \text{s}^{-1}$).
velg	vapor velocity (m s^{-1}).
velf	liquid velocity (m s^{-1}).
q	total heat power transferred to the fluid through the wall (J s^{-1}).
qwg	heat power transferred to vapor through the wall (J s^{-1}).
hig	vapor heat transfer coefficient per unit volume ($\text{J s}^{-1} \text{m}^{-3} \text{K}^{-1}$).
hif	liquid heat transfer coefficient per unit volume ($\text{J s}^{-1} \text{m}^{-3} \text{K}^{-1}$).
satt	saturation temperature at total pressure, as an output variable (K).
tsatt	saturation temperature at total pressure, as a boundary condition (K).
quale	equilibrium quality (non-dimensional).
boron	boron concentration (parts of boron per parts of water).
tempg	vapor temperature (K).
tempf	liquid temperature (K).
fwalgx	wall friction coefficient for vapor phase ($\text{Kg m}^{-3} \text{s}^{-1}$).
fwalfx	wall friction coefficient for liquid phase ($\text{Kg m}^{-3} \text{s}^{-1}$).

RELAP5 output variables : junctions

velgj	vapor velocity (m s^{-1}).
velfj	liquid velocity (m s^{-1}).
mflowgj	vapor mass flow, as a boundary condition (Kg s^{-1}).
mflowfj	liquid mass flow, as a boundary condition (Kg s^{-1}).
voidgj	vapor fraction (non-dimensional).

<code>voidfj</code>	liquid fraction (non-dimensional).
<code>rhogj</code>	vapor density (Kg m^{-3}).
<code>rhofj</code>	liquid density (Kg m^{-3}).
<code>formgj</code>	local pressure drop coefficient for vapor phase (non-dimensional).
<code>formfj</code>	local pressure drop coefficient for liquid phase (non-dimensional).

Appendix B: Partial derivatives of thermodynamic properties

Partial derivatives as a function of C_P , C_ϑ y β

$$\begin{aligned}
 \left(\frac{\partial u}{\partial T}\right)_P &= C_P - P\vartheta\beta \\
 \left(\frac{\partial u}{\partial \vartheta}\right)_P &= \frac{C_P - P\vartheta\beta}{\vartheta\beta} \\
 \left(\frac{\partial T}{\partial s}\right)_{\vartheta} &= \frac{T}{C_\vartheta} \\
 \left(\frac{\partial T}{\partial s}\right)_P &= \frac{T}{C_P} \\
 \left(\frac{\partial \vartheta}{\partial s}\right)_P &= \frac{T\vartheta\beta}{C_P} \\
 \left(\frac{\partial T}{\partial P}\right)_s &= \frac{T\vartheta\beta}{C_P} \\
 \left(\frac{\partial u}{\partial s}\right)_P &= (C_P - P\vartheta\beta)\frac{T}{C_P} \\
 \left(\frac{\partial s}{\partial P}\right)_T &= -\vartheta\beta \\
 \left(\frac{\partial P}{\partial T}\right)_{\vartheta} &= \frac{C_P - C_\vartheta}{T\vartheta\beta} \\
 \left(\frac{\partial P}{\partial \vartheta}\right)_T &= \frac{C_\vartheta - C_P}{T\vartheta^2\beta^2} \\
 \left(\frac{\partial \vartheta}{\partial P}\right)_s &= \frac{C_\vartheta T\vartheta^2\beta^2}{C_P(C_\vartheta - C_P)} \\
 \left(\frac{\partial u}{\partial \vartheta}\right)_T &= \frac{C_P - C_\vartheta - P\vartheta\beta}{\vartheta\beta} \\
 \left(\frac{\partial \vartheta}{\partial h}\right)_P &= \frac{\vartheta\beta}{C_P} \\
 \frac{1}{\gamma} &\equiv \left(\frac{\partial \ln \rho}{\partial \ln P}\right)_s = \frac{PC_\vartheta T\vartheta\beta^2}{C_P(C_P - C_\vartheta)} \\
 \frac{1}{\lambda} &\equiv \left(\frac{d \ln \rho}{d \ln P}\right)_{\text{sat}} = -PT\beta \left[\frac{\vartheta\beta}{C_\vartheta - C_P} + \frac{\vartheta_{fg}}{h_{fg}} \right]
 \end{aligned}$$

Partial derivatives as a function of η , C_P y β

$$\begin{aligned} \left(\frac{\partial \rho}{\partial u}\right)_P &= \frac{\rho^2 \beta}{P\beta - \rho C_P} \\ \left(\frac{\partial u}{\partial P}\right)_\rho &= \eta \frac{\rho C_P - P\beta}{\rho^2 \beta} \\ \left(\frac{\partial P}{\partial s}\right)_\vartheta &= \frac{T\beta\rho^2}{\eta(\rho C_P - P\beta)} \\ \left(\frac{\partial \vartheta}{\partial P}\right)_s &= \frac{\eta(P\beta - \rho C_P)}{C_P \rho^3} \\ \left(\frac{\partial T}{\partial s}\right)_\vartheta &= \frac{T}{C_P} \left(1 - \frac{\rho\beta^2 T}{\eta(P\beta - \rho C_P)}\right) \\ \left(\frac{\partial \vartheta}{\partial s}\right)_u &= \frac{T}{P} \\ \left(\frac{\partial T}{\partial \vartheta}\right)_s &= \frac{T\beta\rho^2}{\eta(P\beta - \rho C_P)} \\ \left(\frac{\partial \vartheta}{\partial s}\right)_T &= -\frac{\eta(P\beta - \rho C_P) - \rho\beta^2 T}{\beta\rho^2 C_P} \\ C_\vartheta &= \frac{\eta C_P (P\beta - \rho C_P)}{\eta(P\beta - \rho C_P) - \rho\beta^2 T} \end{aligned}$$

Appendix C: File babpost.awk

```
# This AWK command file automatically generates the BABIECA input file
# necessary to execute the thermalhydraulic postprocess of a
# given RELAP5/MOD3.2 case.
#
# First of all, you need to write an ASCII file (generically denoted
# as 'infile'), with the format described bellow.
# Words in capital letters must be written literally.
# Lower case words are generic names that make reference to RELAP5
# output files, restart-plot files, volumes to be postprocessed, etc.
# Blanks at both sides of the equal sign '=' are mandatory. The number
# of blanks between words is irrelevant (1 at least, of course).
#
# LINE 1:      FILE = rootword
# LINE 2:      HOST      = host
# LINE 3:      INITTIME  = inittime
# LINE 4:      STARTTIME = starttime
# LINE 5:      ENDTIME   = endtime
# LINE 6:      FREQ      = freq
# LINE 7:      NSYSTEMS  = nsystems
# For each thermalhydraulic subsystem (i = 1 to nsystems):
#   LINE :      SYSNAME  = sysname
#   LINE :      NVOLS    = nvols
#   LINE :      cccvv0000
#   .
#   .
#   .
#   LINE :      cccvv0000
# LINE :      NHEATSTRS  = nheatstrs
# For each heat structure:
#   LINE :      cccg0nn
#   .
#   .
#   .
#
# where:
# 'rootword' is the root name of the RELAP5 input file to be
# postprocessed, which must be appended by the extension '.i'.
# The output file 'rootword.o' must exist prior to the
# execution of this awk file. The restart plot-file will
# be named 'rootword.r'.
# The root name will also be used in the postprocessing
# output files.
# The results of the postprocess of each thermalhydraulic
# subsystem defined in the 'infile' will be written in
# ASCII files with the names 'rootword_sysname.dat'. The results
# The data files 'rootword_cccvv.dat' contain the results of the
# control volume 'cccvv', where ccc stands for the component
# number and vv for the volume number within the component.
# 'host' is the name of the host where RELAP5 executable file is allocated.
# 'inittime' is the transient instant, in seconds, at which the RELAP5
# process is spawned. Useful for restart cases.
# 'starttime' is the transient instant, in seconds, at which the
# postprocess begins.
# 'endtime' is the transient instant, in seconds, at which
# the postprocess ends.
# 'freq' is the frequency used to write output data (in time steps).
# 'nsystems' is the number of thermalhydraulic systems to be postprocessed.
# 'sysname' is the name of the thermalhydraulic system. The results of the
# postprocess of each system will be written in the file
# 'rootword_sysname.dat'.
# 'nvols' is the number of control volumes to be postprocessed in
# the current th subsystem.
```



```

# 'cccvv0000' is a nine digit string denoting the volume(s) to be
# postprocessed.
# The three digit substring 'ccc' refers to the hydrodynamic
# component to which the control volume belongs. The two
# digit substring 'vv' refers to the number of the subvolume
# within the hydrodynamic component (01 for a single volume).
# The last four digits are always 0000.
# 'heatstrs' is the number of heat structures to be postprocessed
# (axial nodes).
# 'cccg0nn' is a seven digit string denoting the heat structure to be
# postprocessed.
# The three digit string 'ccc' stands for the heat structure
# number. Digit 'g' stands for the geometry number (see RELAP5
# manual. The two digit string 'nn' stands for the axial node.
#
# This utility must be executed, from the UNIX prompt, as follows:
# awk -f babpost.awk 'infile' 'rootword.o' > 'rootword.bab'
# where:
# 'infile' is the ASCII file described at the top.
# 'rootword.o' is the RELAP5 output file.
# 'rootword.bab' is the BABIECA file generated by this utility.
#
# Now, you must delete the RELAP5 files 'rootword.o' and
# 'rootword.r', if they exist.
#
# Run pvm with proper host, executable paths and working directories.
#
# To run the thermalhydraulic postprocessor, execute
# BAIECA -n rootword.bab
#
# The variables contained in the output files can be displayed
# with XMGR5.
#
#####
# RELAP5/MOD3.2 postprocessor
#
# Initializing variables.
BEGIN{
# njuns = number of junctions in the problem (internal junctions of
# the systems plus boundaries)
# njuns=0
# newvcount = number of control volumes attached to any of the subsystems
# to be postprocessed, but are not part of them.
# newvcount = 0
# nvolouts = number of output signals per control volume to be postprocessed.
# nvolouts = 28
# njunouts = number of output signals per junction.
# njunouts = 9
# nnewvouts = number of output signals per control volume attached to any
# of the subsystems to be postprocessed, but are not part of them.
# nnewvouts = 1
# TOTVOLS = total number of volumes to be postprocessd
# TOTVOLS = 0
# nhtouts = number of output signals per heat structure
# nhtouts = 5

# PI = 3.1415926535897932
}
# Reading data from 'infile'.
/FILE/{
if(($1 == "FILE") && ($2 == "=")) FILE=$3
getline
if(($1 == "HOST") && ($2 == "=")) HOST = $3
getline
if(($1 == "INITTIME") && ($2 == "=")) INITTIME = $3

```

```

getline
if(($1 == "STARTTIME") && ($2 == "=")) STARTTIME = $3
getline
if(($1 == "ENDTIME") && ($2 == "=")) ENDTIME = $3
getline
if(($1 == "FREQ") && ($2 == "=")) FREQ = $3
getline
if(($1 == "NSYSTEMS") && ($2 == "=")) NSYSTEMS = $3
for(sys=1;sys<=NSYSTEMS;sys++){
    getline
    if($1 == "SYSNAME" && ($2 == "=")) SYSNAME[sys] = $3
    getline
    if($1 == "NVOLS" && ($2 == "=")) NVOLS[sys] = $3
    if(1 == sys){
        initloop = 0
    }
    else{
        initloop += NVOLS[sys-1]
    }
    for(i=1;i<=NVOLS[sys];i++){
        getline
        VOLINSYS[i+initloop]=$1
        newsysvolume = 1
        newvolume = 1
        for(k=1;k<i;k++){
            if(VOLINSYS[k+initloop] == VOLINSYS[i+initloop]){
                newsysvolume = 0
            }
        }
        if(!newsysvolume){
            printf "# TH system with repeated volumes.\n"
            printf "# Subsystem %s. Volumes %s and %s.\n", SYSNAME[sys], \
                VOLINSYS[k+initloop], VOLINSYS[i+initloop]
            stop
        }
        for(k=1;k<=TOTVOLS;k++){
            if(VOLUME[k] == VOLINSYS[i+initloop]){
                newvolume = 0
                break
            }
        }
        if(newvolume){
            VOLUME[++TOTVOLS] = VOLINSYS[i+initloop]
        }
    }
}
getline
if(($1 == "NHEATSTRS") && ($2 == "=")) NHEATSTRS = $3
for(nht=1;nht<=NHEATSTRS;nht++){
    getline
    heatst[nht] = $1
}
}

/~      end time      min.time step    max.time step  option  minor edit /{
getline
getline
MAXDT = $3
NSTEPS = ((ENDTIME-INITTIME)/MAXDT)
}

/OGravitational constant is/{
GRAV = $4
}

```

```

# Searching in 'rootword.o' for all the possible junctions
# attached to the control volumes to be postprocessed.
/^0 Jun\.no\. from vol\. to vol\. liq\.j\.vel/{
  while ($0 !~ /voidfj/){
    getline

# For each line between '0 Jun.no. from vol. ,etc.' and 'voidfj'
# we will check if either 'from vol.' or 'to vol.' is one of our control
# volumes.
    for (i=1;i<=TOTVOLS;i++) {

# 'fstart' and 'tstart' are counters that contain the position
# of the first character to be read from a word that possibly contains
# one of the control volumes.
    fstart=1; tstart=1
    COMPONENT=substr(VOLUME[i],1,3)
    SUBVOLUME=substr(VOLUME[i],4,2)
    DIRECTION=substr(VOLUME[i],9,1)
    if( substr($2,1,1) == "-" ) fstart=2
    if( substr($3,1,1) == "-" ) tstart=2
    pvf=substr($2,fstart, 3)
    pvt=substr($3,tstart,3)
    psvf=substr($2,4+fstart,2)
    psvt=substr($3,4+tstart,2)
    if (pvf == COMPONENT){
      if (psvf==SUBVOLUME) {
        njuns++
        JUNCTION[njuns]=substr($1,1,3) substr($1,5,2) "0000"
        FROMVOL[njuns]=pvf psvf "0000"
        FROMSIGN[njuns]=signi[fstart]
        TOVOL[njuns]=pvt psvt "0000"
        TOSIGN[njuns]=signi[tstart]
        if (substr($2,6+fstart,4) == "0000"){
          INOUTFLAGFROM[njuns]=1
          if (substr($2,4+fstart,2) == "01") INOUTFLAGFROM[njuns]=2
        }
        else {
          INOUTFLAGFROM[njuns]=substr($2,9+fstart,1)
        }
        if (substr($3,6+tstart,4) == "0000"){
          INOUTFLAGTO[njuns]=1
          if (substr($3,4+tstart,2) == "01") INOUTFLAGTO[njuns]=2
        }
        else {INOUTFLAGTO[njuns]=substr($3,9+tstart,1)}
      }
    }
    if (pvt == COMPONENT){
      if (psvt==SUBVOLUME) {
        njuns++
        JUNCTION[njuns]=substr($1,1,3) substr($1,5,2) "0000"
        FROMVOL[njuns]=pvf psvf "0000"
        FROMSIGN[njuns]=signi[fstart]
        TOVOL[njuns]=pvt psvt "0000"
        TOSIGN[njuns]=signi[tstart]
        { if (substr($2,6+fstart,4) == "0000"){
          INOUTFLAGFROM[njuns]=1
          if (substr($2,4+fstart,2) == "01") INOUTFLAGFROM[njuns]=2
        }
        else {INOUTFLAGFROM[njuns]=substr($2,9+fstart,1)}
      }
        if (substr($3,6+tstart,4) == "0000"){
          INOUTFLAGTO[njuns]=1
          if (substr($3,4+tstart,2) == "01") INOUTFLAGTO[njuns]=2
        }
        else {INOUTFLAGTO[njuns]=substr($3,9+tstart,1)}
      }
    }
  }
}

```

```

    }
  }
}
for(k=1;k<njuns;k++){
  if(JUNCTION[k]==JUNCTION[njuns]) njuns--
}
}
}

/^0 Vol no. coord.    flow area    flow length  sp.x. chng.  sp.y. chng.  sp.z. chng./{
while ($0 !~ /Vol no.  i.c.    i.c.value 1  i.c.value 2  i.c.value 3  i.c.value 4/){
  getline
  for(i=1;i<=TOTVOLS;i++) {
    if($1 == VOLUME[i]) {
      HEIGHT[VOLUME[i]]=$7
#      printf "# Height of volume %s: %s\n", VOLUME[i], HEIGHT[VOLUME[i]]
    }
  }
}
}

/^0                                     Input data for heat structure-geometry/{
  if (NHEATSTRS > 0){
    htgeom = $NF
    while ($0 !~ / number of mesh points/){
      getline
    }
    meshpoints = $NF
    getline
    geomtype = $NF
    while ($0 !~ /Omesh point number, mesh point coordinate/){
      getline
    }
    getline
    for(meshind=1; meshind<=meshpoints; meshind++) {
      meshcoord[meshind] = $(2*meshind)
    }
    sumvolfacts=0.0
    for(meshind=1; meshind<meshpoints; meshind++) {
      volfact[meshind] = (meshcoord[meshind+1] * meshcoord[meshind+1])\
        - (meshcoord[meshind] * meshcoord[meshind])
      sumvolfacts += volfact[meshind]
    }
    for(meshind=1; meshind<meshpoints; meshind++) {
      if(sumvolfacts == 0.0) {
        volfact[meshind] = 1.0 / (meshpoints-1)
      }
      else {
        volfact[meshind] /= sumvolfacts
      }
    }
    while ($0 !~ /Omesh point number, composition number/){
      getline
    }
    getline
    nmp=2
    for(nfields=1; nmp<=meshpoints; nfields+=2) {
      field = $nfields
      subslen = length(field)-1
      provmpoint = substr(field,1,subslen)
      mat = $(nfields+1)
      if (substr(mat,1,1) == "-") {
        mat = substr(mat,2,length(mat))
      }
    }
  }
}

```

```

    for(meshind=nmp; meshind<=provmpoint; meshind++) {
        matnumb[meshind-1] = mat
    }
    nmp = provmpoint+1
}
while ($0 !~ /Omesh point number, source factor/){
    getline
}
getline
nmp=2
sumfactors=0.0
for(nfields=1; nmp<=meshpoints; nfields+=2) {
    field = $nfields
    subslen = length(field)-1
    provmpoint = substr(field,1,subslen)
    factor = $(nfields+1)
    for(meshind=nmp; meshind<=provmpoint; meshind++) {
        srfact[meshind-1] = factor
        sumfactors += factor
    }
    nmp = provmpoint+1
}
for(meshind=1; meshind<meshpoints; meshind++){
    if(sumfactors == 0.0) {
        srfact[meshind] = 1.0 / (meshpoints-1)
    }
    else {
        srfact[meshind] /= sumfactors
    }
}
while ($0 !~ /^OStructure      source      internal source/){
    getline
}
getline
getline
for(i=1; $0 !~ /^OStructure left vol.      left      left/; i++) {
    pinsrc[i] = $3
    getline
}
while ($0 !~ /^OStructure right vol.      right      right      right surface/){
    getline
}
getline
getline
for(i=1; $0 !~ /^O      right      right      right      right/; i++){
    prightsrf[i] = $5
    getline
}
for(nht=1;nht<=NHEATSTRS; nht++) {
    heatstroot = substr(heatst[nht],1,4)
    if (heatstroot == htgeom) {
        if(geomtype != "cylindrical") {
            printf "Heat structure %s is not cylindrical.\n", heatst[nht]
            printf "It cannot be postprocessed."
            exit
        }
        nmpoints[nht] = meshpoints
        htmshpoint = nht * 100
        for(meshind=1; meshind<meshpoints; meshind++) {
            htmshpoint++
            htcoord[htmshpoint] = meshcoord[meshind]
            htmatnumb[htmshpoint] = matnumb[meshind]
            srcfactor[htmshpoint] = srfact[meshind]
            volfactor[htmshpoint] = volfact[meshind]
        }
    }
}

```

```

leftind = nht * 100
lefthtmatind = leftind + 1
rightind = (nht * 100) + 1
righthtmatind = leftind + meshpoints -1
composition[leftind] = htmatnumb[lefthtmatind]
composition[rightind] = htmatnumb[righthtmatind]
htmshpoint = leftind + meshpoints
htcoord[htmshpoint] = meshcoord[meshpoints]
rightindex = leftind + meshpoints
htvol[nht] = htstrhgt[nht]*htcoord[rightindex]*htcoord[rightindex]\
            *PI
i = int(substr(heatst[nht],6,2))
insrcfac[nht] = pinsrc[i]
rightsrf[nht] = prightsrf[i]
}
}
}

/^0                                Data for composition/{
if (NHEATSTRS > 0) {
while ($0 ~ /^0                                Data for composition/) {
    curcomp = $NF
    for(nht=1; nht<=NHEATSTRS; nht++) {
        leftind = nht * 100
        rightind = (nht * 100) + 1
        if((composition[leftind] == curcomp) || (composition[rightind] == curcomp)) {
            while ($0 !~ /olumetric heat capacity/){
                getline
            }
            if($4 == "table") {
                deftype[curcomp] = "table"
                getline
                getline
                getline
                vcppoint = 1
                while ($0 ~ /^ /) {
                    for(i=2; i<= NF; i+=2) {
                        comppoint = (curcomp*100) + vcppoint
                        vcp[comppoint] = $(i-1)
                        vcpy[comppoint] = $i
                        vcppoint++
                    }
                    getline
                }
                tablepoints[curcomp] = --vcppoint
                comppoint = (curcomp*100) + 1
                initstengy[comppoint] = 0.0
                for(vcppoint=2; vcpoint<=tablepoints[curcomp]; vcpoint++) {
                    comppoint = (curcomp*100) + vcpoint
                    difx1x0 = vcp[comppoint]-vcp[comppoint-1]
                    dify1y0 = vcpy[comppoint]-vcpy[comppoint-1]
                    slope[comppoint-1] = dify1y0/difx1x0
                    semisumy1y0 = (vcpy[comppoint]+vcpy[comppoint-1])/2.0
                    initstengy[comppoint] = (difx1x0*semisumy1y0) + initstengy[comppoint-1]
                }
            }
            else if($4 == "function") {
                deftype[curcomp] = "function"
                getline
                getline
                getline
                vcppoint = 1
                while ($0 ~ /^ /) {

```

```

        comppoint = (curcomp*100) + vcppoint
        lowtemp[comppoint] = $1
        hightemp[comppoint] = $2
        a0[comppoint] = $3
        a1[comppoint] = $4
        a2[comppoint] = $5
        a3[comppoint] = $6
        a4[comppoint] = $7
        a_1[comppoint] = $8
        cterm[comppoint] = $9
        if (vcppoint == 1){
            initstengy[comppoint] = 0.0
        }
        else {
            initstengy[comppoint] = (lowtemp[comppoint]-cterm[comppoint])*\\
(lowtemp[comppoint]-cterm[comppoint])*((a1[comppoint]/2)+\\
(lowtemp[comppoint]-cterm[comppoint])*((a2[comppoint]/3)+\\
(lowtemp[comppoint]-cterm[comppoint])*((a3[comppoint]/4)+\\
(lowtemp[comppoint]-cterm[comppoint])*((a4[comppoint]/5))))+\\
(a0*lowtemp[comppoint])+a_1[comppoint]*log(lowtemp[comppoint]-cterm[comppoint])-\\
initstengy[comppoint-1]
        }
        vcppoint++
        getline
    }
    funcpoints[curcomp] = --vcppoint

}
else if(($4 == "is") && ($5 == "constant")) {
    deftype[curcomp] = "constant"
    comppoint = (curcomp*100) + 1
    vcpy[comppoint] = $7
    getline
}
}
else getline
}
}
}
}

END{
    for(i=1;i<=njuns;i++) {
        extraval = 1
        for(j=1;j<=TOTVOL[j];j++) {
            if(FROMVOL[i] == VOLUME[j]) {
                extraval = 0
                break
            }
        }
        if (extraval == 1) {
            ++newvcount
            NEWVOL[newvcount] = FROMVOL[i]
        }
    }
    for(i=1;i<=njuns;i++) {
        extraval = 1
        for(j=1;j<=TOTVOL[j];j++) {
            if(TOVOL[i] == VOLUME[j]) {
                extraval = 0
                break
            }
        }
        if (extraval == 1) {
            ++newvcount

```

```

        NEWVOL[newvcount] = TOVOL[i]
    }
}

trtouts = TOTVOLS*nvolouts+njuns*njunouts+newvcount*nnewvouts\
+NHEATSTRS*nhtouts

# VOLPOS["vvol"] = 1
# VOLPOS["avol"] = 2
# VOLPOS["p"] = 3
# VOLPOS["uf"] = 4
# VOLPOS["ug"] = 5
# VOLPOS["voidf"] = 6
# VOLPOS["voidg"] = 7
# VOLPOS["rhof"] = 8
# VOLPOS["rhog"] = 9
# VOLPOS["satt"] = 10
# VOLPOS["tempf"] = 11
# VOLPOS["tempg"] = 12
# VOLPOS["velf"] = 13
# VOLPOS["velg"] = 14
# VOLPOS["betaff"] = 15
# VOLPOS["betagg"] = 16
# VOLPOS["csubpf"] = 17
# VOLPOS["csubpg"] = 18
# VOLPOS["hif"] = 19
# VOLPOS["hig"] = 20
# VOLPOS["gammaw"] = 21
# VOLPOS["gammal"] = 22
# VOLPOS["qwf"] = 23
# VOLPOS["qwg"] = 24
# VOLPOS["drfdp"] = 25
# VOLPOS["drgdg"] = 26
# VOLPOS["fwalfx"] = 27
# VOLPOS["fwalgx"] = 28

# JUNPOS["velfj"] = 1
# JUNPOS["velgj"] = 2
# JUNPOS["voidfj"] = 3
# JUNPOS["voidgj"] = 4
# JUNPOS["rhofj"] = 5
# JUNPOS["rhogj"] = 6
# JUNPOS["ajun"] = 7

# NEWVPOS["p"] = 1

# HEATPOS["htrnrn"] = 1
# HEATPOS["htpown"] = 2
# HEATPOS["htsrfn"] = 3
# HEATPOS["http"] = 4 left side
# HEATPOS["httmp"] = 5 right side

printf "***** \n"
printf "Postprocess of RELAP5 file %s.i\n", FILE
printf "* The following nodes will be postprocessed: \n"
for (i=1;i<=TOTVOLS;i++) {
    printf "%s\n", VOLUME[i]
}
printf "%6d %s %s\n", NSTEPS, MAXDT, INITTIME
printf "1\n"
OUTCOUNT = 1
printf "***** \n"
printf "TIME\n"
printf "%s\n", OUTCOUNT
printf "0\n"
printf "27\n"

```



```

printf "*-----\n"
printf "[T]\n"
TIME = OUTCOUNT++
printf "***** \n"
printf "STEP AT STARTTIME\n"
printf "%s\n", OUTCOUNT
printf "0\n"
printf "27\n"
printf "*-----\n"
printf "STEP([T]~%s)\n", STARTTIME
STEPTIME = OUTCOUNT++
printf "***** \n"
printf "PULSE AT STARTTIME\n"
printf "%s\n", OUTCOUNT
printf "0\n"
printf "15\n"
printf "*-----\n"
printf "5\n"
printf "%7.5f %7.5f %7.5f %7.5f %7.5f\n", INITIME-MAXDT, STARTTIME-(MAXDT/2.0), \
    STARTTIME, STARTTIME+(MAXDT/2.0), \
    ENDTIME

printf "0.0 0.0 1.0 0.0 0.0\n"
PULSETIME = OUTCOUNT++
printf "***** \n"
printf "ACTIVATION SIGNAL FOR MODULE SNDCODE\n"
printf "%s\n", OUTCOUNT
printf "0\n"
printf "15\n"
printf "*-----\n"
printf "4\n"
printf "%3.1f %9.7f %s %3.1f\n", INITIME-100.0, INITIME-0.0000001, \
    INITIME, ENDTIME

printf "0.0 0.0 1.0 1.0\n"
act_signal = OUTCOUNT++
printf "***** \n"
printf "CONSTANT 1\n"
printf "%s\n", OUTCOUNT
printf "0\n"
printf "15\n"
printf "*-----\n"
printf "2\n"
printf "-100.0 100.0\n"
printf "1.0 1.0\n"
CONST1 = OUTCOUNT++
printf "***** \n"
printf "CONSTANT 0\n"
printf "%s\n", OUTCOUNT
printf "0\n"
printf "15\n"
printf "*-----\n"
printf "2\n"
printf "-100.0 100.0\n"
printf "0.0 0.0\n"
CONST0 = OUTCOUNT++
printf "***** \n"
printf "EMISSION MODULE\n"
printf "%s\n", OUTCOUNT
printf "%s\n", act_signal
printf "60\n"
printf "*-----\n"
printf "1\n"
printf "%s\n", HOST
printf "relap5.x -i %s.i -o %s.o -r %s.r\n", \
    FILE, FILE, FILE
printf "1\n"

```

```

printf "0\n"
printf "0\n"
rcodetid = OUTCOUNT++
printf "***** \n"
printf "RECEPTION MODULE\n"
printf "%s\n", OUTCOUNT
printf "%s %s\n", act_signal, rcodetid
printf "61\n"
printf "*-----\n"
printf "%s\n", trtouts
for (i=1;i<=TOTVOLS;i++) {
    printf " vvol-%s\n", VOLUME[i]
    printf " avol-%s\n", VOLUME[i]
    printf " p-%s\n", VOLUME[i]
    printf " uf-%s\n", VOLUME[i]
    printf " ug-%s\n", VOLUME[i]
    printf " voidf-%s\n", VOLUME[i]
    printf " voidg-%s\n", VOLUME[i]
    printf " rhof-%s\n", VOLUME[i]
    printf " rhog-%s\n", VOLUME[i]
    printf " satt-%s\n", VOLUME[i]
    printf " tempf-%s\n", VOLUME[i]
    printf " tempg-%s\n", VOLUME[i]
    printf " velf-%s\n", VOLUME[i]
    printf " velg-%s\n", VOLUME[i]
    printf " betaff-%s\n", VOLUME[i]
    printf " betagg-%s\n", VOLUME[i]
    printf " csubpf-%s\n", VOLUME[i]
    printf " csubpg-%s\n", VOLUME[i]
    printf " hif-%s\n", VOLUME[i]
    printf " hig-%s\n", VOLUME[i]
    printf " gammaw-%s\n", VOLUME[i]
    printf " gammai-%s\n", VOLUME[i]
    printf " qwf-%s\n", VOLUME[i]
    printf " qwg-%s\n", VOLUME[i]
    printf " drfdp-%s\n", VOLUME[i]
    printf " drgdp-%s\n", VOLUME[i]
    printf " fwalfx-%s\n", VOLUME[i]
    printf " fwalgx-%s\n", VOLUME[i]
    vvol[VOLUME[i]] = OUTCOUNT++
    avol[VOLUME[i]] = OUTCOUNT++
    p[VOLUME[i]] = OUTCOUNT++
    uf[VOLUME[i]] = OUTCOUNT++
    ug[VOLUME[i]] = OUTCOUNT++
    voidf[VOLUME[i]] = OUTCOUNT++
    voidg[VOLUME[i]] = OUTCOUNT++
    rhof[VOLUME[i]] = OUTCOUNT++
    rhog[VOLUME[i]] = OUTCOUNT++
    satt[VOLUME[i]] = OUTCOUNT++
    tempf[VOLUME[i]] = OUTCOUNT++
    tempg[VOLUME[i]] = OUTCOUNT++
    velf[VOLUME[i]] = OUTCOUNT++
    velg[VOLUME[i]] = OUTCOUNT++
    betaff[VOLUME[i]] = OUTCOUNT++
    betagg[VOLUME[i]] = OUTCOUNT++
    csubpf[VOLUME[i]] = OUTCOUNT++
    csubpg[VOLUME[i]] = OUTCOUNT++
    hif[VOLUME[i]] = OUTCOUNT++
    hig[VOLUME[i]] = OUTCOUNT++
    gammaw[VOLUME[i]] = OUTCOUNT++
    gammai[VOLUME[i]] = OUTCOUNT++
    qwf[VOLUME[i]] = OUTCOUNT++
    qwg[VOLUME[i]] = OUTCOUNT++
    drfdp[VOLUME[i]] = OUTCOUNT++
    drgdp[VOLUME[i]] = OUTCOUNT++
}

```

```

        fwalfx[VOLUME[i]] = OUTCOUNT++
        fwalgx[VOLUME[i]] = OUTCOUNT++
    }
    for(i=1; i<=njuns;i++) {
        printf " velfj-%s\n", JUNCTION[i]
        printf " velgj-%s\n", JUNCTION[i]
        printf " voidfj-%s\n", JUNCTION[i]
        printf " voidgj-%s\n", JUNCTION[i]
        printf " rhofj-%s\n", JUNCTION[i]
        printf " rhogj-%s\n", JUNCTION[i]
        printf " ajun-%s\n", JUNCTION[i]
        printf " formfj-%s\n", JUNCTION[i]
        printf " formgj-%s\n", JUNCTION[i]
        velfj[JUNCTION[i]] = OUTCOUNT++
        velgj[JUNCTION[i]] = OUTCOUNT++
        voidfj[JUNCTION[i]] = OUTCOUNT++
        voidgj[JUNCTION[i]] = OUTCOUNT++
        rhofj[JUNCTION[i]] = OUTCOUNT++
        rhogj[JUNCTION[i]] = OUTCOUNT++
        ajun[JUNCTION[i]] = OUTCOUNT++
        formfj[JUNCTION[i]] = OUTCOUNT++
        formgj[JUNCTION[i]] = OUTCOUNT++
    }
    for(i=1;i<=newvcount;i++) {
        printf " p-%s\n", NEWVOL[i]
        p[NEWVOL[i]] = OUTCOUNT++
    }
    for(nht=1; nht<=NHEATSTRS; nht++){
        printf " htrnrn-%9d\n", heatst[nht]*100
        printf " htpown-%9d\n", heatst[nht]*100
        printf " htsrfn-%9d\n", heatst[nht]*100
        printf " httpm-%9d\n", (heatst[nht]*100) + 1
        printf " http-%9d\n", (heatst[nht]*100) + nmpoints[nht]
        htrnrn[nht] = OUTCOUNT++
        htpown[nht] = OUTCOUNT++
        htsrfn[nht] = OUTCOUNT++
        httpmleft[nht] = OUTCOUNT++
        httptright[nht] = OUTCOUNT++
    }

    printf "*****\n"
    printf "***** Computation of junction related variables *****\n"
    printf "*****\n"
    for(j=1;j<=njuns;j++) {
        printf "Area * vapor fraction in junction %s\n", JUNCTION[j]
        printf "%s\n", OUTCOUNT
        printf "%s %s\n", ajun[JUNCTION[j]], voidgj[JUNCTION[j]]
        printf "27\n"
        printf "*-----\n"
        printf "[1]*[2]\n"
        avoidg[JUNCTION[j]] = OUTCOUNT++
        printf "***** \n"
        printf "Area * liquid fraction in junction %s\n", JUNCTION[j]
        printf "%s\n", OUTCOUNT
        printf "%s %s\n", ajun[JUNCTION[j]], voidfj[JUNCTION[j]]
        printf "27\n"
        printf "*-----\n"
        printf "[1]*[2]\n"
        avoidf[JUNCTION[j]] = OUTCOUNT++
        printf "***** \n"
        printf "Area * vapor fraction * density in junction %s\n", JUNCTION[j]
        printf "%s\n", OUTCOUNT
        printf "%s %s %s\n", ajun[JUNCTION[j]], voidgj[JUNCTION[j]], rhogj[JUNCTION[j]]
        printf "27\n"
        printf "*-----\n"
    }

```

```

printf "[1]*[2]*[3]\n"
avoidrhog[JUNCTION[j]] = OUTCOUNT++
printf "***** \n"
printf "Area * liquid fraction * density in junction %s\n", JUNCTION[j]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", ajun[JUNCTION[j]], voidfj[JUNCTION[j]], rhofj[JUNCTION[j]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]\n"
avoidrhog[JUNCTION[j]] = OUTCOUNT++
printf "***** \n"
printf "Area * mixture density in junction %s\n", JUNCTION[j]
printf "%s\n", OUTCOUNT
printf "%s %s\n", avoidrhog[JUNCTION[j]], avoidrhof[JUNCTION[j]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
arhomix[JUNCTION[j]] = OUTCOUNT++
printf "***** \n"
printf "Area * vapor fraction * density * velocity in junction %s\n", JUNCTION[j]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", ajun[JUNCTION[j]], voidgj[JUNCTION[j]], \
    rhogj[JUNCTION[j]], velgj[JUNCTION[j]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]*[4]\n"
avoidrhovelg[JUNCTION[j]] = OUTCOUNT++
printf "***** \n"
printf "Area * liquid fraction * density * velocity in junction %s\n", JUNCTION[j]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", ajun[JUNCTION[j]], voidfj[JUNCTION[j]], \
    rhofj[JUNCTION[j]], velfj[JUNCTION[j]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]*[4]\n"
avoidrhovelf[JUNCTION[j]] = OUTCOUNT++
printf "***** \n"
# printf "Mixture velocity in junction %s\n", JUNCTION[j]
# printf "%s\n", OUTCOUNT
# printf "%s %s %s\n", avoidrhovelg[JUNCTION[j]], avoidrhovelf[JUNCTION[j]],
#     arhovmix[JUNCTION[j]]
#
# printf "27\n"
# printf "*-----\n"
# printf "([1]*[2])/[3]\n"
# velmixj[JUNCTION[j]] = OUTCOUNT++
# printf "***** \n"
# printf "Area* mixture density * mixture velocity**2 in junction %s\n",
#     JUNCTION[j]
#
# printf "%s\n", OUTCOUNT
# printf "%s %s %s\n", arhomix[JUNCTION[j]], velmixj[JUNCTION[j]],
#     velmixj[JUNCTION[j]]
#
# printf "27\n"
# printf "*-----\n"
# printf "[1]*[2]*[3]\n"
# arhovel2mix[JUNCTION[j]] = OUTCOUNT++
# printf "***** \n"
# printf "Slip in junction %s\n", JUNCTION[j]
# printf "%s\n", OUTCOUNT
# printf "%s %s\n", velgj[JUNCTION[j]], velfj[JUNCTION[j]]
# printf "27\n"
# printf "*-----\n"
# printf "[1]-[2]\n"
# slip[JUNCTION[j]] = OUTCOUNT++
# printf "***** \n"
# printf "Non homogeneous momentum term in junction %s\n", JUNCTION[j]

```

```

#   printf "%s\n", OUTCOUNT
#   printf "%s %s %s %s %s\n", avoidrhog[JUNCTION[j]], avoidrhof[JUNCTION[j]],
#   slip[JUNCTION[j]], slip[JUNCTION[j]],
#   arhomix[JUNCTION[j]]
#   printf "27\n"
#   printf "*-----\n"
#   printf "([1]*[2]*[3]*[4])/[5]\n"
#   nhommom[JUNCTION[j]] = OUTCOUNT++
#   printf "***** \n"
printf "Pressure in junction (vapor) %s\n", JUNCTION[j]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", p[TOVOL[j]], p[FROMVOL[j]], velgj[JUNCTION[j]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "0.0\n"
pjpg[JUNCTION[j]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Pressure in junction (liquid) %s\n", JUNCTION[j]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", p[TOVOL[j]], p[FROMVOL[j]], velfj[JUNCTION[j]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "0.0\n"
pjf[JUNCTION[j]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Area * vapor fraction * pjpg in junction %s\n", JUNCTION[j]
printf "%s\n", OUTCOUNT
printf "%s %s\n", avoidg[JUNCTION[j]], pjpg[JUNCTION[j]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
avoidpjpg[JUNCTION[j]] = OUTCOUNT++
printf "***** \n"
printf "Area * liquid fraction * pjf in junction %s\n", JUNCTION[j]
printf "%s\n", OUTCOUNT
printf "%s %s\n", avoiddf[JUNCTION[j]], pjf[JUNCTION[j]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
avoidpjf[JUNCTION[j]] = OUTCOUNT++
printf "***** \n"
printf "Local pressure drop (vapor phase) in junction %s\n", JUNCTION[j]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", voidgj[JUNCTION[j]], rhogj[JUNCTION[j]],
velgj[JUNCTION[j]], formgj[JUNCTION[j]]

printf "27\n"
printf "*-----\n"
printf "0.5*[1]*[2]*abs([3])*[3]*[4]\n"
locdpjpg[JUNCTION[j]] = OUTCOUNT++
printf "***** \n"
printf "Local pressure drop (liquid phase) in junction %s\n", JUNCTION[j]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", voidfj[JUNCTION[j]], rhofj[JUNCTION[j]],
velfj[JUNCTION[j]], formfj[JUNCTION[j]]

printf "27\n"
printf "*-----\n"
printf "0.5*[1]*[2]*abs([3])*[3]*[4]\n"
locdpjf[JUNCTION[j]] = OUTCOUNT++
printf "***** \n"
printf "Local pressure drop in junction %s\n", JUNCTION[j]
printf "%s\n", OUTCOUNT

```

```

    printf "%s %s\n", locdpjg[JUNCTION[j]], locdpjf[JUNCTION[j]]
    printf "27\n"
    printf "*-----\n"
    printf "[1]+[2]\n"
    locdpj[JUNCTION[j]] = OUTCOUNT++
    printf "***** \n"
}
# End of junction loop
# printf "OUTPUT FILE LOCAL PRESSURE DROPS\n"
# printf "%s\n", OUTCOUNT
# for(j=1;j<njuns;j++) {
#     printf "%s \\n", locdpj[JUNCTION[j]]
# }
# printf "%s\n", locdpj[JUNCTION[njuns]]
# printf "38\n"
# printf "*-----\n"
# printf "%s_locdpj.dat\n", FILE
# printf "(%s(E17.8))\n", njuns+1
# printf "(%s(A16,x))\n", njuns+1
# printf "%s 0 %6d\n", FREQ, ((STARTTIME-INITTIME)/MAXDT)+2
# for(j=1;j<=njuns;j++) {
#     printf "locdpj-%s\n", JUNCTION[j]
# }
# locdpjoutfile = OUTCOUNT++
printf "***** \n"

for(i=1;i<=TOTVOLSI;i++) {
    printf "***** \n"
    printf "***** Postprocess of control volume %s *****\n", VOLUME[i]
    printf "***** \n"
    printf "Mixture density %s\n", VOLUME[i]
    printf "%s\n", OUTCOUNT
    printf "%s %s %s %s\n", voidg[VOLUME[i]], rhog[VOLUME[i]], \
        voidf[VOLUME[i]], rhof[VOLUME[i]]

    printf "27\n"
    printf "*-----\n"
    printf "[1]*[2]+[3]*[4]\n"
    provrhomix[VOLUME[i]] = OUTCOUNT++
    printf "***** \n"
    printf "Non-zero mixture density %s\n", VOLUME[i]
    printf "%s\n", OUTCOUNT
    printf "%s %s %s\n", CONST1, provrhomix[VOLUME[i]], TIME
    printf "14\n"
    printf "*-----\n"
    printf "3\n"
    printf "%s\n", INITTIME-MAXDT/2.0
    rhomix[VOLUME[i]] = OUTCOUNT++
    OUTCOUNT++
    printf "***** \n"
    printf "Numerator of mixture velocity %s\n", VOLUME[i]
    printf "%s\n", OUTCOUNT
    printf "%s %s %s %s %s %s\n", voidg[VOLUME[i]], rhog[VOLUME[i]], \
        velg[VOLUME[i]], voidf[VOLUME[i]], \
        rhof[VOLUME[i]], velf[VOLUME[i]]

    printf "27\n"
    printf "*-----\n"
    printf "([1]*[2]*[3])+([4]*[5]*[6])\n"
    numvelmix[VOLUME[i]] = OUTCOUNT++
    printf "***** \n"
    printf "Mixture velocity %s\n", VOLUME[i]
    printf "%s\n", OUTCOUNT
    printf "%s %s\n", numvelmix[VOLUME[i]], rhomix[VOLUME[i]]
    printf "27\n"

```

```

printf "*-----\n"
printf "[1]/[2]\n"
velmix[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Mixture momentum %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", vvola[VOLUME[i]], numvelmix[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
mommix[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Time derivative of mixture momentum %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s\n", mommix[VOLUME[i]]
printf "43\n"
printf "*-----\n"
printf "16 1.0\n"
dermommix[VOLUME[i]] = OUTCOUNT++
printf "***** \n"

printf "Auxiliary variable for csubvg %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s\n", drgdp[VOLUME[i]], p[VOLUME[i]], \
    betagg[VOLUME[i]], rhog[VOLUME[i]], \
    csubpg[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*([2]*[3])-(4)*[5])\n"
auxcvg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Denominator of csubvg %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", auxcvg[VOLUME[i]], rhog[VOLUME[i]], \
    betagg[VOLUME[i]], tempg[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]-([2]*[3]*[3]*[4]))\n"
provdencvg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non-zero denominator of csubvg %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, provdencvg[VOLUME[i]], TIME
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "%s\n", INITTIME-MAXDT/2.0
dencvg[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "csubvg %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", auxcvg[VOLUME[i]], csubpg[VOLUME[i]], \
    dencvg[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]/[3]\n"
csubvg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Auxiliary variable for csubvf %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s\n", drfddp[VOLUME[i]], p[VOLUME[i]], \
    betaff[VOLUME[i]], rhof[VOLUME[i]], \

```

```

                                csubpf[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*([2]*[3])-([4]*[5])\n"
auxcvf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Denominator of csubvf %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", auxcvf[VOLUME[i]], rhof[VOLUME[i]], \
                                betaff[VOLUME[i]], tempf[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]-([2]*[3]*[3]*[4]))\n"
provdencvf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non-zero denominator of csubvf %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, provdencvf[VOLUME[i]], TIME
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "%s\n", INITTIME-MAXDT/2.0
dencvf[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "csubvf %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", auxcvf[VOLUME[i]], csubpf[VOLUME[i]], \
                                dencvf[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]/[3]\n"
csubvf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Saturated vapor variables %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", p[VOLUME[i]], CONST1
printf "62\n"
printf "*-----\n"
printf "P x\n"
printf "cpg cvg betagg hg rhog\n"
satcpg[VOLUME[i]] = OUTCOUNT++
satcvg[VOLUME[i]] = OUTCOUNT++
satbetagg[VOLUME[i]] = OUTCOUNT++
sathg[VOLUME[i]] = OUTCOUNT++
sathrog[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Saturated liquid variables %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", p[VOLUME[i]], CONST0
printf "62\n"
printf "*-----\n"
printf "P x\n"
printf "cpf cvf betaff hf rhof\n"
satcpf[VOLUME[i]] = OUTCOUNT++
satcvf[VOLUME[i]] = OUTCOUNT++
satbetaff[VOLUME[i]] = OUTCOUNT++
sathf[VOLUME[i]] = OUTCOUNT++
sathrof[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non-zero vapor density %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, rhog[VOLUME[i]], TIME
printf "14\n"
printf "*-----\n"

```



```

printf "3\n"
printf "%s\n", INITTIME-MAXDT/2.0
nzzrhog[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Vapor enthalpy %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", ug[VOLUME[i]], p[VOLUME[i]], nzzrhog[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+([2]/[3])\n"
hg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non-zero liquid density %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, rho[VOLUME[i]], TIME
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "%s\n", INITTIME-MAXDT/2.0
nzzrhof[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Liquid enthalpy %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", uf[VOLUME[i]], p[VOLUME[i]], nzzrhof[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+([2]/[3])\n"
hf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Saturation enthalpy jump %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", sathg[VOLUME[i]], sathf[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
hfg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Saturation volume jump %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", satrhog[VOLUME[i]], satrhof[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "(1.0/[1])-(1.0/[2])\n"
vfg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor generation rate %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", gammaw[VOLUME[i]], gammai[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
gammag[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid generation rate %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s\n", gammag[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "-1.0*[1]\n"
gammaf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid generation rate in volume %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT

```

```

printf "%s %s\n", vvol[VOLUME[i]], gammaf[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
genmf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Computation of hstarg %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", hg[VOLUME[i]], sathg[VOLUME[i]], gammai[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "0.0\n"
hstarg[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Computation of hstarf %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", sathf[VOLUME[i]], hf[VOLUME[i]], gammai[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "0.0\n"
hstarf[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Computation of hprimeg %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", hg[VOLUME[i]], sathg[VOLUME[i]], gammaw[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "0.0\n"
hprimeg[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Computation of hprimef %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", sathf[VOLUME[i]], hf[VOLUME[i]], gammaw[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "0.0\n"
hprimef[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Wall interfacial heat power (vapor) %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", hprimeg[VOLUME[i]]+1, hprimef[VOLUME[i]], \
    hprimeg[VOLUME[i]], gammaw[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "(1.0 - [1])*([2]-[3])*[4]\n"
qiwg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Wall interfacial heat power (liquid) %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", hprimeg[VOLUME[i]]+1, hprimef[VOLUME[i]], \
    hprimeg[VOLUME[i]], gammaw[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*([2]-[3])*[4]\n"
qiwf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Bulk interfacial heat power (vapor) %s\n", VOLUME[i]

```

```

printf "%s\n", OUTCOUNT
printf "%s %s %s\n", hig[VOLUME[i]], satt[VOLUME[i]], tempg[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*([2]-[3])\n"
qibg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Bulk interfacial heat power (liquid) %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", hif[VOLUME[i]], satt[VOLUME[i]], tempf[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*([2]-[3])\n"
qibf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Total interfacial heat power (vapor) %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", qiwg[VOLUME[i]], qibg[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
qig[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Total interfacial heat power (liquid) %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", qiwf[VOLUME[i]], qibf[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
qif[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Enthalpy transferred to vapor through the interphase %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", gammai[VOLUME[i]], hstarg[VOLUME[i]], \
    gammaw[VOLUME[i]], hprimeg[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]+[3]*[4]\n"
gammahig[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Enthalpy transferred to liquid through the interphase %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", gammai[VOLUME[i]], hstarf[VOLUME[i]], \
    gammaw[VOLUME[i]], hprimef[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "-1.0*([1]*[2]+[3]*[4])\n"
gammahif[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Gamma * h (vapor) %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", gammag[VOLUME[i]], hg[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
gammaghg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Gamma * h (liquid) %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", gammaf[VOLUME[i]], hf[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
gammahf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"

```

```

printf "Non-zero csubpg %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, csubpg[VOLUME[i]], TIME
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "%s\n", INITTIME-MAXDT/2.0
nzcsubpg[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Vapor isobaric expansion coefficient %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", betagp[VOLUME[i]], nzhog[VOLUME[i]], nzcsubpg[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]/([2]*[3])\n"
pig[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non-zero csubpf %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, csubpf[VOLUME[i]], TIME
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "%s\n", INITTIME-MAXDT/2.0
nzcsubpf[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Liquid isobaric expansion coefficient %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", betaff[VOLUME[i]], nzrhof[VOLUME[i]], nzcsubpf[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]/([2]*[3])\n"
pif[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor swelling %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", vvof[VOLUME[i]], gammag[VOLUME[i]], satrhog[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "([1]*[2])/[3]\n"
swellg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid swelling %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", vvof[VOLUME[i]], gammag[VOLUME[i]], satrhof[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "([1]*[2])/[3]\n"
swellf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non-zero vvof %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, vvof[VOLUME[i]], TIME
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "%s\n", INITTIME-MAXDT/2.0
nzvvof[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Auxiliary term 1 for thermal expansion plus swelling (vapor) %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s\n", qwg[VOLUME[i]], nzvvof[VOLUME[i]], qig[VOLUME[i]], \

```

```

                                gammahig[VOLUME[i]], gammag[VOLUME[i]], hg[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]/[2])+[3]+[4]-([5]*[6])\n"
thswaux1g[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Thermal expansion plus swelling (vapor) %s\n", \
                                VOLUME[i]

printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s\n", pig[VOLUME[i]], thswaux1g[VOLUME[i]], \
                                gammag[VOLUME[i]], nzrhog[VOLUME[i]], vvvol[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "(([1]*[2])+([3]/[4]))*[5]\n"
thswg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Nq term with q wall (vapor) %s\n", \
                                VOLUME[i]

printf "%s\n", OUTCOUNT
printf "%s %s %s\n", qwg[VOLUME[i]], pig[VOLUME[i]], satrhog[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]\n"
qtermg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Nq term with interfacial heat (vapor) %s\n", \
                                VOLUME[i]

printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", vvvol[VOLUME[i]], qig[VOLUME[i]], pig[VOLUME[i]], \
                                satrhog[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]*[4]\n"
qitermg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Nq term with gamma (vapor) %s\n", \
                                VOLUME[i]

printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s\n", gammahig[VOLUME[i]], gammag[VOLUME[i]], \
                                nzrhog[VOLUME[i]], satrhog[VOLUME[i]], \
                                pig[VOLUME[i]], vvvol[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "(((5)*[1])-(2)*((1.0/[3])-(1.0/[4]))))*[6]*[4]\n"
gammtg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Auxiliary term 1 for thermal expansion plus swelling (liquid) %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s\n", qwf[VOLUME[i]], nzvvvol[VOLUME[i]], qif[VOLUME[i]], \
                                gammahif[VOLUME[i]], gammaf[VOLUME[i]], hf[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]/[2])+[3]+[4]-([5]*[6])\n"
thswaux1f[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Thermal expansion plus swelling (liquid) %s\n", \
                                VOLUME[i]

printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s\n", pif[VOLUME[i]], thswaux1f[VOLUME[i]], \
                                gammaf[VOLUME[i]], nzrhof[VOLUME[i]], vvvol[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "(([1]*[2])+([3]/[4]))*[5]\n"
thswf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"

```

```

printf "Nq term with q wall (liquid) %s\n", \
      VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", qwf[VOLUME[i]], pif[VOLUME[i]], satrhof[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]\n"
qwttermf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Nq term with interfacial heat (liquid) %s\n", \
      VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", vvof[VOLUME[i]], qif[VOLUME[i]], pif[VOLUME[i]], \
      satrhof[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]*[4]\n"
qitermf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Nq term with gamma (liquid) %s\n", \
      VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s\n", gammahif[VOLUME[i]], gammaf[VOLUME[i]], \
      nzrhof[VOLUME[i]], satrhof[VOLUME[i]], \
      pif[VOLUME[i]], vvof[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "((([5]*[1])-( [2]*((1.0/[3])-(1.0/[4]))))*[6]*[4])\n"
gammtf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"

printf "Vapor thermal expansion %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", thswg[VOLUME[i]], swellg[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
thexpg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid thermal expansion %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", thswf[VOLUME[i]], swellf[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
thexpf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor isentropic compressibility %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s\n", tempg[VOLUME[i]], p[VOLUME[i]], betagg[VOLUME[i]], \
      csbvg[VOLUME[i]], nzrhog[VOLUME[i]], nzcsbpg[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "([1]*[2]*[3]*[4])/([5]*[6]*([6]-[4]))\n"
compf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor fraction * isentropic compressibility %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", voidg[VOLUME[i]], compg[VOLUME[i]]
printf "27\n"
printf "*-----\n"

```

```

printf "[1]*[2]\n"
voidcomp[ VOLUME[i] ] = OUTCOUNT++
printf "***** \n"
printf "Time derivative of vapor fraction * isentropic compressibility %s\n", \
    VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s\n", voidcomp[ VOLUME[i] ]
printf "43\n"
printf "*-----\n"
printf "16 1.0\n"
dervoidcomp[ VOLUME[i] ] = OUTCOUNT++
printf "***** \n"
printf "Non zero vapor isentropic compressibility %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, comp[ VOLUME[i] ], comp[ VOLUME[i] ]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzcomp[ VOLUME[i] ] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Liquid isentropic compressibility %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s\n", temp[ VOLUME[i] ], p[ VOLUME[i] ], betaff[ VOLUME[i] ], \
    csubvf[ VOLUME[i] ], nzrhof[ VOLUME[i] ], nzsubpf[ VOLUME[i] ]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]*[3]*[4])/([5]*[6]*([6]-[4]))\n"
comp[ VOLUME[i] ] = OUTCOUNT++
printf "***** \n"
printf "Liquid fraction * isentropic compressibility %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", voidf[ VOLUME[i] ], comp[ VOLUME[i] ]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
voidcompf[ VOLUME[i] ] = OUTCOUNT++
printf "***** \n"
printf "Time derivative of liquid fraction * isentropic compressibility %s\n", \
    VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s\n", voidcompf[ VOLUME[i] ]
printf "43\n"
printf "*-----\n"
printf "16 1.0\n"
dervoidcompf[ VOLUME[i] ] = OUTCOUNT++
printf "***** \n"
printf "Non zero liquid isentropic compressibility %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, compf[ VOLUME[i] ], compf[ VOLUME[i] ]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzcompf[ VOLUME[i] ] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Vapor isentropic compressibility integrated in the whole node %s\n", \
    VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", vvof[ VOLUME[i] ], voidg[ VOLUME[i] ], compg[ VOLUME[i] ]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]\n"

```

```

nodecomp[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid isentropic compressibility integrated in the whole node %s\n",
      VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", vv[VOLUME[i]], voidf[VOLUME[i]], compf[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]\n"
nodecomp[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Total node compressibility %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", nodecomp[VOLUME[i]], nodecomp[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
totnodecomp[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non zero total node compressibility %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, totnodecomp[VOLUME[i]], totnodecomp[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nztotncomp[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Vapor compressibility along the saturation line %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s %s %s\n", p[VOLUME[i]], satt[VOLUME[i]], satbetagg[VOLUME[i]], \
      vfg[VOLUME[i]], hfg[VOLUME[i]], satrhog[VOLUME[i]], \
      satcvg[VOLUME[i]], satcpg[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "-1.0*[1]*[2]*[3]*(((4)/[5])+(3)/([6]*([7]-[8])))\n"
satcomp[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid compressibility along the saturation line %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s %s %s\n", p[VOLUME[i]], satt[VOLUME[i]], satbetaff[VOLUME[i]], \
      vfg[VOLUME[i]], hfg[VOLUME[i]], satrhof[VOLUME[i]], \
      satcvf[VOLUME[i]], satcpf[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "-1.0*[1]*[2]*[3]*(((4)/[5])+(3)/([6]*([7]-[8])))\n"
satcomp[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor volume %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", voidg[VOLUME[i]], vv[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
volg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid volume %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", voidf[VOLUME[i]], vv[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
volg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"

```



```

printf "Time derivative of liquid volume %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s\n", volf[VOLUME[i]]
printf "43\n"
printf "*-----\n"
printf "16 1.0\n"
dvfdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", volg[VOLUME[i]], rhog[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
massg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", volf[VOLUME[i]], rhof[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
massf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor wall friction %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
# printf "%s %s %s\n", massg[VOLUME[i]], velg[VOLUME[i]], fwalgx[VOLUME[i]]
printf "%s %s %s\n", vvol[VOLUME[i]], velg[VOLUME[i]], fwalgx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]\n"
wfricg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid wall friction %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
# printf "%s %s %s\n", massf[VOLUME[i]], velf[VOLUME[i]], fwalfx[VOLUME[i]]
printf "%s %s %s\n", vvol[VOLUME[i]], velf[VOLUME[i]], fwalfx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]\n"
wfricf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor buoyancy term %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", avol[VOLUME[i]], voidg[VOLUME[i]], rhog[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "%s*%s*[1]*[2]*[3]\n", GRAV, HEIGHT[VOLUME[i]]
buoyg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid buoyancy term %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", avol[VOLUME[i]], voidf[VOLUME[i]], rhof[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "%s*%s*[1]*[2]*[3]\n", GRAV, HEIGHT[VOLUME[i]]
buoyf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor non-equilibrium mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", volg[VOLUME[i]], satrhog[VOLUME[i]], rhog[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*([2]-[3])\n"
nqmg[VOLUME[i]] = OUTCOUNT++

```

```

printf "***** \n"
printf "Liquid non-equilibrium mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", volf[VOLUME[i]], satrhof[VOLUME[i]], rhof[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*([2]-[3])\n"
nqmf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
for(j=1;j<=njuns;j++) {
    inletx[j] = ""
    outletx[j] = ""
    inley[j] = ""
    outley[j] = ""
    inletz[j] = ""
    outletz[j] = ""
    if(TOVOL[j] == VOLUME[i]) {
        if (INOUTFLAGTO[j] == 1) {
            inletx[j] = "+"
        }
        else if(INOUTFLAGTO[j] == 2) {
            outletx[j] = "-"
        }
        else if(INOUTFLAGTO[j] == 3) {
            inley[j] = "+"
        }
        else if(INOUTFLAGTO[j] == 4) {
            outley[j] = "-"
        }
        else if(INOUTFLAGTO[j] == 5) {
            inletz[j] = "+"
        }
        else if(INOUTFLAGTO[j] == 6) {
            outletz[j] = "-"
        }
    }
    else if(FROMVOL[j] == VOLUME[i]) {
        if (INOUTFLAGFROM[j] == 1) {
            inletx[j] = "-"
        }
        else if(INOUTFLAGFROM[j] == 2) {
            outletx[j] = "+"
        }
        else if(INOUTFLAGFROM[j] == 3) {
            inley[j] = "-"
        }
        else if(INOUTFLAGFROM[j] == 4) {
            outley[j] = "+"
        }
        else if(INOUTFLAGFROM[j] == 5) {
            inletz[j] = "-"
        }
        else if(INOUTFLAGFROM[j] == 6) {
            outletz[j] = "+"
        }
    }
}
}
# End of junction loop

printf "Inlet area x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletx[j] != "") {
        printf "%s ", ajun[JUNCTION[j]]
    }
}

```

```

    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletx[j] != ""){
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inareax[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidg x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletx[j] != "") {
        printf "%s ", avoidg[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletx[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inavoidgx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidf x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletx[j] != "") {
        printf "%s ", avoidf[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletx[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inavoidfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidg * rhog x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletx[j] != "") {
        printf "%s ", avoidrhog[JUNCTION[j]]
    }
}
}

```

```

printf "\n"
printf "27\n"
printf "-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletx[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inavoidrhogx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidf * rhof x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletx[j] != "") {
        printf "%s ", avoidrhof[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletx[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inavoidrhofx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidg * rhog * velg x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletx[j] != "") {
        printf "%s ", avoidrhovelg[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletx[j] != "") {
        printf "%s[%s]", inletx[j], auxcount++
    }
}
printf "\n"
inavoidrhovelgx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidf * rhog * velf x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletx[j] != "") {
        printf "%s ", avoidrhovelf[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"

```

```

printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletx[j] != "") {
        printf "%s[%s]", inletx[j], auxcount++
    }
}
printf "\n"
inavoidrhovelfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidg * pressure x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletx[j] != "") {
        printf "%s ", avoidpjpg[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletx[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inavoidpjg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidf * pressure x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletx[j] != "") {
        printf "%s ", avoidpjf[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletx[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inavoidpjfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outletx[j] != "") {
        printf "%s ", ajun[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"

```

```

auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outletx[j] != ""){
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outareax[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidg x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outletx[j] != "") {
        printf "%s ", avoidg[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outletx[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outavoidgx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidf x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outletx[j] != "") {
        printf "%s ", avoidf[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outletx[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outavoidfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidg * rhog x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outletx[j] != "") {
        printf "%s ", avoidrhog[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {

```

```

        if(outletx[j] != "") {
            printf "+[%s]", auxcount++
        }
    }
    printf "\n"
    outavoidrhogx[VOLUME[i]] = OUTCOUNT++
    printf "***** \n"
    printf "Outlet area * voidf * rhof x %s\n", VOLUME[i]
    printf "%s\n", OUTCOUNT
    printf "%s ", CONSTO
    for(j=1;j<=njuns;j++) {
        if(outletx[j] != "") {
            printf "%s ", avoidrhof[JUNCTION[j]]
        }
    }
    printf "\n"
    printf "27\n"
    printf "*-----\n"
    printf "[1]"
    auxcount = 2
    for(j=1;j<=njuns;j++) {
        if(outletx[j] != "") {
            printf "+[%s]", auxcount++
        }
    }
    printf "\n"
    outavoidrhofox[VOLUME[i]] = OUTCOUNT++
    printf "***** \n"
    printf "Outlet area * voidg * rhog * velg x %s\n", VOLUME[i]
    printf "%s\n", OUTCOUNT
    printf "%s ", CONSTO
    for(j=1;j<=njuns;j++) {
        if(outletx[j] != "") {
            printf "%s ", avoidrhovelg[JUNCTION[j]]
        }
    }
    printf "\n"
    printf "27\n"
    printf "*-----\n"
    printf "[1]"
    auxcount = 2
    for(j=1;j<=njuns;j++) {
        if(outletx[j] != "") {
            printf "%s[%s]", outletx[j], auxcount++
        }
    }
    printf "\n"
    outavoidrhovelgx[VOLUME[i]] = OUTCOUNT++
    printf "***** \n"
    printf "Outlet area * voidf * rhog * velf x %s\n", VOLUME[i]
    printf "%s\n", OUTCOUNT
    printf "%s ", CONSTO
    for(j=1;j<=njuns;j++) {
        if(outletx[j] != "") {
            printf "%s ", avoidrhovelf[JUNCTION[j]]
        }
    }
    printf "\n"
    printf "27\n"
    printf "*-----\n"
    printf "[1]"
    auxcount = 2
    for(j=1;j<=njuns;j++) {
        if(outletx[j] != "") {
            printf "%s[%s]", outletx[j], auxcount++
        }
    }

```

```

    }
}
printf "\n"
outavoidrhovelfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidg * pressure x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outletx[j] != "") {
        printf "%s ", avoidpjpg[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outletx[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outavoidpjpgx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidf * pressure x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outletx[j] != "") {
        printf "%s ", avoidpjf[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outletx[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outavoidpjfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non zero inlet area x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, inareax[VOLUME[i]], inareax[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzinareax[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Non zero outlet area x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, outareax[VOLUME[i]], outareax[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzoutareax[VOLUME[i]] = OUTCOUNT++

```



```

OUTCOUNT++
printf "***** \n"
printf "Vapor fraction at node inlet x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinareax[VOLUME[i]]+1, inavoidgx[VOLUME[i]], \
      nzinareax[VOLUME[i]], voidg[VOLUME[i]]

printf "27\n"
printf "*****\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
invoidfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid fraction at node inlet x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinareax[VOLUME[i]]+1, inavoidfx[VOLUME[i]], \
      nzinareax[VOLUME[i]], voidf[VOLUME[i]]

printf "27\n"
printf "*****\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
invoidfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor fraction at node outlet x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutareax[VOLUME[i]]+1, outavoidgx[VOLUME[i]], \
      nzoutareax[VOLUME[i]], voidg[VOLUME[i]]

printf "27\n"
printf "*****\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
outvoidgx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid fraction at node outlet x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutareax[VOLUME[i]]+1, outavoidfx[VOLUME[i]], \
      nzoutareax[VOLUME[i]], voidf[VOLUME[i]]

printf "27\n"
printf "*****\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
outvoidfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non zero inlet area * voidg x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, inavoidgx[VOLUME[i]], inavoidgx[VOLUME[i]]
printf "14\n"
printf "*****\n"
printf "3\n"
printf "1.0e-10\n"
nzinavoidgx[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Non zero inlet area * voidf x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, inavoidfx[VOLUME[i]], inavoidfx[VOLUME[i]]
printf "14\n"
printf "*****\n"
printf "3\n"
printf "1.0e-10\n"
nzinavoidfx[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Non zero outlet area * voidg x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, outavoidgx[VOLUME[i]], outavoidgx[VOLUME[i]]
printf "14\n"
printf "*****\n"
printf "3\n"
printf "1.0e-10\n"

```

```

nzoutavoidgx [VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Non zero outlet area * voidf x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, outavoidfx[VOLUME[i]], outavoidfx[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzoutavoidfx[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Vapor density at node inlet x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidgx[VOLUME[i]]+1, inavoidrhogx[VOLUME[i]], \
    nzinavoidgx[VOLUME[i]], rhog[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
inrhogx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid density at node inlet x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidfx[VOLUME[i]]+1, inavoidrhofx[VOLUME[i]], \
    nzinavoidfx[VOLUME[i]], rhof[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
inrhofx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor density at node outlet x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidgx[VOLUME[i]]+1, outavoidrhogx[VOLUME[i]], \
    nzoutavoidgx[VOLUME[i]], rhog[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
outrhogx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid density at node outlet x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidfx[VOLUME[i]]+1, outavoidrhofx[VOLUME[i]], \
    nzoutavoidfx[VOLUME[i]], rhof[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
outrhofx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet mixture density x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s\n", nzinareax[VOLUME[i]]+1, inavoidrhogx[VOLUME[i]], \
    inavoidrhofx[VOLUME[i]], nzinareax[VOLUME[i]], \
    rhomix[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*([2]+[3])/[4])+((1.0-[1])*[5])\n"
inrhomix[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet mixture density x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s\n", nzoutareax [VOLUME[i]]+1, outavoidrhogx [VOLUME[i]], \
    outavoidrhofx [VOLUME[i]], nzoutareax [VOLUME[i]], \
    rhomix[VOLUME[i]]

printf "27\n"

```

```

printf "*-----\n"
printf "([1]*([2]+[3])/[4])+((1.0-[1])*[5])\n"
outrhomixx [VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet pressure (vapor) x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidgx[VOLUME[i]]+1, inavoidpjgx[VOLUME[i]], \
    nzinavoidgx[VOLUME[i]], p[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
provinpjgx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet pressure (liquid) x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidfx[VOLUME[i]]+1, inavoidpjfx[VOLUME[i]], \
    nzinavoidfx[VOLUME[i]], p[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
provinpjfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet pressure (vapor) x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidgx[VOLUME[i]]+1, outavoidpjgx[VOLUME[i]], \
    nzoutavoidgx[VOLUME[i]], p[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
provoutpjgx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet pressure (liquid) x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidfx[VOLUME[i]]+1, outavoidpjfx[VOLUME[i]], \
    nzoutavoidfx[VOLUME[i]], p[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
provoutpjfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"

printf "Inlet area y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inlety[j] != "") {
        printf "%s ", ajun[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inlety[j] != ""){
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inareay[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidg y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO

```

```

for(j=1;j<=njuns;j++) {
    if(inletj[j] != "") {
        printf "%s ", avoidg[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletj[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inavoidgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidf y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletj[j] != "") {
        printf "%s ", avoidf[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletj[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inavoidfy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidg * rhog y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletj[j] != "") {
        printf "%s ", avoidrhog[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletj[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inavoidrhogy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidf * rhof y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletj[j] != "") {

```

```

        printf "%s ", avoidrhof[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletj[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inavoidrhofy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidg * rhog * velg y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletj[j] != "") {
        printf "%s ", avoidrhovelg[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletj[j] != "") {
        printf "%s[%s]", inletj[j], auxcount++
    }
}
printf "\n"
inavoidrhovelgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidf * rhog * velf y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletj[j] != "") {
        printf "%s ", avoidrhovelf[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletj[j] != "") {
        printf "%s[%s]", inletj[j], auxcount++
    }
}
printf "\n"
inavoidrhovelfy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidg * pressure y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletj[j] != "") {
        printf "%s ", avoidpjpg[JUNCTION[j]]
    }
}

```

```

}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletj[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inavoidpjgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidf * pressure y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletj[j] != "") {
        printf "%s ", avoidpjf[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletj[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inavoidpjfy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outletj[j] != "") {
        printf "%s ", ajun[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outletj[j] != ""){
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outareay[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidg y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outletj[j] != "") {
        printf "%s ", avoidg[JUNCTION[j]]
    }
}
printf "\n"

```

```

printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outlety[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outavoidgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidf y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outlety[j] != "") {
        printf "%s ", avoidf[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outlety[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outavoidfy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidg * rhog y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outlety[j] != "") {
        printf "%s ", avoidrhog[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outlety[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outavoidrhogy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidf * rhof y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outlety[j] != "") {
        printf "%s ", avoidrhof[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"

```

```

printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outlety[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outavoidrhofy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidg * rhog * velg y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outlety[j] != "") {
        printf "%s ", avoidrhovelg[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outlety[j] != "") {
        printf "%s[%s]", outlety[j], auxcount++
    }
}
printf "\n"
outavoidrhovelgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidf * rhog * velf y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outlety[j] != "") {
        printf "%s ", avoidrhovelf[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outlety[j] != "") {
        printf "%s[%s]", outlety[j], auxcount++
    }
}
printf "\n"
outavoidrhovelfy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidg * pressure y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outlety[j] != "") {
        printf "%s ", avoidpjpg[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2

```



```

for(j=1;j<=njuns;j++) {
    if(outlety[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outavoidpjgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidf * pressure y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONST0
for(j=1;j<=njuns;j++) {
    if(outlety[j] != "") {
        printf "%s ", avoidpjf[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outlety[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outavoidpjfy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non zero inlet area y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, inareay[VOLUME[i]], inareay[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzinareay[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Non zero outlet area y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, outareay[VOLUME[i]], outareay[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzoutareay[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Vapor fraction at node inlet y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinareay[VOLUME[i]]+1, inavoidgy[VOLUME[i]], \
    nzinareay[VOLUME[i]], voidg[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+(1.0-[1])*[4]\n"
invoidgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid fraction at node inlet y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinareay[VOLUME[i]]+1, inavoidfy[VOLUME[i]], \
    nzinareay[VOLUME[i]], voidf[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+(1.0-[1])*[4]\n"

```

```

invoidfy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor fraction at node outlet y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutareay[VOLUME[i]]+1, outavoidgy[VOLUME[i]], \
    nzoutareay[VOLUME[i]], voidg[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+(1.0-[1])*[4])\n"
outvoidgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid fraction at node outlet y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutareay[VOLUME[i]]+1, outavoidfy[VOLUME[i]], \
    nzoutareay[VOLUME[i]], voidf[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+(1.0-[1])*[4])\n"
outvoidfy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non zero inlet area * voidg y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, inavoidgy[VOLUME[i]], inavoidgy[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzinavoidgy[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Non zero inlet area * voidf y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, inavoidfy[VOLUME[i]], inavoidfy[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzinavoidfy[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Non zero outlet area * voidg y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, outavoidgy [VOLUME[i]], outavoidgy [VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzoutavoidgy [VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Non zero outlet area * voidf y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, outavoidfy[VOLUME[i]], outavoidfy[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzoutavoidfy[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Vapor density at node inlet y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidgy[VOLUME[i]]+1, inavoidrhogy[VOLUME[i]], \
    nzinavoidgy[VOLUME[i]], rhog[VOLUME[i]]

printf "27\n"

```

```

printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
inrhogy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid density at node inlet y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidfy[VOLUME[i]]+1, inavoidrhogy[VOLUME[i]], \
    nzinavoidfy[VOLUME[i]], rhof[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
inrhogy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor density at node outlet y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidgy[VOLUME[i]]+1, outavoidrhogy[VOLUME[i]], \
    nzoutavoidgy[VOLUME[i]], rhog[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
outrhogy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid density at node outlet y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidfy[VOLUME[i]]+1, outavoidrhogy[VOLUME[i]], \
    nzoutavoidfy[VOLUME[i]], rhof[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
outrhogy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet mixture density y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s\n", nzinareay[VOLUME[i]]+1, inavoidrhogy[VOLUME[i]], \
    inavoidrhogy[VOLUME[i]], nzinareay[VOLUME[i]], \
    rhomix[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*([2]+[3])/[4])+((1.0-[1])*[5])\n"
inrhomixy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet mixture density y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s\n", nzoutareay[VOLUME[i]]+1, outavoidrhogy[VOLUME[i]], \
    outavoidrhogy[VOLUME[i]], nzoutareay[VOLUME[i]], \
    rhomix[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*([2]+[3])/[4])+((1.0-[1])*[5])\n"
outrhomixy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet pressure (vapor) y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidgy[VOLUME[i]]+1, inavoidpjgy[VOLUME[i]], \
    nzinavoidgy[VOLUME[i]], p[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
inpjgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet pressure (liquid) y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidfy[VOLUME[i]]+1, inavoidpjfy[VOLUME[i]], \
    nzinavoidfy[VOLUME[i]], p[VOLUME[i]]

printf "27\n"

```

```

printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
inpjfy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet pressure (vapor) y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidgy[VOLUME[i]]+1, outavoidpjgy[VOLUME[i]], \
    nzoutavoidgy[VOLUME[i]], p[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
outpjgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet pressure (liquid) y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidfy[VOLUME[i]]+1, outavoidpjfy[VOLUME[i]], \
    nzoutavoidfy[VOLUME[i]], p[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
outpjfy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"

printf "Inlet area z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletz[j] != "") {
        printf "%s ", ajun[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletz[j] != ""){
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inareaz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidg z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletz[j] != "") {
        printf "%s ", avoidg[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletz[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inavoidgz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"

```

```

printf "Inlet area * voidf z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletz[j] != "") {
        printf "%s ", avoidf[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletz[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inavoidfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidg * rhog z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletz[j] != "") {
        printf "%s ", avoidrhog[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletz[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inavoidrhogz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidf * rhof z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletz[j] != "") {
        printf "%s ", avoidrhof[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletz[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inavoidrhofz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidg * rhog * velg z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT

```

```

printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletz[j] != "") {
        printf "%s ", avoidrhovelg[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletz[j] != "") {
        printf "%s[%s]", inletz[j], auxcount++
    }
}
printf "\n"
inavoidrhovelgz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidf * rhog * velf z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletz[j] != "") {
        printf "%s ", avoidrhovelf[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletz[j] != "") {
        printf "%s[%s]", inletz[j], auxcount++
    }
}
printf "\n"
inavoidrhovelfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidg * pressure z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(inletz[j] != "") {
        printf "%s ", avoidpjpg[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(inletz[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
inavoidpjpgz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet area * voidf * pressure z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {

```

```

        if(inletz[j] != "") {
            printf "%s ", avoidpjf[JUNCTION[j]]
        }
    }
    printf "\n"
    printf "27\n"
    printf "*-----\n"
    printf "[1]"
    auxcount = 2
    for(j=1;j<=njuns;j++) {
        if(inletz[j] != "") {
            printf "+[%s]", auxcount++
        }
    }
    printf "\n"
    inavoidpjfz[VOLUME[i]] = OUTCOUNT++
    printf "***** \n"
    printf "Outlet area z %s\n", VOLUME[i]
    printf "%s\n", OUTCOUNT
    printf "%s ", CONSTO
    for(j=1;j<=njuns;j++) {
        if(outletz[j] != "") {
            printf "%s ", ajun[JUNCTION[j]]
        }
    }
    printf "\n"
    printf "27\n"
    printf "*-----\n"
    printf "[1]"
    auxcount = 2
    for(j=1;j<=njuns;j++) {
        if(outletz[j] != ""){
            printf "+[%s]", auxcount++
        }
    }
    printf "\n"
    outareaz[VOLUME[i]] = OUTCOUNT++
    printf "***** \n"
    printf "Outlet area * voidg z %s\n", VOLUME[i]
    printf "%s\n", OUTCOUNT
    printf "%s ", CONSTO
    for(j=1;j<=njuns;j++) {
        if(outletz[j] != "") {
            printf "%s ", avoidg[JUNCTION[j]]
        }
    }
    printf "\n"
    printf "27\n"
    printf "*-----\n"
    printf "[1]"
    auxcount = 2
    for(j=1;j<=njuns;j++) {
        if(outletz[j] != "") {
            printf "+[%s]", auxcount++
        }
    }
    printf "\n"
    outavoidgz[VOLUME[i]] = OUTCOUNT++
    printf "***** \n"
    printf "Outlet area * voidf z %s\n", VOLUME[i]
    printf "%s\n", OUTCOUNT
    printf "%s ", CONSTO
    for(j=1;j<=njuns;j++) {
        if(outletz[j] != "") {
            printf "%s ", avoidf[JUNCTION[j]]
        }
    }

```

```

    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outletz[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outavoidfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidg * rhog z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outletz[j] != "") {
        printf "%s ", avoidrhog[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outletz[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outavoidrhogz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidf * rhof z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outletz[j] != "") {
        printf "%s ", avoidrhof[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outletz[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outavoidrhofz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidg * rhog * velg z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outletz[j] != "") {
        printf "%s ", avoidrhovelg[JUNCTION[j]]
    }
}
}

```



```

printf "\n"
printf "27\n"
printf "-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outletz[j] != "") {
        printf "%s[%s]", outletz[j], auxcount++
    }
}
printf "\n"
outavoidrhovelgz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidf * rhog * velf z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outletz[j] != "") {
        printf "%s ", avoidrhovelf[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outletz[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outavoidrhovelfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidg * pressure z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outletz[j] != "") {
        printf "%s ", avoidpjpg[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"
printf "-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outletz[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outavoidpjpgz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet area * voidf * pressure z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(j=1;j<=njuns;j++) {
    if(outletz[j] != "") {
        printf "%s ", avoidpjf[JUNCTION[j]]
    }
}
printf "\n"
printf "27\n"

```

```

printf "*-----\n"
printf "[1]"
auxcount = 2
for(j=1;j<=njuns;j++) {
    if(outletz[j] != "") {
        printf "+[%s]", auxcount++
    }
}
printf "\n"
outavoidpjfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non zero inlet area z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, inareaz[VOLUME[i]], inareaz[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzinareaz[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Non zero outlet area z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, outareaz[VOLUME[i]], outareaz[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzoutareaz[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Vapor fraction at node inlet z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinareaz[VOLUME[i]]+1, inavoidgz[VOLUME[i]], \
    nzinareaz[VOLUME[i]], voidg[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
invoidgz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid fraction at node inlet z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinareaz[VOLUME[i]]+1, inavoidfz[VOLUME[i]], \
    nzinareaz[VOLUME[i]], voidf[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
invoidfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor fraction at node outlet z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutareaz[VOLUME[i]]+1, outavoidgz[VOLUME[i]], \
    nzoutareaz[VOLUME[i]], voidg[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
outvoidgz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid fraction at node outlet z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutareaz[VOLUME[i]]+1, outavoidfz[VOLUME[i]], \
    nzoutareaz[VOLUME[i]], voidf[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"

```

```

outvoidfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non zero inlet area * voidf z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, inavoidgz[VOLUME[i]], inavoidgz[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzinavoidgz[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Non zero inlet area * voidf z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, inavoidfz[VOLUME[i]], inavoidfz[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzinavoidfz[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Non zero outlet area * voidf z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, outavoidgz [VOLUME[i]], outavoidgz [VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzoutavoidgz [VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Non zero outlet area * voidf z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, outavoidfz[VOLUME[i]], outavoidfz[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzoutavoidfz[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Vapor density at node inlet z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidgz[VOLUME[i]]+1, inavoidrhogz[VOLUME[i]], \
    nzinavoidgz[VOLUME[i]], rhog[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
inrhogz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid density at node inlet z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidfz[VOLUME[i]]+1, inavoidrhofz[VOLUME[i]], \
    nzinavoidfz[VOLUME[i]], rhof[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
inrhofz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor density at node outlet z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidgz[VOLUME[i]]+1, outavoidrhogz[VOLUME[i]], \
    nzoutavoidgz[VOLUME[i]], rhog[VOLUME[i]]

printf "27\n"

```

```

printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
outrhogz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid density at node outlet z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidfz[VOLUME[i]]+1, outavoidrhofz[VOLUME[i]], \
      nzoutavoidfz[VOLUME[i]], rhof[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
outrhofz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet mixture density z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s\n", nzinareaz[VOLUME[i]]+1, inavoidrhogz[VOLUME[i]], \
      inavoidrhofz[VOLUME[i]], nzinareaz[VOLUME[i]], \
      rhomix[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*([2]+[3])/[4])+((1.0-[1])*[5])\n"
inrhomixz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet mixture density z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s\n", nzoutareaz[VOLUME[i]]+1, outavoidrhogz[VOLUME[i]], \
      outavoidrhofz[VOLUME[i]], nzoutareaz[VOLUME[i]], \
      rhomix[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*([2]+[3])/[4])+((1.0-[1])*[5])\n"
outrhomixz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet pressure (vapor) z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidgz[VOLUME[i]]+1, inavoidpjgz[VOLUME[i]], \
      nzinavoidgz[VOLUME[i]], p[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
inpjgz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet pressure (liquid) z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidfz[VOLUME[i]]+1, inavoidpjfz[VOLUME[i]], \
      nzinavoidfz[VOLUME[i]], p[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
inpjfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet pressure (vapor) z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidgz[VOLUME[i]]+1, outavoidpjgz[VOLUME[i]], \
      nzoutavoidgz[VOLUME[i]], p[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
outpjgz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet pressure (liquid) z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidfz[VOLUME[i]]+1, outavoidpjfz[VOLUME[i]], \
      nzoutavoidfz[VOLUME[i]], p[VOLUME[i]]

printf "27\n"

```

```

printf "*-----\n"
printf "([1]*[2]/[3])+((1.0-[1])*[4])\n"
outpjfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non-zero inpjgx %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, provinpjgx[VOLUME[i]], TIME
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "%s\n", INITTIME-MAXDT/2.0
inpjgx[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Non-zero outpjgx %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, provoutpjgx[VOLUME[i]], TIME
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "%s\n", INITTIME-MAXDT/2.0
outpjgx[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "In - out vapor pressure %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inpjgx[VOLUME[i]], outpjgx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
deltapgx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non-zero inpjfx %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, provinpjfx[VOLUME[i]], TIME
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "%s\n", INITTIME-MAXDT/2.0
inpjfx[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Non-zero outpjfx %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, provoutpjfx[VOLUME[i]], TIME
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "%s\n", INITTIME-MAXDT/2.0
outpjfx[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "In - out liquid pressure %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inpjfx[VOLUME[i]], outpjfx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
deltapfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non zero inavoidrhogx %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, inavoidrhogx[VOLUME[i]], inavoidrhogx[VOLUME[i]]
printf "14\n"
printf "*-----\n"

```

```

printf "3\n"
printf "1.0e-10\n"
nzinavoidrhogx[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Inlet vapor volumetric flow x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidrhogx[VOLUME[i]]+1, inavoidgx[VOLUME[i]], \
    inavoidrhovelgx[VOLUME[i]], nzinavoidrhogx[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]/[4]\n"
invflowgx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non zero inavoidrhof x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, inavoidrhofx[VOLUME[i]], inavoidrhofx[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzinavoidrhofx[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Inlet liquid volumetric flow x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidrhofx[VOLUME[i]]+1, inavoidfx[VOLUME[i]], \
    inavoidrhovelfx[VOLUME[i]], nzinavoidrhofx[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]/[4]\n"
invflowfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non zero outavoidrhog x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, outavoidrhogx[VOLUME[i]], outavoidrhogx[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzoutavoidrhogx[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Outlet vapor volumetric flow x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidrhogx [VOLUME[i]]+1, outavoidgx [VOLUME[i]], \
    outavoidrhovelgx [VOLUME[i]], nzoutavoidrhogx [VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]/[4]\n"
outvflowgx [VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non zero outavoidrhof x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, outavoidrhofx[VOLUME[i]], outavoidrhofx[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzoutavoidrhofx[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Outlet liquid volumetric flow x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidrhofx[VOLUME[i]]+1, outavoidfx[VOLUME[i]], \

```

```

outavoidrhovelfx[VOLUME[i]], nzoutavoidrhovfx[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]/[4]\n"
outvflowfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor inlet velocity x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", nzinavoidgx[VOLUME[i]]+1, invflowgx[VOLUME[i]], \
    nzinavoidgx[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]/[3]\n"
invelgx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid inlet velocity x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", nzinavoidfx[VOLUME[i]]+1, invflowfx[VOLUME[i]], \
    nzinavoidfx[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]/[3]\n"
invelfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor outlet velocity x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", nzoutavoidgx[VOLUME[i]]+1, outvflowgx[VOLUME[i]], \
    nzoutavoidgx[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]/[3]\n"
outvelgx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid outlet velocity x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", nzoutavoidfx[VOLUME[i]]+1, outvflowfx[VOLUME[i]], \
    nzoutavoidfx[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]/[3]\n"
outvelfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor inlet momentum flow x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inavoidrhovelfx[VOLUME[i]], invelgx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
inmomflowgx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid inlet momentum flow x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inavoidrhovelfx[VOLUME[i]], invelfx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
inmomflowfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet momentum flow x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inmomflowgx[VOLUME[i]], inmomflowfx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
inmomflowx[VOLUME[i]] = OUTCOUNT++

```

```

printf "***** \n"
printf "Inlet slip x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", invelgx[VOLUME[i]], invelfx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
inslipx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet (A * rhog * voidg) * (A * rhof * voidf) x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inavoidrhogx[VOLUME[i]], inavoidrhofx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
inprodavoidrhox[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet non-homogeneous convective term x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", inprodavoidrhox[VOLUME[i]], inslipx[VOLUME[i]],
    nzinareax[VOLUME[i]], inrhomixx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[2]/([3]*[4])\n"
innonhmx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor outlet momentum flow x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outavoidrhovelgx[VOLUME[i]], outvelgx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
outmomflowgx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid outlet momentum flow x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outavoidrhovelfx[VOLUME[i]], outvelfx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
outmomflowfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet momentum flow x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outmomflowgx[VOLUME[i]], outmomflowfx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
outmomflowx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet slip x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outvelgx[VOLUME[i]], outvelfx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
outslipx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet (A * rhog * voidg) * (A * rhof * voidf) x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outavoidrhogx[VOLUME[i]], outavoidrhofx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
outprodavoidrhox[VOLUME[i]] = OUTCOUNT++

```



```

printf "***** \n"
printf "Outlet non-homogeneous convective term x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", outprodavoidrhox[VOLUME[i]], outslipx[VOLUME[i]],
      nzinareax[VOLUME[i]], outrhomixx[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[2]/([3]*[4])\n"
outnonhmx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"

printf "Non zero inavoidrhogy %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, inavoidrhogy[VOLUME[i]], inavoidrhogy[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzinavoidrhogy[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Inlet vapor volumetric flow y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidrhogy[VOLUME[i]]+1, inavoidgy[VOLUME[i]], \
      inavoidrhovelgy[VOLUME[i]], nzinavoidrhogy[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]/[4]\n"
invflowgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non zero inavoidrhof y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, inavoidrhofy[VOLUME[i]], inavoidrhofy[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzinavoidrhofy[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Inlet liquid volumetric flow y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidrhofy[VOLUME[i]]+1, inavoidfy[VOLUME[i]], \
      inavoidrhovelgy[VOLUME[i]], nzinavoidrhofy[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]/[4]\n"
invflowfy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non zero outavoidrhog y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, outavoidrhogy[VOLUME[i]], outavoidrhogy[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzoutavoidrhogy[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Outlet vapor volumetric flow y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidrhogy[VOLUME[i]]+1, outavoidgy[VOLUME[i]], \
      outavoidrhovelgy[VOLUME[i]], nzoutavoidrhogy[VOLUME[i]]

```

```

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]/[4]\n"
outvflowgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non zero outavoidrhof y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, outavoidrhofy[VOLUME[i]], outavoidrhofy[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzoutavoidrhofy[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Outlet liquid volumetric flow y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidrhofy[VOLUME[i]]+1, outavoidfy[VOLUME[i]], \
    outavoidrhovelgy[VOLUME[i]], nzoutavoidrhofy[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]/[4]\n"
outvflowgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor inlet velocity y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", nzinavoidgy[VOLUME[i]]+1, invflowgy[VOLUME[i]], \
    nzinavoidgy[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]/[3]\n"
invelgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid inlet velocity y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", nzinavoidfy[VOLUME[i]]+1, invflowfy[VOLUME[i]], \
    nzinavoidfy[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]/[3]\n"
invelgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor outlet velocity y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", nzoutavoidgy[VOLUME[i]]+1, outvflowgy[VOLUME[i]], \
    nzoutavoidgy[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]/[3]\n"
outvelgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid outlet velocity y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", nzoutavoidfy[VOLUME[i]]+1, outvflowfy[VOLUME[i]], \
    nzoutavoidfy[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]/[3]\n"
outvelgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor inlet momentum flow y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inavoidrhovelgy[VOLUME[i]], invelgy[VOLUME[i]]
printf "27\n"
printf "*-----\n"

```

```

printf "[1]*[2]\n"
inmomflowgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid inlet momentum flow y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inavoidrhovelfy[VOLUME[i]], invelfy[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
inmomflowgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet momentum flow y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inmomflowgy[VOLUME[i]], inmomflowgy[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
inmomflowgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet slip y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", invelgy[VOLUME[i]], invelfy[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
inslipy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet (A * rhog * voidg) * (A * rhof * voidf) y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inavoidrhogy[VOLUME[i]], inavoidrhofx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
inprodavoidrhoy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet non-homogeneous convective term y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", inprodavoidrhoy[VOLUME[i]], inslipy[VOLUME[i]],
    nzinareay[VOLUME[i]], inrhomixy[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[2]/([3]*[4])\n"
innonhmy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor outlet momentum flow y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outavoidrhovelfy[VOLUME[i]], outvelgy[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
outmomflowgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid outlet momentum flow y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outavoidrhovelfy[VOLUME[i]], outvelgy[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
outmomflowgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet momentum flow y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outmomflowgy[VOLUME[i]], outmomflowgy[VOLUME[i]]
printf "27\n"
printf "*-----\n"

```

```

printf "[1]+[2]\n"
outmomflowy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet slip y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outvelgy[VOLUME[i]],outvelfy[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
outslipy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet (A * rhog * voidg) * (A * rhof * voidf) y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n",outavoidrhogy[VOLUME[i]],outavoidrhofx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
outprodavoidrhoy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet non-homogeneous convective term y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n",outprodavoidrhoy[VOLUME[i]],outslipy[VOLUME[i]],
nzinareay[VOLUME[i]],outrhomixy[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[2]/([3]*[4])\n"
outnonhmy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"

printf "Non zero inavoidrhogz %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, inavoidrhogz[VOLUME[i]], inavoidrhogz[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzinavoidrhogz[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Inlet vapor volumetric flow z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidrhogz[VOLUME[i]]+1, inavoidgz[VOLUME[i]], \
inavoidrhoelgz[VOLUME[i]], nzinavoidrhogz[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]/[4]\n"
invflowgz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non zero inavoidrhof z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, inavoidrhofz[VOLUME[i]], inavoidrhofz[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzinavoidrhofz[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Inlet liquid volumetric flow z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzinavoidrhofz[VOLUME[i]]+1, inavoidfz[VOLUME[i]], \
inavoidrhoelfz[VOLUME[i]], nzinavoidrhofz[VOLUME[i]]

printf "27\n"
printf "*-----\n"

```

```

printf "[1]*[2]*[3]/[4]\n"
invflowfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non zero outavoidrhog z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, outavoidrhogz[VOLUME[i]], outavoidrhogz[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzoutavoidrhogz[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Outlet vapor volumetric flow z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidrhogz [VOLUME[i]]+1, outavoidgz [VOLUME[i]], \
outavoidrhovelgz [VOLUME[i]], nzoutavoidrhogz [VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]/[4]\n"
outvflowgz [VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non zero outavoidrhof z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, outavoidrhofz[VOLUME[i]], outavoidrhofz[VOLUME[i]]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzoutavoidrhofz[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Outlet liquid volumetric flow z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", nzoutavoidrhofz[VOLUME[i]]+1, outavoidfz[VOLUME[i]], \
outavoidrhovelgz[VOLUME[i]], nzoutavoidrhofz[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]/[4]\n"
outvflowfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor inlet velocity y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", nzinavoidgz[VOLUME[i]]+1, invflowgz[VOLUME[i]], \
nzinavoidgz[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]/[3]\n"
invelgz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid inlet velocity z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", nzinavoidfz[VOLUME[i]]+1, invflowfz[VOLUME[i]], \
nzinavoidfz[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]/[3]\n"
invelfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor outlet velocity z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", nzoutavoidgz[VOLUME[i]]+1, outvflowgz[VOLUME[i]], \
nzoutavoidgz[VOLUME[i]]

printf "27\n"
printf "*-----\n"

```

```

printf "[1]*[2]/[3]\n"
outvelgz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid outlet velocity z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", nzoutavoidfz[VOLUME[i]]+1, outvflowfz[VOLUME[i]], \
      nzoutavoidfz[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]/[3]\n"
outvelfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor inlet momentum flow z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inavoidrhovelgz[VOLUME[i]], invelgz[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
inmomflowgz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid inlet momentum flow z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inavoidrhovelfz[VOLUME[i]], invelfz[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
inmomflowfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet momentum flow z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inmomflowgz[VOLUME[i]], inmomflowfz[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
inmomflowz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet slip z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", invelgz[VOLUME[i]], invelfz[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
inslipz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet (A * rhog * voidg) * (A * rhof * voidf) z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inavoidrhogz[VOLUME[i]], inavoidrhofx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
inprodavoidrhoz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet non-homogeneous convective term z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", inprodavoidrhoz[VOLUME[i]], inslipz[VOLUME[i]],
      nzinareaz[VOLUME[i]], inrhomixz[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[2]/([3]*[4])\n"
innonhmxz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor outlet momentum flow z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outavoidrhovelgz[VOLUME[i]], outvelgz[VOLUME[i]]
printf "27\n"

```

```

printf "*-----\n"
printf "[1]*[2]\n"
outmomflowgz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid outlet momentum flow z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outavoidrhovelfz[VOLUME[i]], outvelfz[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
outmomflowfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet momentum flow z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outmomflowgz[VOLUME[i]], outmomflowfz[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
outmomflowz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet slip z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outvelgz[VOLUME[i]], outvelfz[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
outslipz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet (A * rhog * voidg) * (A * rhof * voidf) z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outavoidrhogz[VOLUME[i]], outavoidrhofx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
outprodavoidrhoz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet non-homogeneous convective term z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", outprodavoidrhoz[VOLUME[i]], outslipz[VOLUME[i]],
    nzinareaz[VOLUME[i]], outrhomixz[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[2]/([3]*[4])\n"
outnonhmx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"

printf "In -out mixture momentum flow %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s\n", inmomflowx[VOLUME[i]], inmomflowy[VOLUME[i]],
    inmomflowz[VOLUME[i]], outmomflowx[VOLUME[i]],
    outmomflowy[VOLUME[i]], outmomflowz[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]+[2]+[3])-( [4]+[5]+[6])\n"
mommixconv[VOLUME[i]] = OUTCOUNT++
printf "***** \n"

printf "In - out non homogeneous convective term %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s\n", innonhmx[VOLUME[i]], innonhmy[VOLUME[i]],
    innonhmx[VOLUME[i]], outnonhmx[VOLUME[i]],
    outnonhmy[VOLUME[i]], outnonhmx[VOLUME[i]]

printf "27\n"

```

```

printf "*-----\n"
printf "([1]+[2]+[3])-([4]+[5]+[6])\n"
nonhmconv[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Pressure drop in volume %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s %s %s\n", wfricg[VOLUME[i]], wfricf[VOLUME[i]],
    buoyg[VOLUME[i]], buoyf[VOLUME[i]],
    mommixconv[VOLUME[i]], nonhmconv[VOLUME[i]],
    dermommix[VOLUME[i]], avol[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "([1]+[2]+[3]+[4]-[5]-[6]+[7])/[8]\n"
deltap[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet pressure in volume %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", p[VOLUME[i]], deltap[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+0.5*[2]\n"
inp[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet pressure in volume %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", p[VOLUME[i]], deltap[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-0.5*[2]\n"
outp[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor acoustic term %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s\n", avol[VOLUME[i]], voidg[VOLUME[i]], \
    compg[VOLUME[i]], velg[VOLUME[i]], \
    inpjgx[VOLUME[i]], outpjgx[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]*[4]*LN([5]/[6])\n"
acg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid acoustic term %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s\n", avol[VOLUME[i]], voidf[VOLUME[i]], \
    compf[VOLUME[i]], velf[VOLUME[i]], \
    inpjfx[VOLUME[i]], outpjfx[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]*[3]*[4]*LN([5]/[6])\n"
acf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"

# printf "Inlet mixture velocity x %s\n", VOLUME[i]
# printf "%s\n", OUTCOUNT
# printf "%s %s %s %s %s %s %s %s\n", invoidgx[VOLUME[i]], inrhogx[VOLUME[i]], \
#     invelgx[VOLUME[i]], invoidfx[VOLUME[i]], \
#     inrhofx[VOLUME[i]], invelfx[VOLUME[i]], \
#     inrhomixx[VOLUME[i]]

# printf "27\n"
# printf "*-----\n"
# printf "([1]*[2]*[3]+[4]*[5]*[6])/[7]\n"
# inmixvelx[VOLUME[i]] = OUTCOUNT++
# printf "***** \n"
# printf "Outlet mixture velocity x %s\n", VOLUME[i]

```



```

#   printf "%s\n", OUTCOUNT
#   printf "%s %s %s %s %s %s\n", outvoidgx[VOLUME[i]], outrhogx[VOLUME[i]], \
#   outvelgx[VOLUME[i]], outvoidfx[VOLUME[i]], \
#   outrhofx[VOLUME[i]], outvelfx[VOLUME[i]], \
#   outrhomixx[VOLUME[i]]
#   printf "27\n"
#   printf "*-----\n"
#   printf "([1]*[2]*[3]+[4]*[5]*[6])/[7]\n"
#   outmixvelx[VOLUME[i]] = OUTCOUNT++
#   printf "***** \n"
printf "Volumetric flow in - volumetric flow out (vapor) %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s\n", invflowgx[VOLUME[i]], invflowgy[VOLUME[i]], \
    invflowgz[VOLUME[i]], outvflowgx[VOLUME[i]], \
    outvflowgy[VOLUME[i]], outvflowgz[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]+[2]+[3]-[4]-[5]-[6]\n"
jtermg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Volumetric flow in - volumetric flow out (liquid) %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s\n", invflowfx[VOLUME[i]], invflowfy[VOLUME[i]], \
    invflowfz[VOLUME[i]], outvflowfx[VOLUME[i]], \
    outvflowfy[VOLUME[i]], outvflowfz[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]+[2]+[3]-[4]-[5]-[6]\n"
jtermf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Volumetric term + thermal exp. + swelling (vapor) %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", jtermg[VOLUME[i]], thswg[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
jthswg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Volumetric term + thermal exp. + swelling (liquid) %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", jtermf[VOLUME[i]], thswf[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
jthswf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vol. term + thermal exp. + swelling + ac. term (vapor) %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", jthswg[VOLUME[i]], acg[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
jthswacg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vol. term + thermal exp. + swelling + ac. term (liquid) %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", jthswf[VOLUME[i]], acf[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
jthswacf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non filtered postprocessed time derivative of ln p %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", jthswacg[VOLUME[i]], jthswacf[VOLUME[i]], \

```

```

                nztotncomp[VOLUME[i]], nztotncomp[VOLUME[i]]+1
printf "27\n"
printf "*-----\n"
printf "[4]*([1]+[2])/[3]\n"
nfpostdlnpdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non-zero pressure in node %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, p[VOLUME[i]], TIME
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "%s\n", INITTIME-MAXDT/2.0
nzp[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "ln pressure in node %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s\n", nzp[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "LN[1]\n"
lnp[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non-zero avol in volume %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, avol[VOLUME[i]], TIME
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "%s\n", INITTIME-MAXDT/2.0
nzavol[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Second term of vapor acoustic term in volume %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", dervoidcompv[VOLUME[i]], vvov[VOLUME[i]], nzavol[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]/[3]\n"
ac2g[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Second term of liquid acoustic term in volume %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", dervoidcompf[VOLUME[i]], vvov[VOLUME[i]], nzavol[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]/[3]\n"
ac2f[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "First term of vapor acoustic term in volume %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", lnp[VOLUME[i]], ac2g[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
ac1g[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "First term of liquid acoustic term in volume %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", lnp[VOLUME[i]], ac2f[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
ac1f[VOLUME[i]] = OUTCOUNT++

```

```

printf "***** \n"
printf "Time derivative of ln p in node %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s\n", lnp[VOLUME[i]]
printf "43\n"
printf "*-----\n"
printf "16 1.0\n"
dlnpdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Non filtered post. time derivative of liquid volume %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", jthswacf[VOLUME[i]], \
nodecompf[VOLUME[i]], nfpostdlnpdt[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]-( [2]*[3] )\n"
nfpostdvfdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Postprocessed time derivative of vapor mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s %s %s\n", inavoidrhovelgx[VOLUME[i]], \
inavoidrhovelgy[VOLUME[i]], \
inavoidrhovelgz[VOLUME[i]], \
outavoidrhovelgx[VOLUME[i]], \
outavoidrhovelgy[VOLUME[i]], \
outavoidrhovelgz[VOLUME[i]], \
vvol[VOLUME[i]], gammag[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]+[2]+[3]-[4]-[5]-[6]+([7]*[8])\n"
nfpostdmgdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Time derivative of vapor mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s\n", massg[VOLUME[i]]
printf "43\n"
printf "*-----\n"
printf "16 1.0\n"
dmgdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Postprocessed time derivative of liquid mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s %s %s\n", inavoidrhovelfx[VOLUME[i]], \
inavoidrhovelfy[VOLUME[i]], \
inavoidrhovelfz[VOLUME[i]], \
outavoidrhovelfx[VOLUME[i]], \
outavoidrhovelfy[VOLUME[i]], \
outavoidrhovelfz[VOLUME[i]], \
vvol[VOLUME[i]], gammaf[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]+[2]+[3]-[4]-[5]-[6]+([7]*[8])\n"
nfpostdmfdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Time derivative of liquid mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s\n", massf[VOLUME[i]]
printf "43\n"
printf "*-----\n"
printf "16 1.0\n"
dmfdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Density of saturated vapor at node inlet x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inpjgx[VOLUME[i]], CONST1

```

```

printf "62\n"
printf "*-----\n"
printf "P x\n"
printf "rhog\n"
insatrhogx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Density of saturated liquid at node inlet x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inpjfx[VOLUME[i]], CONST0
printf "62\n"
printf "*-----\n"
printf "P x\n"
printf "rhof\n"
insatrhofx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Density of saturated vapor at node outlet x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outpjgx[VOLUME[i]], CONST1
printf "62\n"
printf "*-----\n"
printf "P x\n"
printf "rhog\n"
outsatrhogx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Density of saturated liquid at node outlet x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outpjfx[VOLUME[i]], CONST0
printf "62\n"
printf "*-----\n"
printf "P x\n"
printf "rhof\n"
outsatrhofx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Density of saturated vapor at node inlet y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inpjgy[VOLUME[i]], CONST1
printf "62\n"
printf "*-----\n"
printf "P x\n"
printf "rhog\n"
insatrhogy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Density of saturated liquid at node inlet y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inpjfy[VOLUME[i]], CONST0
printf "62\n"
printf "*-----\n"
printf "P x\n"
printf "rhof\n"
insatrhofy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Density of saturated vapor at node outlet y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outpjgy[VOLUME[i]], CONST1
printf "62\n"
printf "*-----\n"
printf "P x\n"
printf "rhog\n"
outsatrhogy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Density of saturated liquid at node outlet y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outpjfy[VOLUME[i]], CONST0
printf "62\n"
printf "*-----\n"

```

```

printf "P x\n"
printf "rhof\n"
outsatrhofy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Density of saturated vapor at node inlet z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inpjgz[VOLUME[i]], CONST1
printf "62\n"
printf "*-----\n"
printf "P x\n"
printf "rhog\n"
insatrhogz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Density of saturated liquid at node inlet z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", inpjfz[VOLUME[i]], CONST0
printf "62\n"
printf "*-----\n"
printf "P x\n"
printf "rhof\n"
insatrhofz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Density of saturated vapor at node outlet z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outpjgz[VOLUME[i]], CONST1
printf "62\n"
printf "*-----\n"
printf "P x\n"
printf "rhog\n"
outsatrhogz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Density of saturated liquid at node outlet z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outpjfz[VOLUME[i]], CONST0
printf "62\n"
printf "*-----\n"
printf "P x\n"
printf "rhof\n"
outsatrhofz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor equilibrium thermal expansion * -1 %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s\n", satcompz[VOLUME[i]], nzcompz[VOLUME[i]], \
    jthswg[VOLUME[i]], nfpostdvfdt[VOLUME[i]], \
    nzcompz[VOLUME[i]]+1

printf "27\n"
printf "*-----\n"
printf "[5]*([1]-[2])*([3]+[4])/[2]\n"
eqthexpz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid equilibrium thermal expansion * -1 %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s\n", satcompf[VOLUME[i]], nzcompf[VOLUME[i]], \
    jthswf[VOLUME[i]], nfpostdvfdt[VOLUME[i]], \
    nzcompf[VOLUME[i]]+1

printf "27\n"
printf "*-----\n"
printf "[5]*([1]-[2])*([3]-[4])/[2]\n"
eqthexpf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet saturated mass flow for vapor phase x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", invflowgx[VOLUME[i]], insatrhogx[VOLUME[i]]
printf "27\n"
printf "*-----\n"

```

```

printf "[1]*[2]\n"
insatmflowgx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet saturated mass flow for liquid phase x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", invflowfx[VOLUME[i]], insatrhofx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
insatmflowfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet saturated mass flow for vapor phase x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outvflowgx[VOLUME[i]], outsatrhogx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
outsatmflowgx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet saturated mass flow for liquid phase x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outvflowfx[VOLUME[i]], outsatrhofx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
outsatmflowfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet non-equilibrium mass flow for vapor phase x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", insatmflowgx[VOLUME[i]], inavoidrhovelgx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
innqmflowgx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet non-equilibrium mass flow for liquid phase x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", insatmflowfx[VOLUME[i]], inavoidrhovelfx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
innqmflowfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet non-equilibrium mass flow for vapor phase x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outsatmflowgx[VOLUME[i]], outavoidrhovelgx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
outnqmflowgx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet non-equilibrium mass flow for liquid phase x %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outsatmflowfx[VOLUME[i]], outavoidrhovelfx[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
outnqmflowfx[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet saturated mass flow for vapor phase y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", invflowgy[VOLUME[i]], insatrhogy[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"

```

```

insatmflowgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet saturated mass flow for liquid phase y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", invflowfy[VOLUME[i]], insatrhofy[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
insatmflowfy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet saturated mass flow for vapor phase y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outvflowgy[VOLUME[i]], outsatrhogy[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
outsatmflowgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet saturated mass flow for liquid phase y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outvflowfy[VOLUME[i]], outsatrhofy[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
outsatmflowfy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet non-equilibrium mass flow for vapor phase y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", insatmflowgy[VOLUME[i]], inavoidrhovelgy[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
innqmflowgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet non-equilibrium mass flow for liquid phase y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", insatmflowfy[VOLUME[i]], inavoidrhovelgy[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
innqmflowfy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet non-equilibrium mass flow for vapor phase y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outsatmflowgy[VOLUME[i]], outavoidrhovelgy[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
outnqmflowgy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet non-equilibrium mass flow for liquid phase y %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outsatmflowfy[VOLUME[i]], outavoidrhovelgy[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
outnqmflowfy[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet saturated mass flow for vapor phase z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", invflowgz[VOLUME[i]], insatrhogz[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
insatmflowgz[VOLUME[i]] = OUTCOUNT++

```

```

printf "***** \n"
printf "Inlet saturated mass flow for liquid phase z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", invflowfz[VOLUME[i]], insatrhofz[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
insatmflowfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet saturated mass flow for vapor phase z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outvflowgz[VOLUME[i]], outsatrhogz[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
outsatmflowgz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet saturated mass flow for liquid phase z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outvflowfz[VOLUME[i]], outsatrhofz[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
outsatmflowfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet non-equilibrium mass flow for vapor phase z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", insatmflowgz[VOLUME[i]], inavoidrhovelgz[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
innqmflowgz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet non-equilibrium mass flow for liquid phase z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", insatmflowfz[VOLUME[i]], inavoidrhovelfz[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
innqmflowfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet non-equilibrium mass flow for vapor phase z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outsatmflowgz[VOLUME[i]], outavoidrhovelgz[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
outnqmflowgz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet non-equilibrium mass flow for liquid phase z %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", outsatmflowfz[VOLUME[i]], outavoidrhovelfz[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]-[2]\n"
outnqmflowfz[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet mass flow for vapor phase %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", inavoidrhovelgx[VOLUME[i]], inavoidrhovelgy[VOLUME[i]], \
    inavoidrhovelgz[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]+[3]\n"
inmflowg[VOLUME[i]] = OUTCOUNT++

```



```

printf "***** \n"
printf "Outlet mass flow for vapor phase %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", outavoidrhovelgx[VOLUME[i]], outavoidrhovelgy[VOLUME[i]], \
    outavoidrhovelgz[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]+[2]+[3]\n"
outmflowg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet mass flow for liquid phase %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", inavoidrhovelfx[VOLUME[i]], inavoidrhovelfy[VOLUME[i]], \
    inavoidrhovelfz[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]+[2]+[3]\n"
inmflowf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet mass flow for liquid phase %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", outavoidrhovelfx[VOLUME[i]], outavoidrhovelfy[VOLUME[i]], \
    outavoidrhovelfz[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]+[2]+[3]\n"
outmflowf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet non-equilibrium mass flow for vapor phase %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", innqmflogx[VOLUME[i]], innqmflogy[VOLUME[i]], \
    innqmflogz[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]+[2]+[3]\n"
innqmflog[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet non-equilibrium mass flow for vapor phase %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", outnqmflogx[VOLUME[i]], outnqmflogy[VOLUME[i]], \
    outnqmflogz[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]+[2]+[3]\n"
outnqmflog[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Inlet non-equilibrium mass flow for liquid phase %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", innqmflofx[VOLUME[i]], innqmflofy[VOLUME[i]], \
    innqmflofz[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]+[2]+[3]\n"
innqmflof[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Outlet non-equilibrium mass flow for liquid phase %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", outnqmflofx[VOLUME[i]], outnqmflofy[VOLUME[i]], \
    outnqmflofz[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[1]+[2]+[3]\n"
outnqmflof[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Vapor non-equilibrium thermal expansion %s\n", VOLUME[i]

```

```

printf "%s\n", OUTCOUNT
printf "%s %s %s\n", satrhog[VOLUME[i]], thexpg[VOLUME[i]], eqthexpg[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*([2]+[3])\n"
nqthexpg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Liquid non-equilibrium thermal expansion %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", satrhof[VOLUME[i]], thexpf[VOLUME[i]], eqthexpf[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*([2]+[3])\n"
nqthexpf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Postprocessed time derivative of vapor non-equilibrium mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s %s\n", innqmflowgx[VOLUME[i]], innqmflowgy[VOLUME[i]], \
innqmflowgz[VOLUME[i]], outnqmflowgx[VOLUME[i]], \
outnqmflowgy[VOLUME[i]], outnqmflowgz[VOLUME[i]], \
nqthexpg[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[7]+[1]+[2]+[3]-[4]-[5]-[6]\n"
nfpostdnqmgdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Postprocessed time derivative of liquid non-equilibrium mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s %s %s\n", innqmflowfx[VOLUME[i]], innqmflowfy[VOLUME[i]], \
innqmflowfz[VOLUME[i]], outnqmflowfx[VOLUME[i]], \
outnqmflowfy[VOLUME[i]], outnqmflowfz[VOLUME[i]], \
nqthexpf[VOLUME[i]]

printf "27\n"
printf "*-----\n"
printf "[7]+[1]+[2]+[3]-[4]-[5]-[6]\n"
nfpostdnqmfdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Time derivative of vapor non-equilibrium mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s\n", nqmg[VOLUME[i]]
printf "43\n"
printf "*-----\n"
printf "16 1.0\n"
dnqmgdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Time derivative of liquid non-equilibrium mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s\n", nqmf[VOLUME[i]]
printf "43\n"
printf "*-----\n"
printf "16 1.0\n"
dnqmfdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Initial value of postlnp %s\n", VOLUME[i]
printf "%s 0 2\n", OUTCOUNT
printf "%s %s %s\n", OUTCOUNT, lnp[VOLUME[i]], PULSETIME
printf "14 0 0.0 0.0\n"
printf "*-----\n"
printf "3\n"
printf "0.5\n"
initlnp[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Postprocessed time derivative of ln p %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT

```

```

printf "%s %s\n", STEPTIME, nfpostdlnpdt[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
postdlnpdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Integral of postdlnpdt %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s\n", postdlnpdt[VOLUME[i]]
printf "1 0 0.0\n"
printf "*-----\n"
printf "0 0\n"
printf "1.0e+10 -1.0e+10 0.0\n"
intdlnpdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Postprocessed ln p %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", initlnp[VOLUME[i]], intdlnpdt[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
postlnp[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Initial value of postvf %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", OUTCOUNT, volf[VOLUME[i]], PULSETIME
printf "14 0 0.0 0.0\n"
printf "*-----\n"
printf "3\n"
printf "0.5\n"
initvf[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Postprocessed time derivative of liquid volume %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", STEPTIME, nfpostdvfdt[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
postdvfdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Integral of postdvfdt %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s\n", postdvfdt[VOLUME[i]]
printf "1 0 0.0\n"
printf "*-----\n"
printf "0 0\n"
printf "1.0e+10 -1.0e+10 0.0\n"
intdvfdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Postprocessed liquid volume %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", initvf[VOLUME[i]], intdvfdt[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
postvf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Initial value of vapor mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", OUTCOUNT, massg[VOLUME[i]], PULSETIME
printf "14 0 0.0 0.0\n"
printf "*-----\n"
printf "3\n"
printf "0.5\n"

```

```

initmg[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Postprocessed time derivative of vapor mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", STEPTIME, nfpostdmgdt[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
postdmgdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Integral of postdmgdt %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s\n", postdmgdt[VOLUME[i]]
printf "1 0 0.0\n"
printf "*-----\n"
printf "0 0\n"
printf "1.0e+10 -1.0e+10 0.0\n"
intdmgdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Postprocessed vapor mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", initmg[VOLUME[i]], intdmgdt[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
postmg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Initial value of liquid mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", OUTCOUNT, massf[VOLUME[i]], PULSETIME
printf "14 0 0.0 0.0\n"
printf "*-----\n"
printf "3\n"
printf "0.5\n"
initmf[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Postprocessed time derivative of liquid mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", STEPTIME, nfpostdmfdt[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
postdmfdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Integral of postdmfdt %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s\n", postdmfdt[VOLUME[i]]
printf "1 0 0.0\n"
printf "*-----\n"
printf "0 0\n"
printf "1.0e+10 -1.0e+10 0.0\n"
intdmfdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Postprocessed liquid mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", initmf[VOLUME[i]], intdmfdt[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
postmf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Initial value of non equilibrium vapor mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT

```

```

printf "%s %s %s\n", OUTCOUNT, nqmg[VOLUME[i]], PULSETIME
printf "14 0 0.0 0.0\n"
printf "*-----\n"
printf "3\n"
printf "0.5\n"
initnqmg[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Postprocessed time derivative of non equilibrium vapor mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", STEPTIME, nfpostdnqmgdt[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
postdnqmgdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Integral of postdnqmgdt %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s\n", postdnqmgdt[VOLUME[i]]
printf "1 0 0.0\n"
printf "*-----\n"
printf "0 0\n"
printf "1.0e+10 -1.0e+10 0.0\n"
intdnqmgdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Postprocessed non equilibrium vapor mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", initnqmg[VOLUME[i]], intdnqmgdt[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
postnqmg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Initial value of non equilibrium liquid mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", OUTCOUNT, nqmf[VOLUME[i]], PULSETIME
printf "14 0 0.0 0.0\n"
printf "*-----\n"
printf "3\n"
printf "0.5\n"
initnqmf[VOLUME[i]] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Post. time derivative of non equilibrium liquid mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", STEPTIME, nfpostdnqmfdt[VOLUME[i]]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
postdnqmfdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Integral of postdnqmfdt %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s\n", postdnqmfdt[VOLUME[i]]
printf "1 0 0.0\n"
printf "*-----\n"
printf "0 0\n"
printf "1.0e+10 -1.0e+10 0.0\n"
intdnqmfdt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Postprocessed non equilibrium liquid mass %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s\n", initnqmf[VOLUME[i]], intdnqmfdt[VOLUME[i]]
printf "27\n"
printf "*-----\n"

```

```

printf "[1]+[2]\n"
postnqmf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Term with gamma bulk in the non-eq vapor mass equation %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s %s %s\n", hstarg[VOLUME[i]]+1, vvol[VOLUME[i]], \
    gammai[VOLUME[i]], pig[VOLUME[i]], \
    hstarg[VOLUME[i]], hg[VOLUME[i]], \
    rhog[VOLUME[i]], satrhog[VOLUME[i]]

printf "27\n"
printf "*****\n"
printf "[1]*[2]*[3]*[8]*([4]*([5]-[6])+(1.0/[7])-(1.0/[8]))\n"
nqgammaigt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Term with gamma bulk in the non-eq liquid mass equation %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s %s %s\n", hstarf[VOLUME[i]]+1, vvol[VOLUME[i]], \
    gammai[VOLUME[i]], pif[VOLUME[i]], \
    hstarf[VOLUME[i]], hf[VOLUME[i]], \
    rhof[VOLUME[i]], satrhof[VOLUME[i]]

printf "27\n"
printf "*****\n"
printf "([1]-1.0)*[2]*[3]*[8]*([4]*([5]-[6])+(1.0/[7])-(1.0/[8]))\n"
nqgammaigt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Term with gamma wall in the non-eq vapor mass equation %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s %s %s\n", hprimeg[VOLUME[i]]+1, vvol[VOLUME[i]], \
    gammaw[VOLUME[i]], pig[VOLUME[i]], \
    hprimeg[VOLUME[i]], hg[VOLUME[i]], \
    rhog[VOLUME[i]], satrhog[VOLUME[i]]

printf "27\n"
printf "*****\n"
printf "[1]*[2]*[3]*[8]*([4]*([5]-[6])+(1.0/[7])-(1.0/[8]))\n"
nqgammaigt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Term with gamma wall in the non-eq liquid mass equation %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s %s %s\n", hprimef[VOLUME[i]]+1, vvol[VOLUME[i]], \
    gammaw[VOLUME[i]], pif[VOLUME[i]], \
    hprimef[VOLUME[i]], hf[VOLUME[i]], \
    rhof[VOLUME[i]], satrhof[VOLUME[i]]

printf "27\n"
printf "*****\n"
printf "([1]-1.0)*[2]*[3]*[8]*([4]*([5]-[6])+(1.0/[7])-(1.0/[8]))\n"
nqgammaigt[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Approximated interfacial heat in the bulk (vapor) %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s\n", hstarg[VOLUME[i]]+1, satrhog[VOLUME[i]], \
    rhog[VOLUME[i]], hig[VOLUME[i]], \
    betagg[VOLUME[i]]

printf "27\n"
printf "*****\n"
printf "[1]*([3]-[2])*[4]/([3]*[5])\n"
appqibg[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "Approximated interfacial heat in the bulk (liquid) %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s\n", hstarf[VOLUME[i]]+1, satrhof[VOLUME[i]], \
    rhof[VOLUME[i]], hif[VOLUME[i]], \
    betaff[VOLUME[i]]

printf "27\n"
printf "*****\n"

```

```

printf "([1]-1.0)*([2]-[3])*[4]/([3]*[5])\n"
appqibf[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "OUTPUT FILE VOLUME %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s %s %s %s %s %s %s ", \
    lnp[VOLUME[i]], postlnp[VOLUME[i]], \
    volf[VOLUME[i]], postvf[VOLUME[i]], \
    massg[VOLUME[i]], postmg[VOLUME[i]], \
    massf[VOLUME[i]], postmf[VOLUME[i]], \
    nqmg[VOLUME[i]], postnqmg[VOLUME[i]], \
    nqmf[VOLUME[i]], postnqmf[VOLUME[i]]
printf "\n"
printf "38\n"
printf "*-----\n"
printf "%s_%s.dat\n", FILE, VOLUME[i]
printf "(%s(E17.8))\n", 12+1
printf "(%s(A16,x))\n", 12+1
printf "%s 0 %6d\n", FREQ, ((STARTTIME-INITTIME)/MAXDT)+2
printf "lnp-%s\n", VOLUME[i]
printf "postlnp-%s\n", VOLUME[i]
printf "volf-%s\n", VOLUME[i]
printf "postvf-%s\n", VOLUME[i]
printf "massg-%s\n", VOLUME[i]
printf "postmg-%s\n", VOLUME[i]
printf "massf-%s\n", VOLUME[i]
printf "postmf-%s\n", VOLUME[i]
printf "nqmg-%s\n", VOLUME[i]
printf "postnqmg-%s\n", VOLUME[i]
printf "nqmf-%s\n", VOLUME[i]
printf "postnqmf-%s\n", VOLUME[i]
outfile[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
# printf "OUTPUT FILE: DERIVATIVES. VOLUME %s\n", VOLUME[i]
# printf "%s\n", OUTCOUNT
# printf "%s %s %s %s %s %s %s %s %s %s %s %s ", \
#     dlnpdt[VOLUME[i]], postdlnpdt[VOLUME[i]], \
#     dvfdt[VOLUME[i]], postdvfdt[VOLUME[i]], \
#     dmgt[VOLUME[i]], postdmgt[VOLUME[i]], \
#     dmfdt[VOLUME[i]], postdmfdt[VOLUME[i]], \
#     dnqmgdt[VOLUME[i]], postdnqmgdt[VOLUME[i]], \
#     dnqmfdt[VOLUME[i]], postdnqmfdt[VOLUME[i]]
# printf "\n"
# printf "38\n"
# printf "*-----\n"
# printf "%s_%s_der.dat\n", FILE, VOLUME[i]
# printf "(%s(E17.8))\n", 12+1
# printf "(%s(A16,x))\n", 12+1
# printf "%s 0 %6d\n", FREQ, ((STARTTIME-INITTIME)/MAXDT)+2
# printf "dlnpdt-%s\n", VOLUME[i]
# printf "postdlnpdt-%s\n", VOLUME[i]
# printf "dvfdt-%s\n", VOLUME[i]
# printf "postdvfdt-%s\n", VOLUME[i]
# printf "dmgt-%s\n", VOLUME[i]
# printf "postdmgt-%s\n", VOLUME[i]
# printf "dmfdt-%s\n", VOLUME[i]
# printf "postdmfdt-%s\n", VOLUME[i]
# printf "dnqmgdt-%s\n", VOLUME[i]
# printf "postdnqmgdt-%s\n", VOLUME[i]
# printf "dnqmfdt-%s\n", VOLUME[i]
# printf "postdnqmfdt-%s\n", VOLUME[i]
# deroutfile[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "OUTPUT FILE: TERMS1. VOLUME %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT

```

```

printf "%s %s %s %s %s %s %s %s %s %s\n", \
    jtermg[VOLUME[i]], jtermf[VOLUME[i]], \
    thexpg[VOLUME[i]], thexpf[VOLUME[i]], \
    swellg[VOLUME[i]], swellf[VOLUME[i]], \
    acg[VOLUME[i]], acf[VOLUME[i]], \
    nodecompg[VOLUME[i]], nodecompf[VOLUME[i]]
printf "\n"
printf "38\n"
printf "*-----\n"
printf "%s_%s_terms1.dat\n", FILE, VOLUME[i]
printf "(%s(E17.8))\n", 10+1
printf "(%s(A16,x))\n", 10+1
printf "%s 0 %6d\n", FREQ, ((STARTTIME-INITTIME)/MAXDT)+2
printf "jtermg-%s\n", VOLUME[i]
printf "jtermf-%s\n", VOLUME[i]
printf "thexpg-%s\n", VOLUME[i]
printf "thexpf-%s\n", VOLUME[i]
printf "swellg-%s\n", VOLUME[i]
printf "swellf-%s\n", VOLUME[i]
printf "acg-%s\n", VOLUME[i]
printf "acf-%s\n", VOLUME[i]
printf "nodecompg-%s\n", VOLUME[i]
printf "nodecompf-%s\n", VOLUME[i]
terms1outfile[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "OUTPUT FILE: TERMS2. VOLUME %s\n", VOLUME[i]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s %s %s %s %s\n", \
    qwtermg[VOLUME[i]], qwtermf[VOLUME[i]], \
    qitermg[VOLUME[i]], qitermf[VOLUME[i]], \
    gammtg[VOLUME[i]], gammtf[VOLUME[i]], \
    eqthexpg[VOLUME[i]], eqthexpf[VOLUME[i]], \
    nqthexpg[VOLUME[i]], nqthexpf[VOLUME[i]]
printf "%s %s %s %s %s %s %s %s\n", \
    nqgammaigt[VOLUME[i]], nqgammaift[VOLUME[i]], \
    nqgammaawt[VOLUME[i]], nqgammaawtf[VOLUME[i]], \
    qibg[VOLUME[i]], qibf[VOLUME[i]], \
    appqibg[VOLUME[i]], appqibf[VOLUME[i]]
printf "\n"
printf "38\n"
printf "*-----\n"
printf "%s_%s_terms2.dat\n", FILE, VOLUME[i]
printf "(%s(E17.8))\n", 18+1
printf "(%s(A16,x))\n", 18+1
printf "%s 0 %6d\n", FREQ, ((STARTTIME-INITTIME)/MAXDT)+2
printf "qwtermg-%s\n", VOLUME[i]
printf "qwtermf-%s\n", VOLUME[i]
printf "qitermg-%s\n", VOLUME[i]
printf "qitermf-%s\n", VOLUME[i]
printf "gammtg-%s\n", VOLUME[i]
printf "gammtf-%s\n", VOLUME[i]
printf "eqthexpg-%s\n", VOLUME[i]
printf "eqthexpf-%s\n", VOLUME[i]
printf "nqthexpg-%s\n", VOLUME[i]
printf "nqthexpf-%s\n", VOLUME[i]
printf "nqgammaigt-%s\n", VOLUME[i]
printf "nqgammaift-%s\n", VOLUME[i]
printf "nqgammaawt-%s\n", VOLUME[i]
printf "nqgammaawtf-%s\n", VOLUME[i]
printf "qibg-%s\n", VOLUME[i]
printf "qibf-%s\n", VOLUME[i]
printf "appqibg-%s\n", VOLUME[i]
printf "appqibf-%s\n", VOLUME[i]
terms2outfile[VOLUME[i]] = OUTCOUNT++
printf "***** \n"

```



```

# printf "OUTPUT FILE: TERMS3. VOLUME %s\n", VOLUME[i]
# printf "%s\n", OUTCOUNT
# printf "%s %s %s %s %s %s %s %s %s %s", \
#   deltap[VOLUME[i]], wfricg[VOLUME[i]], \
#   wfricf[VOLUME[i]], buoyg[VOLUME[i]], buoyf[VOLUME[i]], \
#   mommixconv[VOLUME[i]], nonhmconv[VOLUME[i]], \
#   dermmommix[VOLUME[i]], avol[VOLUME[i]], \
#   deltapgx[VOLUME[i]], \
#   deltapfx[VOLUME[i]]
# printf "\n"
# printf "38\n"
# printf "-----\n"
# printf "%s_%s_terms3.dat\n", FILE, VOLUME[i]
# printf "(%s(E17.8))\n", 11+1
# printf "(%s(A16,x))\n", 11+1
# printf "%s 0 %6d\n", FREQ, ((STARTTIME-INITTIME)/MAXDT)+2
# printf "deltap-%s\n", VOLUME[i]
# printf "wfricg-%s\n", VOLUME[i]
# printf "wfricf-%s\n", VOLUME[i]
# printf "buoyg-%s\n", VOLUME[i]
# printf "buoyf-%s\n", VOLUME[i]
# printf "mommixconv-%s\n", VOLUME[i]
# printf "nonhmconv-%s\n", VOLUME[i]
# printf "dermmommix-%s\n", VOLUME[i]
# printf "avol-%s\n", VOLUME[i]
# printf "deltapgx-%s\n", VOLUME[i]
# printf "deltapfx-%s\n", VOLUME[i]
# terms3outfile[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
# printf "OUTPUT FILE: TERMS4. VOLUME %s\n", VOLUME[i]
# printf "%s\n", OUTCOUNT
# printf "%s %s %s %s %s %s %s %s %s %s", \
#   deltap[VOLUME[i]], deltapgx[VOLUME[i]], deltapfx[VOLUME[i]], \
#   inpjgx[VOLUME[i]], outpjgx[VOLUME[i]], \
#   inpjfx[VOLUME[i]], outpjfx[VOLUME[i]], \
#   inavoidpjgx[VOLUME[i]], outavoidpjgx[VOLUME[i]], \
#   inavoidpjfx[VOLUME[i]], outavoidpjfx[VOLUME[i]]
#
# printf "\n"
# printf "38\n"
# printf "-----\n"
# printf "%s_%s_terms4.dat\n", FILE, VOLUME[i]
# printf "(%s(E17.8))\n", 11+1
# printf "(%s(A16,x))\n", 11+1
# printf "%s 0 %6d\n", FREQ, ((STARTTIME-INITTIME)/MAXDT)+2
# printf "deltap-%s\n", VOLUME[i]
# printf "deltapgx-%s\n", VOLUME[i]
# printf "deltapfx-%s\n", VOLUME[i]
# printf "inpjgx-%s\n", VOLUME[i]
# printf "outpjgx-%s\n", VOLUME[i]
# printf "inpjfx-%s\n", VOLUME[i]
# printf "outpjfx-%s\n", VOLUME[i]
# printf "inavoidpjgx-%s\n", VOLUME[i]
# printf "outavoidpjgx-%s\n", VOLUME[i]
# printf "inavoidpjfx-%s\n", VOLUME[i]
# printf "outavoidpjfx-%s\n", VOLUME[i]
# terms3outfile[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
# printf "OUTPUT FILE: NQ. VOLUME %s\n", VOLUME[i]
# printf "%s\n", OUTCOUNT
# printf "%s %s %s %s %s %s %s %s", \
#   insatmflowfx[VOLUME[i]], outsatmflowfx[VOLUME[i]], \
#   invflowfx[VOLUME[i]], outvflowfx[VOLUME[i]], \
#   insatrhofx[VOLUME[i]], outsatrhofx[VOLUME[i]], \
#   inpjfx[VOLUME[i]], outpjfx[VOLUME[i]]

```

```

#   printf "\n"
#   printf "38\n"
#   printf "-----\n"
#   printf "%s_%s_nq.dat\n", FILE, VOLUME[i]
#   printf "(%s(E17.8))\n", 8+1
#   printf "(%s(A16,x))\n", 8+1
#   printf "%s 0 %6d\n", FREQ, ((STARTTIME-INITTIME)/MAXDT)+2
#   printf "insatmflowfx-%s\n", VOLUME[i]
#   printf "outsatmflowfx-%s\n", VOLUME[i]
#   printf "invflowfx-%s\n", VOLUME[i]
#   printf "outvflowfx-%s\n", VOLUME[i]
#   printf "insatrhofx-%s\n", VOLUME[i]
#   printf "outsatrhofx-%s\n", VOLUME[i]
#   printf "inpjfx-%s\n", VOLUME[i]
#   printf "outpjfx-%s\n", VOLUME[i]
#   nqoutfile[VOLUME[i]] = OUTCOUNT++
#   printf "***** \n"
# End of volume loop
}

for(sys=1;sys<=NSYSTEMS;sys++) {
#   printf "TOTVOLS = %s\n", TOTVOLS
#   printf "SYSNAME = %s\n", SYSNAME[sys]
  if(1 == sys){
    initloop = 0
  }
  else{
    initloop += NVOLS[sys-1]
  }
  for(i=1;i<=TOTVOLS;i++){
    volflag[i] = ""
  }
  for(i=1;i<=TOTVOLS;i++){
    for(k=1;k<=NVOLS[sys];k++){
      if(VOLINSYS[k+initloop] == VOLUME[i]){
        volflag[i] = "+"
      }
    }
  }
#   printf "volflag[%s] = %s\n", i, volflag[i]
}
printf "***** \n"
printf "***** Postprocess of TH system %s *****\n", \
      SYSNAME[sys]
printf "***** \n"
printf "In - out vapor pressure in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
  if(volflag[i] != "") {
    printf "\\n%s ", deltapgx[VOLUME[i]]
  }
}
printf "\n"
printf "27\n"
printf "-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
  if(volflag[i] != "") {
    printf "\\n%s[%s]", volflag[i], auxcount++
  }
}
printf "\n"
sysdeltapgx[sys] = OUTCOUNT++

```

```

printf "***** \n"
printf "In - out liquid pressure in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", deltapfx[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*****\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysdeltapfx[sys] = OUTCOUNT++
printf "***** \n"
printf "ln p * vapor compressibility in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s %s ", lnp[VOLUME[i]], nodecompf[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*****\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]*[%s]", volflag[i], auxcount++, auxcount++
    }
}
printf "\n"
syslnpcompf[sys] = OUTCOUNT++
printf "***** \n"
printf "ln p * liquid compressibility in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s %s ", lnp[VOLUME[i]], nodecompf[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*****\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]*[%s]", volflag[i], auxcount++, auxcount++
    }
}
printf "\n"
syslnpcompf[sys] = OUTCOUNT++
printf "***** \n"
printf "Vapor compressibility in system %s\n", SYSNAME[sys]

```

```

printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", nodecomp[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
syscomp[sys] = OUTCOUNT++
printf "***** \n"
printf "Liquid compressibility in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", nodecomp[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
syscomp[sys] = OUTCOUNT++
printf "***** \n"
printf "Vapor volume in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", vol[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysvol[sys] = OUTCOUNT++
printf "***** \n"
printf "Liquid volume in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO

```

```

for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", volf[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysvolff[sys] = OUTCOUNT++
printf "***** \n"
printf "Time derivative of liquid volume in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s\n", sysvolff[sys]
printf "43\n"
printf "*-----\n"
printf "16 1.0\n"
sysdvfdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Vapor mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", massg[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysmassg[sys] = OUTCOUNT++
printf "***** \n"
printf "Time derivative of vapor mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s\n", sysmassg[sys]
printf "43\n"
printf "*-----\n"
printf "16 1.0\n"
sysdmgt[sys] = OUTCOUNT++
printf "***** \n"
printf "Liquid mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", massf[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"

```

```

printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysmassf[sys] = OUTCOUNT++
printf "***** \n"
printf "Time derivative of liquid mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s\n", sysmassf[sys]
printf "43\n"
printf "*-----\n"
printf "16 1.0\n"
sysdmf[sys] = OUTCOUNT++
printf "***** \n"
printf "Vapor non-equilibrium mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", nqmg[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysnqmg[sys] = OUTCOUNT++
printf "***** \n"
printf "Time derivative of non-eq. vapor mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s\n", sysnqmg[sys]
printf "43\n"
printf "*-----\n"
printf "16 1.0\n"
sysdnqmgdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Liquid non-equilibrium mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", nqmf[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}

```

```

}
printf "\n"
sysnqmf[sys] = OUTCOUNT++
printf "***** \n"
printf "Time derivative of non-eq. liquid mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s\n", sysnqmf[sys]
printf "43\n"
printf "*-----\n"
printf "16 1.0\n"
sysdnqmf[sys] = OUTCOUNT++
printf "***** \n"
printf "Vapor inlet mass flow in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", inmflowg[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysinmflowg[sys] = OUTCOUNT++
printf "***** \n"
printf "Vapor outlet mass flow in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", outmflowg[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysoutmflowg[sys] = OUTCOUNT++
printf "***** \n"
printf "Liquid inlet mass flow in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", inmflowf[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"

```

```

printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysinmflowf[sys] = OUTCOUNT++
printf "***** \n"
printf "Liquid outlet mass flow in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", outmflowf[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysoutmflowf[sys] = OUTCOUNT++
printf "***** \n"
printf "Vapor volumetric flow in - volumetric flow out in system %s\n", \
        SYSNAME[sys]

printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", jtermg[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysjtermg[sys] = OUTCOUNT++
printf "***** \n"
printf "Liquid volumetric flow in - volumetric flow out in system %s\n", \
        SYSNAME[sys]

printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", jtermf[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"

```



```

printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysjtermf[sys] = OUTCOUNT++
printf "***** \n"
printf "Vapor thermal expansion in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", thexpg[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
systhexpg[sys] = OUTCOUNT++
printf "***** \n"
printf "Liquid thermal expansion in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", thexpf[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
systhexpf[sys] = OUTCOUNT++
printf "***** \n"
printf "Vapor swelling in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", swellg[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2

```

```

for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysswellg[sys] = OUTCOUNT++
printf "***** \n"
printf "Liliquid swelling in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", swellf[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysswellf[sys] = OUTCOUNT++
printf "***** \n"
printf "Main term of vapor acoustic term in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", acg[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysmainacg[sys] = OUTCOUNT++
printf "***** \n"
printf "Main term of liquid acoustic term in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", acf[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {

```

```

        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysmainacf[sys] = OUTCOUNT++
printf "***** \n"
printf "First term of vapor acoustic term in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", ac1g[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*****\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysac1g[sys] = OUTCOUNT++
printf "***** \n"
printf "First term of liquid acoustic term in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", ac1f[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*****\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysac1f[sys] = OUTCOUNT++
printf "***** \n"
printf "Second term of vapor acoustic term in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", ac2g[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*****\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}

```

```

}
printf "\n"
sysac2g[sys] = OUTCOUNT++
printf "***** \n"
printf "Second term of liquid acoustic term in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", ac2f[VOLUME[i]]
    }
}
}
printf "\n"
printf "27\n"
printf "-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
}
printf "\n"
sysac2f[sys] = OUTCOUNT++
printf "***** \n"
printf "Vapor non-equilibrium thermal expansion in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", nqthexpg[VOLUME[i]]
    }
}
}
printf "\n"
printf "27\n"
printf "-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
}
printf "\n"
sysnqthexpg[sys] = OUTCOUNT++
printf "***** \n"
printf "Liquid non-equilibrium thermal expansion in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", nqthexpf[VOLUME[i]]
    }
}
}
printf "\n"
printf "27\n"
printf "-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
}
printf "\n"

```

```

sysnqthexpf[sys] = OUTCOUNT++
printf "***** \n"
printf "Vapor generation rate in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s %s ", vvol[VOLUME[i]], gammag[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]*[%s]", volflag[i], auxcount++, auxcount++
    }
}
printf "\n"
sysgenrg[sys] = OUTCOUNT++
printf "***** \n"
printf "Liquid generation rate in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s %s ", vvol[VOLUME[i]], gammag[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]*[%s]", volflag[i], auxcount++, auxcount++
    }
}
printf "\n"
sysgenrf[sys] = OUTCOUNT++
printf "***** \n"
printf "Inlet vapor non-equilibrium mass flow in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", innqmflowg[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysinnqmflowg[sys] = OUTCOUNT++
printf "***** \n"

```

```

printf "Outlet vapor non-equilibrium mass flow in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", outnqmflowg[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysoutnqmflowg[sys] = OUTCOUNT++
printf "***** \n"
printf "Inlet liquid non-equilibrium mass flow in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", innqmflowf[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysinnqmflowf[sys] = OUTCOUNT++
printf "***** \n"
printf "Outlet liquid non-equilibrium mass flow in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", outnqmflowf[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysoutnqmflowf[sys] = OUTCOUNT++
printf "***** \n"
printf "Inlet vapor non-equilibrium mass flow in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT

```

```

printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", innqmflowg[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysinnqmflowg[sys] = OUTCOUNT++
printf "***** \n"
printf "Outlet vapor non-equilibrium mass flow in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", outnqmflowg[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysoutnqmflowg[sys] = OUTCOUNT++
printf "***** \n"
printf "Non zero compg in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, syscompg[sys], syscompg[sys]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzsyscompg[sys] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Non zero compf in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", CONST1, syscompf[sys], syscompf[sys]
printf "14\n"
printf "*-----\n"
printf "3\n"
printf "1.0e-10\n"
nzsyscompf[sys] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "ln p in vapor phase, system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", nzsyscompg[sys]+1, syslnpcompg[sys], \
        nzsyscompf[sys]
printf "27\n"

```

```

printf "*-----\n"
printf "[1]*[2]/[3]\n"
syslnpg[sys] = OUTCOUNT++
printf "***** \n"
printf "ln p in liquid phase, system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", nzsyscompf[sys]+1, syslnpcompf[sys], \
      nzsyscompf[sys]

printf "27\n"
printf "*-----\n"
printf "[1]*[2]/[3]\n"
syslnpf[sys] = OUTCOUNT++
printf "***** \n"
printf "Vapor acoustic term in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", sysmainacg[sys], sysac1g[sys], syslnpg[sys], \
      sysac2g[sys]

printf "27\n"
printf "*-----\n"
printf "[1]+[2]-([3]*[4])\n"
sysacg[sys] = OUTCOUNT++
printf "***** \n"
printf "Liquid acoustic term in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", sysmainacf[sys], sysac1f[sys], syslnpf[sys], \
      sysac2f[sys]

printf "27\n"
printf "*-----\n"
printf "[1]+[2]-([3]*[4])\n"
sysacf[sys] = OUTCOUNT++
printf "***** \n"
printf "ln p in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", syslnpcompf[sys], syslnpcompf[sys], \
      syscompf[sys], syscompf[sys]

printf "27\n"
printf "*-----\n"
printf "([1]+[2])/([3]+[4])\n"
syslnp[sys] = OUTCOUNT++
printf "***** \n"
printf "Time derivative of ln p in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s\n", syslnp[sys]
printf "43\n"
printf "*-----\n"
printf "16 1.0\n"
sysdlpdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Pressure in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s\n", syslnp[sys]
printf "27\n"
printf "*-----\n"
printf "EXP([1])\n"
sysp[sys] = OUTCOUNT++
printf "***** \n"
printf "Vapor jterm+thexp+swell+ac in %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", sysjtermg[sys], systhexp[sys], sysswellg[sys], \
      sysacg[sys]

printf "27\n"
printf "*-----\n"
printf "[1]+[2]+[3]+[4]\n"
sysjthswacg[sys] = OUTCOUNT++
printf "***** \n"

```



```

printf "Liquid jterm+thexp+swell+ac in %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s\n", sysjtermf[sys], systhexpf[sys], sysswellf[sys], \
      sysacf[sys]

printf "27\n"
printf "*-----\n"
printf "[1]+[2]+[3]+[4]\n"
sysjthswacf[sys] = OUTCOUNT++
printf "***** \n"
printf "Non-filtered postprocessed time derivative of ln p in system %s\n", \
      SYSNAME[sys]

printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s\n", sysjthswacg[sys], sysjthswacf[sys], \
      nzsyscompf[sys]+1, nzsyscompf[sys], \
      nzsyscompf[sys]+1, nzsyscompf[sys]

printf "27\n"
printf "*-----\n"
printf "([1]+[2])/([3]*[4])+([5]*[6])\n"
nfsyspostdlnpdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Initial value of ln p in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", OUTCOUNT, syslnp[sys], PULSETIME
printf "14 0 0.0 0.0\n"
printf "*-----\n"
printf "3\n"
printf "0.5\n"
sysinitlnp[sys] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Postprocessed time derivative of ln p in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s\n", STEPTIME, nfsyspostdlnpdt[sys]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
syspostdlnpdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Integral of postdlnpdt in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s\n", syspostdlnpdt[sys]
printf "1 0 0.0\n"
printf "*-----\n"
printf "0 0\n"
printf "1.0e+10 -1.0e+10 0.0\n"
sysintdlnpdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Postprocessed ln p in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s\n", sysinitlnp[sys], sysintdlnpdt[sys]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
syspostlnp[sys] = OUTCOUNT++
printf "***** \n"
printf "Postprocessed pressure in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s\n", syspostlnp[sys]
printf "27\n"
printf "*-----\n"
printf "EXP([1])\n"
syspostp[sys] = OUTCOUNT++
printf "***** \n"
printf "Non-filtered postprocessed time derivative of liquid volume "
printf "in system %s\n", SYSNAME[sys]

```

```

printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s %s\n", sysjtermf[sys], systhexpf[sys], \
                                     sysswellf[sys], sysacf[sys], \
                                     nzsyscompf[sys]+1, nzsyscompf[sys], \
                                     syspostdlnpdt[sys]

printf "27\n"
printf "*-----\n"
printf "[1]+[2]+[3]+[4]-([5]*[6]*[7])\n"
nfsyspostdvfdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Initial value of liquid volume in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", OUTCOUNT, sysvolf[sys], PULSETIME
printf "14 0 0.0 0.0\n"
printf "*-----\n"
printf "3\n"
printf "0.5\n"
sysinitvolf[sys] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Postprocessed time derivative of liquid volume in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s\n", STEPTIME, nfsyspostdvfdt[sys]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
syspostdvfdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Integral of postdvfdt in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s\n", syspostdvfdt[sys]
printf "1 0 0.0\n"
printf "*-----\n"
printf "0 0\n"
printf "1.0e+10 -1.0e+10 0.0\n"
sysintdvfdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Postprocessed liquid volume in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s\n", sysinitvolf[sys], sysintdvfdt[sys]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
syspostvolf[sys] = OUTCOUNT++
printf "***** \n"
printf "Non-filtered postprocessed time derivative of vapor mass in system %s\n", \
                                     SYSNAME[sys]

printf "%s\n", OUTCOUNT
printf "%s %s %s\n", sysinmflowg[sys], sysoutmflowg[sys], sysgenrg[sys]
printf "27\n"
printf "*-----\n"
printf "[1]+[3]-[2]\n"
nfsyspostdmgdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Initial value of vapor mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", OUTCOUNT, sysmassg[sys], PULSETIME
printf "14 0 0.0 0.0\n"
printf "*-----\n"
printf "3\n"
printf "0.5\n"
sysinitmg[sys] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Postprocessed time derivative of vapor mass in system %s\n", SYSNAME[sys]

```

```

printf "%s\n", OUTCOUNT
printf "%s %s\n", STEPTIME, nfsyspostdmgdt[sys]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
syspostdmgdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Integral of postdmgdt %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s\n", syspostdmgdt[sys]
printf "1 0 0.0\n"
printf "*-----\n"
printf "0 0\n"
printf "1.0e+10 -1.0e+10 0.0\n"
sysintdmgdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Postprocessed vapor mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s\n", sysinitmg[sys], sysintdmgdt[sys]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
syspostmg[sys] = OUTCOUNT++
printf "***** \n"
printf "Non-filtered postprocessed time derivative of liquid mass in system %s\n", \
SYSNAME[sys]

printf "%s\n", OUTCOUNT
printf "%s %s %s\n", sysinmflow[sys], sysoutmflow[sys], sysgenrf[sys]
printf "27\n"
printf "*-----\n"
printf "[1]+[3]-[2]\n"
nfsyspostdmfdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Initial value of liquid mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", OUTCOUNT, sysmassf[sys], PULSETIME
printf "14 0 0.0 0.0\n"
printf "*-----\n"
printf "3\n"
printf "0.5\n"
sysinitmf[sys] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Postprocessed time derivative of liquid mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s\n", STEPTIME, nfsyspostdmfdt[sys]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
syspostdmfdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Integral of postdmfdt %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s\n", syspostdmfdt[sys]
printf "1 0 0.0\n"
printf "*-----\n"
printf "0 0\n"
printf "1.0e+10 -1.0e+10 0.0\n"
sysintdmfdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Postprocessed liquid mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s\n", sysinitmf[sys], sysintdmfdt[sys]
printf "27\n"
printf "*-----\n"

```

```

printf "[1]+[2]\n"
syspostmf[sys] = OUTCOUNT++
printf "***** \n"
printf "Non-filtered post. time der. of vapor non-eq. mass in system %s\n", \
      SYSNAME[sys]

printf "%s\n", OUTCOUNT
printf "%s %s %s\n", sysnqthexp[sys], sysinnqmflow[sys], sysoutnmflow[sys]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]-[3]\n"
nfsyspostdnqmgdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Initial value of non-eq. vapor mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", OUTCOUNT, sysnqmg[sys], PULSETIME
printf "14 0 0.0 0.0\n"
printf "*-----\n"
printf "3\n"
printf "0.5\n"
sysinitnqmg[sys] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
printf "Post. time derivative of non-eq. vapor mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s\n", STEPTIME, nfsyspostdnqmgdt[sys]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
syspostdnqmgdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Integral of postdnqmgdt %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s\n", syspostdnqmgdt[sys]
printf "1 0 0.0\n"
printf "*-----\n"
printf "0 0\n"
printf "1.0e+10 -1.0e+10 0.0\n"
sysintdnqmgdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Postprocessed vapor non-eq. mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s\n", sysinitnqmg[sys], sysintdnqmgdt[sys]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
syspostnqmg[sys] = OUTCOUNT++
printf "***** \n"
printf "Non-filtered post. time der. of liquid non-eq. mass in system %s\n", \
      SYSNAME[sys]

printf "%s\n", OUTCOUNT
printf "%s %s %s\n", sysnqthexp[sys], sysinnqmflow[sys], sysoutnmflow[sys]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]-[3]\n"
nfsyspostdnqmf[sys] = OUTCOUNT++
printf "***** \n"
printf "Initial value of non-eq. liquid mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s %s\n", OUTCOUNT, sysnqmf[sys], PULSETIME
printf "14 0 0.0 0.0\n"
printf "*-----\n"
printf "3\n"
printf "0.5\n"
sysinitnqmf[sys] = OUTCOUNT++
OUTCOUNT++

```

```

printf "***** \n"
printf "Post. time derivative of non-eq. liquid mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s\n", STEPTIME, nfsyspostdnqmfdt[sys]
printf "27\n"
printf "*-----\n"
printf "[1]*[2]\n"
syspostdnqmfdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Integral of postdnqmfdt %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s\n", syspostdnqmfdt[sys]
printf "1 0 0.0\n"
printf "*-----\n"
printf "0 0\n"
printf "1.0e+10 -1.0e+10 0.0\n"
sysintdnqmfdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Postprocessed liquid non-eq. mass in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s\n", sysinitnqmfdt[sys], sysintdnqmfdt[sys]
printf "27\n"
printf "*-----\n"
printf "[1]+[2]\n"
syspostnqmfdt[sys] = OUTCOUNT++
printf "***** \n"
printf "Vapor mixture momentum in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", mommix[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysmommix[sys] = OUTCOUNT++
printf "***** \n"
printf "Time derivative of mixture momentum in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s\n", sysmommix[sys]
printf "43\n"
printf "*-----\n"
printf "16 1.0\n"
dersysmommix[sys] = OUTCOUNT++
printf "***** \n"
printf "Vapor wall friction in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", wfricg[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"

```

```

printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
syswfricg[sys] = OUTCOUNT++
printf "***** \n"
printf "Liquid wall friction in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", wfricf[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
syswfricf[sys] = OUTCOUNT++
printf "***** \n"
printf "Vapor buoyancy in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", buoyg[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysbuoyg[sys] = OUTCOUNT++
printf "***** \n"
printf "Liquid buoyancy in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", buoyf[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"

```

```

auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysbuoyf[sys] = OUTCOUNT++
printf "***** \n"
printf "Mixture momentum convective term in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", mommixconv[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysmommixconv[sys] = OUTCOUNT++
printf "***** \n"
printf "Non homogeneous convective term in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", nonhmconv[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s[%s]", volflag[i], auxcount++
    }
}
printf "\n"
sysnonhmconv[sys] = OUTCOUNT++
printf "***** \n"
printf "Time derivative of mixture momentum in system %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s ", CONSTO
for(i=1;i<=TOTVOLS;i++) {
    if(volflag[i] != "") {
        printf "\\n%s ", dermommix[VOLUME[i]]
    }
}
printf "\n"
printf "27\n"
printf "*-----\n"
printf "[1]"
auxcount = 2
for(i=1;i<=TOTVOLS;i++) {

```



```

        sysdlnpdt[sys],    syspostdlnpdt[sys], \
        sysdvfdt[sys],    syspostdvfdt[sys], \
        sysdmgdt[sys],    syspostdmgdt[sys], \
        sysdmfdt[sys],    syspostdmfdt[sys], \
        sysdnqmgdt[sys],  syspostdnqmgdt[sys], \
        sysdnqmfdt[sys],  syspostdnqmfdt[sys]
printf "\n"
printf "38\n"
printf "*-----\n"
printf "%s_%s_der.dat\n", FILE, SYSNAME[sys]
printf "(%s(E17.8))\n", 12+1
printf "(%s(A16,x))\n", 12+1
printf "%s 0 %6d\n", FREQ, ((STARTTIME-INITTIME)/MAXDT)+2
printf "sysdlnpdt-%s\n", SYSNAME[sys]
printf "syspostdlnpdt-%s\n", SYSNAME[sys]
printf "sysdvfdt-%s\n", SYSNAME[sys]
printf "syspostdvfdt-%s\n", SYSNAME[sys]
printf "sysdmgdt-%s\n", SYSNAME[sys]
printf "syspostdmgdt-%s\n", SYSNAME[sys]
printf "sysdmfdt-%s\n", SYSNAME[sys]
printf "syspostdmfdt-%s\n", SYSNAME[sys]
printf "sysdnqmgdt-%s\n", SYSNAME[sys]
printf "syspostdnqmgdt-%s\n", SYSNAME[sys]
printf "sysdnqmfdt-%s\n", SYSNAME[sys]
printf "syspostdnqmfdt-%s\n", SYSNAME[sys]
sysderoutfile[VOLUME[i]] = OUTCOUNT++
printf "***** \n"
printf "OUTPUT FILE: TERMS. VOLUME %s\n", SYSNAME[sys]
printf "%s\n", OUTCOUNT
printf "%s %s %s %s %s %s %s %s %s %s %s \\n", \
        sysjtermg[sys], sysjtermf[sys], \
        systhexpg[sys], systhexpf[sys], \
        sysswellg[sys], sysswellf[sys], \
        sysacg[sys], sysacf[sys], \
        syscompg[sys], syscompf[sys]
printf "%s %s %s \\n", \
        sysnqthexpg[sys], sysnqthexpf[sys], \
        sysdeltap[sys]
printf "%s %s %s %s %s %s %s %s %s", \
        syswfricg[sys], syswfricf[sys], \
        sysbuoyg[sys], sysbuoyf[sys], \
        sysmommixconv[sys], sysnonhmconv[sys], \
        sysdermommix[sys], \
        sysdeltapgx[sys], sysdeltapfx[sys]
printf "\n"
printf "38\n"
printf "*-----\n"
printf "%s_%s_terms.dat\n", FILE, SYSNAME[sys]
printf "(%s(E17.8))\n", 22+1
printf "(%s(A16,x))\n", 22+1
printf "%s 0 %6d\n", FREQ, ((STARTTIME-INITTIME)/MAXDT)+2
printf "sysjtermg-%s\n", SYSNAME[sys]
printf "sysjtermf-%s\n", SYSNAME[sys]
printf "systhexpg-%s\n", SYSNAME[sys]
printf "systhexpf-%s\n", SYSNAME[sys]
printf "sysswellg-%s\n", SYSNAME[sys]
printf "sysswellf-%s\n", SYSNAME[sys]
printf "sysacg-%s\n", SYSNAME[sys]
printf "sysacf-%s\n", SYSNAME[sys]
printf "syscompg-%s\n", SYSNAME[sys]
printf "syscompf-%s\n", SYSNAME[sys]
printf "sysnqthexpg-%s\n", SYSNAME[sys]
printf "sysnqthexpf-%s\n", SYSNAME[sys]
printf "sysdeltap-%s\n", SYSNAME[sys]
printf "syswfricg-%s\n", SYSNAME[sys]

```

```

printf "syswfricf-%s\n", SYSNAME[sys]
printf "sysbuoyg-%s\n", SYSNAME[sys]
printf "sysbuoyf-%s\n", SYSNAME[sys]
printf "sysmommixconv-%s\n", SYSNAME[sys]
printf "sysnonhmconv-%s\n", SYSNAME[sys]
printf "sysdermommix-%s\n", SYSNAME[sys]
printf "sysdeltapgx-%s\n", SYSNAME[sys]
printf "sysdeltapfx-%s\n", SYSNAME[sys]
systemsooutfile[VOLUME[i]] = OUTCOUNT++
printf "***** \n"

# End of system loop
}

for(nht=1; nht<=NHEATSTRS; nht++) {
    leftind = nht * 100
    rightind = nht * 100 + 1
    printf "Time derivative of average stored energy in htstr %s\n", heatst[nht]
    printf "%s 0 2\n", OUTCOUNT
    printf "%s %s\n", htpown[nht], htrnrn[nht]
    printf "27\n"
    printf "*-----\n"
    printf "2.0*(([1]*%s/[s]-[2])/)%s\n", insrcfac[nht], rightsrf[nht], \
        htcoord[rightindex]
    deavgdt[nht] = OUTCOUNT++
    printf "***** \n"
    if (deftype[composition[leftind]] == "table") {
        for(vcppoint=1; vcppoint<tablepoints[composition[leftind]]; vcppoint++) {
            comppoint = (composition[leftind]*100) + vcppoint
            printf "Stored energy in left side of heatst %s. Region %s\n", \
                heatst[nht], vcppoint
            printf "%s 0 2\n", OUTCOUNT
            printf "%s\n", httmpleft[nht]
            printf "27\n"
            printf "*-----\n"
            printf "([1]-%s)*(%s+(", vcp[comppoint], \
                vcpy[comppoint]
            if (slope[comppoint] == 0.0) {
                printf "0.0"
            }
            else {
                printf "%s", slope[comppoint]
            }
            printf "*(%s)/2.0))+", vcp[comppoint]
            if (vcppoint == 1) {
                printf "0.0\n"
            }
            else {
                printf "%s\n", initstengy[comppoint]
            }
            provleftstengy[comppoint] = OUTCOUNT++
            printf "***** \n"
        }
    }
    for(vcppoint=1; vcppoint<tablepoints[composition[leftind]]; vcppoint++) {
        comppoint = (composition[leftind]*100) + vcppoint
        printf "Temperature gt x0 in region %s. Heatst %s left side\n", \
            vcppoint, heatst[nht]
        printf "%s 0 2\n", OUTCOUNT
        printf "%s %s %s\n", CONST0, CONST1, httmpleft[nht]
        printf "14\n"
        printf "*-----\n"
        printf "3\n"
        printf "%s\n", vcp[comppoint]
        tempgtxo[comppoint] = OUTCOUNT++
        OUTCOUNT++
    }
}

```

```

    printf "***** \n"
}
for(vcppoint=2; vcppoint<=tablepoints[composition[leftind]]; vcppoint++) {
    comppoint = (composition[leftind]*100) + vcppoint
    printf "Temperature lt x1 in region %s. Heatst %s left side\n", \
        vcppoint-1, heatst[nht]
    printf "%s 0 2\n", OUTCOUNT
    printf "%s %s %s\n", CONST1, CONST0, httmpleft[nht]
    printf "14\n"
    printf "-----\n"
    printf "3\n"
    printf "%s\n", vcpv[comppoint]
    templtx1[comppoint-1] = OUTCOUNT++
    OUTCOUNT++
    printf "***** \n"
}
for(vcppoint=1; vcppoint<tablepoints[composition[leftind]]; vcppoint++) {
    comppoint = (composition[leftind]*100) + vcppoint
    printf "Stored energy in the range x0-x1 in region %s. Heatst %s left side\n", \
        vcppoint, heatst[nht]
    printf "%s 0 2\n", OUTCOUNT
    printf "%s %s %s\n", tempgtxo[comppoint], templtx1[comppoint], \
        provleftstengy[comppoint]
    printf "27\n"
    printf "-----\n"
    printf "[1]*[2]*[3]\n"
    leftstengyxox1[comppoint] = OUTCOUNT++
    printf "***** \n"
}
printf "Stored energy in left side. Heatst %s\n", heatst[nht]
printf "%s 0 2\n", OUTCOUNT
printf "%s", CONST0
for(vcppoint=1; vcppoint<tablepoints[composition[leftind]]; vcppoint++) {
    comppoint = (composition[leftind]*100) + vcppoint
    printf " %s", leftstengyxox1[comppoint]
}
printf "\n"
printf "27\n"
printf "-----\n"
printf "[1]"
for(vcppoint=2; vcppoint<=tablepoints[composition[leftind]]; vcppoint++) {
    printf "+[%s]", vcppoint
}
printf "\n"
leftstengy[nht] = OUTCOUNT++
printf "***** \n"
}
else if(deftype[composition[leftind]] == "function") {
    for(vcppoint=1; vcppoint<=funcpoints[composition[leftind]]; vcppoint++) {
        comppoint = (composition[leftind]*100) + vcppoint
        printf "Stored energy in left side of heatst %s. Region %s\n", \
            heatst[nht], vcppoint
        printf "%s 0 2\n", OUTCOUNT
        printf "%s\n", httmpleft[nht]
        printf "27\n"
        printf "-----\n"
        printf "([1]-%s)*([1]-%s)*((%s/2.0)+((([1]-%s)*((%s/3.0)+", \
            cterm[comppoint], cterm[comppoint], a1[comppoint], \
            cterm[comppoint], a2[comppoint]
        printf "((([1]-%s)*(%s/4.0)+((([1]-%s)*(%s/5.0)))))+", cterm[comppoint], \
            a3[comppoint], cterm[comppoint], \
            a4[comppoint]
        printf "%s*[1]+%s*LN([1])+", a0[comppoint], a_1[comppoint]
        if (vcppoint == 1) {
            printf "0.0\n"
        }
    }
}

```

```

    }
    else {
        printf "%s\n", initstengy[comppoint]
    }
    provleftstengy[comppoint] = OUTCOUNT++
    printf "***** \n"
}
for(vcppoint=1; vcppoint<=funcpoints[composition[leftind]]; vcppoint++) {
    comppoint = (composition[leftind]*100) + vcppoint
    printf "Temperature gt lowtemp in region %s. Heatst %s left side\n", \
        vcppoint, heatst[nht]
    printf "%s 0 2\n", OUTCOUNT
    printf "%s %s %s\n", CONST0, CONST1, httmpleft[nht]
    printf "14\n"
    printf "-----\n"
    printf "3\n"
    printf "%s\n", lowtemp[comppoint]
    tempgtlow[comppoint] = OUTCOUNT++
    OUTCOUNT++
    printf "***** \n"
}
for(vcppoint=1; vcppoint<=funcpoints[composition[leftind]]; vcppoint++) {
    comppoint = (composition[leftind]*100) + vcppoint
    printf "Temperature lt hightemp in region %s. Heatst %s left side\n", \
        vcppoint, heatst[nht]
    printf "%s 0 2\n", OUTCOUNT
    printf "%s %s %s\n", CONST0, CONST1, httmpleft[nht]
    printf "14\n"
    printf "-----\n"
    printf "3\n"
    printf "%s\n", hightemp[comppoint]
    templhigh[comppoint] = OUTCOUNT++
    OUTCOUNT++
    printf "***** \n"
}
for(vcppoint=1; vcppoint<=funcpoints[composition[leftind]]; vcppoint++) {
    comppoint = (htmatnumb[leftind]*100) + vcppoint
    printf "Stored energy in region %s. Heatst %s left side\n", \
        vcppoint, heatst[nht]
    printf "%s 0 2\n", OUTCOUNT
    printf "%s %s %s\n", tempgtlow[comppoint], templhigh[comppoint], \
        provleftstengy[comppoint]
    printf "27\n"
    printf "-----\n"
    printf "[1]*[2]*[3]\n"
    leftstengyinreg[comppoint] = OUTCOUNT++
    printf "***** \n"
}
printf "Stored energy in left side. Heatst %s\n", heatst[nht]
printf "%s 0 2\n", OUTCOUNT
printf "%s", CONST0
for(vcppoint=1; vcppoint<=funcpoints[composition[leftind]]; vcppoint++) {
    comppoint = (composition[leftind]*100) + vcppoint
    printf " %s", leftstengyinreg[comppoint]
}
printf "\n"
printf "27\n"
printf "-----\n"
printf "[1]"
for(vcppoint=1; vcppoint<=funcpoints[composition[leftind]]; vcppoint++) {
    printf "+[%s]", vcppoint+1
}
printf "\n"
leftstengy[nht] = OUTCOUNT++
printf "***** \n"

```

```

}
else if(deftype[composition[leftind]] == "constant") {
    compoint = (composition[leftind]*100) + 1
    printf "Stored energy in left side. Heatst %s\n", heatst[nht]
    printf "%s 0 2\n", OUTCOUNT
    printf "%s\n", httpmleft[nht]
    printf "27\n"
    printf "*-----\n"
    printf "([1]-300.0)*%s", vcpy[compoint]
    leftstengy[nht] = OUTCOUNT++
    printf "***** \n"
}

if (deftype[composition[rightind]] == "table") {
    for(vcppt=1; vcppt<tablepoints[composition[rightind]]; vcppt++) {
        compoint = (composition[rightind]*100) + vcppt
        printf "Stored energy in right side of heatst %s. Region %s\n", \
            heatst[nht], vcppt
        printf "%s 0 2\n", OUTCOUNT
        printf "%s\n", httpmright[nht]
        printf "27\n"
        printf "*-----\n"
        printf "([1]-%s)*(%s+(", vcp[compoint], \
            vcpy[compoint]
        if (slope[compoint] == 0.0) {
            printf "0.0"
        }
        else {
            printf "%s", slope[compoint]
        }
        printf "*( [1]-%s)/2.0))+", vcp[compoint]
        if (vcppt == 1) {
            printf "0.0\n"
        }
        else {
            printf "%s\n", initstengy[compoint]
        }
        provrightstengy[compoint] = OUTCOUNT++
        printf "***** \n"
    }
    for(vcppt=1; vcppt<tablepoints[composition[rightind]]; vcppt++) {
        compoint = (composition[rightind]*100) + vcppt
        printf "Temperature gt x0 in region %s. Heatst %s right side\n", \
            vcppt, heatst[nht]
        printf "%s 0 2\n", OUTCOUNT
        printf "%s %s %s\n", CONST0, CONST1, httpmright[nht]
        printf "14\n"
        printf "*-----\n"
        printf "3\n"
        printf "%s\n", vcp[compoint]
        tempgtxo[compoint] = OUTCOUNT++
        OUTCOUNT++
        printf "***** \n"
    }
    for(vcppt=2; vcppt<=tablepoints[composition[rightind]]; vcppt++) {
        compoint = (composition[rightind]*100) + vcppt
        printf "Temperature lt x1 in region %s. Heatst %s right side\n", \
            vcppt-1, heatst[nht]
        printf "%s 0 2\n", OUTCOUNT
        printf "%s %s %s\n", CONST1, CONST0, httpmright[nht]
        printf "14\n"
        printf "*-----\n"
        printf "3\n"
        printf "%s\n", vcp[compoint]
        templtx1[compoint-1] = OUTCOUNT++
    }
}

```

```

    OUTCOUNT++
    printf "***** \n"
}
for(vcppoint=1; vcppoint<tablepoints[composition[rightind]]; vcppoint++) {
    comppoint = (composition[rightind]*100) + vcppoint
    printf "Stored energy in the range x0-x1 in region %s. Heatst %s right side\n", \
        vcppoint, heatst[nht]
    printf "%s 0 2\n", OUTCOUNT
    printf "%s %s %s\n", tempgtxo[comppoint], templtx1[comppoint], \
        provrightstengy[comppoint]

    printf "27\n"
    printf "-----\n"
    printf "[1]*[2]*[3]\n"
    rightstengyxox1[comppoint] = OUTCOUNT++
    printf "***** \n"
}

printf "Stored energy in right side. Heatst %s\n", heatst[nht]
printf "%s 0 2\n", OUTCOUNT
printf "%s", CONSTO
for(vcppoint=1; vcppoint<tablepoints[composition[rightind]]; vcppoint++) {
    comppoint = (composition[rightind]*100) + vcppoint
    printf " %s", rightstengyxox1[comppoint]
}

printf "\n"
printf "27\n"
printf "-----\n"
printf "[1]"
for(vcppoint=2; vcppoint<=tablepoints[composition[rightind]]; vcppoint++) {
    printf "+[%s]", vcppoint
}

printf "\n"
rightstengy[nht] = OUTCOUNT++
printf "***** \n"
}

else if(deftype[composition[rightind]] == "function") {
    for(vcppoint=1; vcppoint<=funcpoints[composition[rightind]]; vcppoint++) {
        comppoint = compoint = (composition[rightind]*100) + vcppoint
        printf "Stored energy in right side of heatst %s. Region %s\n", \
            vcppoint, heatst[nht]
        printf "%s 0 2\n", OUTCOUNT
        printf "%s\n", httptright[nht]
        printf "27\n"
        printf "-----\n"
        printf "([1]-%s)*([1]-%s)*((%s/2.0)+((([1]-%s)*((%s/3.0)+", \
            cterm[comppoint], cterm[comppoint], a1[comppoint], \
            cterm[comppoint], a2[comppoint]
        printf "((([1]-%s)*(%s/4.0)+((([1]-%s)*(%s/5.0)))))+", cterm[comppoint], \
            a3[comppoint], cterm[comppoint], a4[comppoint]
        printf "%s*[1]+%s*LN([1])+", a0[comppoint], a_1[comppoint], \
            initstengy[comppoint]

        if (vcppoint == 1) {
            printf "0.0\n"
        }
        else {
            printf "%s\n", initstengy[comppoint]
        }
        provrightstengy[comppoint] = OUTCOUNT++
        printf "***** \n"
    }

    for(vcppoint=1; vcppoint<=funcpoints[composition[rightind]]; vcppoint++) {
        compoint = (composition[rightind]*100) + vcppoint
        printf "Temperature gt lowtemp in region %s. Heatst %s right side\n", \
            vcppoint, heatst[nht]
        printf "%s 0 2\n", OUTCOUNT
        printf "%s %s %s\n", CONSTO, CONST1, httptright[nht]
    }
}

```

```

printf "14\n"
printf "-----\n"
printf "3\n"
printf "%s\n", lowtemp[comppoint]
tempgtlow[comppoint] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
}
for(vcppoint=1; vcppoint<=funcpoints[composition[rightind]]; vcppoint++) {
comppoint = (composition[rightind]*100) + vcppoint
printf "Temperature lt hightemp in region %s. Heatst %s right side\n", \
vcppoint, heatst[nht]
printf "%s 0 2\n", OUTCOUNT
printf "%s %s %s\n", CONST0, CONST1, httptright[nht]
printf "14\n"
printf "-----\n"
printf "3\n"
printf "%s\n", hightemp[comppoint]
templthigh[comppoint] = OUTCOUNT++
OUTCOUNT++
printf "***** \n"
}
for(vcppoint=1; vcppoint<=funcpoints[composition[rightind]]; vcppoint++) {
comppoint = (composition[rightind]*100) + vcppoint
printf "Stored energy in region %s. Heatst %s right side\n", \
vcppoint, heatst[nht]
printf "%s 0 2\n", OUTCOUNT
printf "%s %s %s\n", tempgtlow[comppoint], templthigh[comppoint], \
provrightstengy[comppoint]

printf "27\n"
printf "-----\n"
printf "[1]*[2]*[3]\n"
rightstengyinreg[comppoint] = OUTCOUNT++
printf "***** \n"
}
printf "Stored energy in right side. Heatst %s\n", heatst[nht]
printf "%s 0 2\n", OUTCOUNT
printf "%s", CONST0
for(vcppoint=1; vcppoint<=funcpoints[composition[rightind]]; vcppoint++) {
comppoint = (composition[rightind]*100) + vcppoint
printf " %s", rightstengyinreg[comppoint]
}
printf "\n"
printf "27\n"
printf "-----\n"
printf "[1]"
for(vcppoint=1; vcppoint<=funcpoints[composition[rightind]]; vcppoint++) {
printf "+[%s]", vcppoint+1
}
printf "\n"
rightstengy[nht] = OUTCOUNT++
printf "***** \n"
}
else if(deftype[composition[rightind]] == "constant") {
comppoint = (composition[rightind]*100) + 1
printf "Stored energy in right side. Heatst %s\n", heatst[nht]
printf "%s 0 2\n", OUTCOUNT
printf "%s\n", httptright[nht]
printf "27\n"
printf "-----\n"
printf "([1]-300.0)*%s", vcpy[comppoint]
rightstengy[nht] = OUTCOUNT++
printf "***** \n"
}
printf "Left stengy + 2*right stengy in heatst %s\n", heatst[nht]

```

```

printf "%s 0 2\n", OUTCOUNT
printf "%s %s\n", leftstengy[nht], rightstengy[nht]
printf "27\n"
printf "*-----\n"
printf "[1]+2.0*[2]\n"
e1plus2e2[nht] = OUTCOUNT++
printf "***** \n"
printf "Time derivative of e1 plus 2*e2 in heatst %s\n", heatst[nht]
printf "%s 0 2\n", OUTCOUNT
printf "%s\n", e1plus2e2[nht]
printf "43\n"
printf "*-----\n"
printf "16 1.0\n"
de1plus2e2dt[nht] = OUTCOUNT++
printf "***** \n"
printf "Post. time derivative of av. stored engy. in heatst %s\n", heatst[nht]
printf "%s 0 2\n", OUTCOUNT
printf "%s\n", de1plus2e2dt[nht]
printf "27\n"
printf "*-----\n"
printf "[1]/3.0\n"
postdeavgdt[nht] = OUTCOUNT++
printf "***** \n"
printf "Time derivative of e1 plus 2*e2 - 3*eaavg in heatst %s\n", heatst[nht]
printf "%s 0 2\n", OUTCOUNT
printf "%s %s\n", de1plus2e2dt[nht], deavgdt[nht]
printf "27\n"
printf "*-----\n"
printf "[1]-3.0*[2]\n"
de1p2e23e3dt[nht] = OUTCOUNT++
printf "***** \n"

# End of heat structures loop
}
if(NHEATSTRS > 0) {
printf "OUTPUT FILE HEAT STRUCTURES \n"
printf "%s\n", OUTCOUNT
for (nht=1; nht<=NHEATSTRS; nht++) {
printf "%s %s %s ", deavgdt[nht], de1p2e23e3dt[nht], postdeavgdt[nht]
}
printf "\n"
printf "38\n"
printf "*-----\n"
printf "%s_heatst.dat\n", FILE
printf "(%s(E17.8))\n", 3*NHEATSTRS+1
printf "(%s(A16,x))\n", 3*NHEATSTRS+1
printf "%s 0 %6d\n", FREQ, ((STARTTIME-INITTIME)/MAXDT)+2
for (nht=1; nht<=NHEATSTRS; nht++) {
printf "deavgdt-%s\n", heatst[nht]
printf "de1p2e23e3dt-%s\n", heatst[nht]
printf "postdeavgdt-%s\n", heatst[nht]
}
heatstoutfile = OUTCOUNT++
}
printf "***** \n"
}

```