INL/EXT-10-19984

Development, Selection, Implementation, and Testing of Architectural Features and Solution Techniques for Next Generation of System Simulation Codes to Support the Safety Case of the LWR Life Extension

Robert Nourgaliev
Nam Dinh
Robert Youngblood

December 2010

INL/EXT-10-19984

Development, Selection, Implementation, and Testing of Architectural Features and Solution Techniques for Next Generation of System Simulation Codes to Support the Safety Case of the LWR Life Extension

Robert Nourgaliev
Nam Dinh
Robert Youngblood

December 2010

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

Development, Selection, Implementation, and Testing of Architectural Features and Solution Techniques for Next Generation of System Simulation Codes to Support the Safety Case of the LWR Life Extension

Robert Nourgaliev, Nam Dinh, and Robert Youngblood

December 13, 2010

Idaho National Laboratory

This Page is Intentionally Left Blank

Development, Selection, Implementation, and Testing of Architectural Features and Solution Techniques for Next Generation of System Simulation Codes to Support the Safety Case of the LWR Life Extension

ROBERT NOURGALIEV, NAM DINH, AND ROBERT YOUNGBLOOD

Nuclear Science & Technology
Idaho National Laboratory
P.O. Box 1625, Idaho Falls, ID 83415-3840

“RISMC β_2”, 196 p.
December 13, 2010

This Page is Intentionally Left Blank

Abstract

DEVELOPMENT, demonstration, and validation of the Risk-Informed Safety Margin Characterization (RISMC) methodology and tools are planned and performed in the U.S. Department of Energy's LWR Sustainability Program, with the objective to support decision-making on extending plant life beyond 60 years. This report documents current status, results and insights derived from research and development on architecture and algorithms of a next generation system safety simulation code, whose goal is to enable characterization of safety margins in nuclear power plants.

Architecture, functions, and solution techniques as conceptualized, selected, and implemented in a β_2 code version are described in the report. Progress made to date including lessons learned from testing the computational framework and numerical methods on a set of test problems provide a technical basis for assessment and revision of the code architecture and algorithms. Most notably, the experience suggests that the computational framework and solution algorithms developed have potential to meet complex requirements of a next-generation system code, including simulation accuracy and speed, and developmental scalability. Substantial challenges remain in modeling and validation needed to enable multi-physics, multi-scale simulations as required in an advanced system safety code.

This Page is Intentionally Left Blank

Acknowledgements

THIS work has been authored by Battelle Energy Alliance, LLC under contract No. DE-AC07-05ID14517 (INL/EXT-10-19984) with the U.S. Department of Energy. The United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manual, or allow others to do so, for United States Government purposes.

The work described in this report has benefitted from fruitful interactions with experts including academic, industry, and INL collaborators in the LWR Sustainability Program's Risk-Informed Safety Margin Characterization R&D.

This Page is Intentionally Left Blank

Contents

This Page is Intentionally Left Blank

List of Tables

This Page is Intentionally Left Blank

List of Figures

Table 1 : Acronyms.

Abbreviation	**Concept**
BEPU	Best Estimate Plus Uncertainty
BWR	Boiling Water Reactor
CDF	Core Damage Frequency
CFD	Computational Fluid Dynamics
CIAU	Code with Internal Assessment of Uncertainty
CSAU	Code Scaling Applicability and Uncertainty
DBA	Design Basis Accident
DG (DGM)	Discontinuous Galerkin (Method)
DOE	Department of Energy
DPRA	Dynamic PRA
JFNK	Jacobian-Free Newton Krylov
EMDAP	Evaluation Model Development and Assessment Process
FAB	Feed and Bleed
FEM	Finite Element Method
LWR	Light Water Reactor
MP-MS	Multi-Physics, Multi-Scale
NRC	Nuclear Regulatory Commission
PDF	Probability Density Function
PIRT	Phenomena Identification and Ranking Table
PRA	Probabilistic Risk Assessment
PSA	Probabilistic Safety Analysis
PWR	Pressurized Water Reactor
QMU	Quantification of Margins and Uncertainty
QRA	Quantitative Risk Analysis
rDG	Reconstructed Discontinuous Galerkin
RISMC	Risk-Informed Safety Margin Characterization
ROAAM	Risk-Oriented Accident Analysis Methodology
SAR	Safety Analysis Report
SMAP	Safety Margins Action Plan
STH	System Thermal Hydraulics Code
TNF	Technology-Neutral Framework
UC	Use Case
VU	Verification and Validation, and Uncertainty Quantification

This Page is Intentionally Left Blank

Chapter 1

Introduction

THIS report documents current status of conceptualization, development, testing, and demonstration of a next-generation system safety simulation code to support risk-informed safety margin characterization (RISMC code). It builds on experience and insights gained from R&D performed at INL and collaborating institutions, both under the LWR Sustainability Program and beyond.

The project has its origin in an initiative started at the Idaho National Laboratory (INL) in FY 2009 as laboratory-directed research and development (R&D), with the goal to establish the methodology and technology for a next-generation safety simulation tool. In the authors' minds, the notion of "next generation" has effectively boiled off to the concept of Risk-Informed Safety Margin Characterization, whose "logo" is shown in Figure 1.1. The idea is to extend from the deterministic margin into probabilistic loading versus probabilistic capacity, and in doing so, delineating and capturing uncertainty of all types and origins. Although in practice, margins is a multi-dimensional notion, the logo conveys, comprehensively and quite accurately, the drive and the spirit of this project.

Ultimately, the RISMC code must integrate the deterministic/mechanistic method of system process description with the stochastic/probabilistic method of reliability/risk assessment to provide a complete, consistent and comprehensive characterization of safety margins in a nuclear power plant; see Figure 1.2. In other words, it enables "virtual plant safety testing", where changes in plant conditions, operating procedures, applications of innovative elements (fuels, claddings, monitoring), etc. show their impact on plant safety margins broadly defined.

1.1 Code functions

Thus, the RISMC code's basic functions are to compute loading and to compute capacity on a specified systems, structures, and components (SSCs) at any given time of the plant life, including during the postulated life extension regime. Traditionally, "safety margins" are measured in term of stressors e.g. peak cladding temperature, which are surrogate criteria for safety [1].

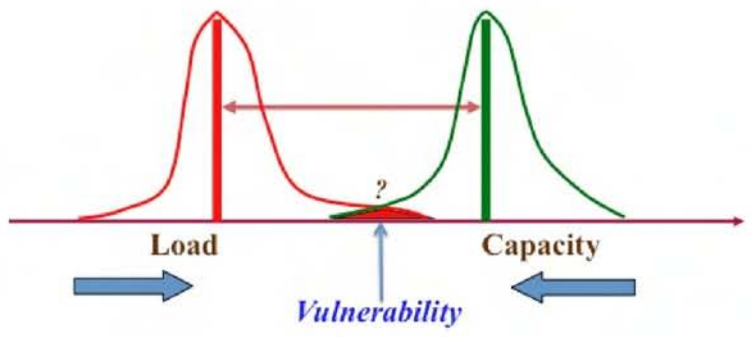

Fig. 1.1 : The logo of the RISMC concept, which is designed to bring together methods of deterministic and probabilistic analysis. The "safety margin" is a manifestation of integrated uncertainty. The two-ended arrow shows the traditional deterministic margin. The left arrow reflects the economic drive to achieve higher power (uprate), higher fuel burnup. The right arrow reflects degradation due to plant aging. Howver, these trends can be reversed with an effective aging management program, improved operating procedures, maintenance, and operator training, among others. The "vulnerability" is scenarios of safety and operability concerns. Their early identification, prevention and mitigation constitute the essence of Risk Assessment and Management.

Stressors come in different categories: mechanical, thermal, irradiation and

[1] As example of how complex the surrogate metrics can be, we refer to an OECD/NEA study which exhibits 34 parameters used to represent nuclear fuel's design, operating, and safety criteria.One objective of the RISMC research is to suggest a simple and adequate formulation of safety margins . One of the supporting function of the RISMC code is to provide comprehensive visualization of the safety margins.

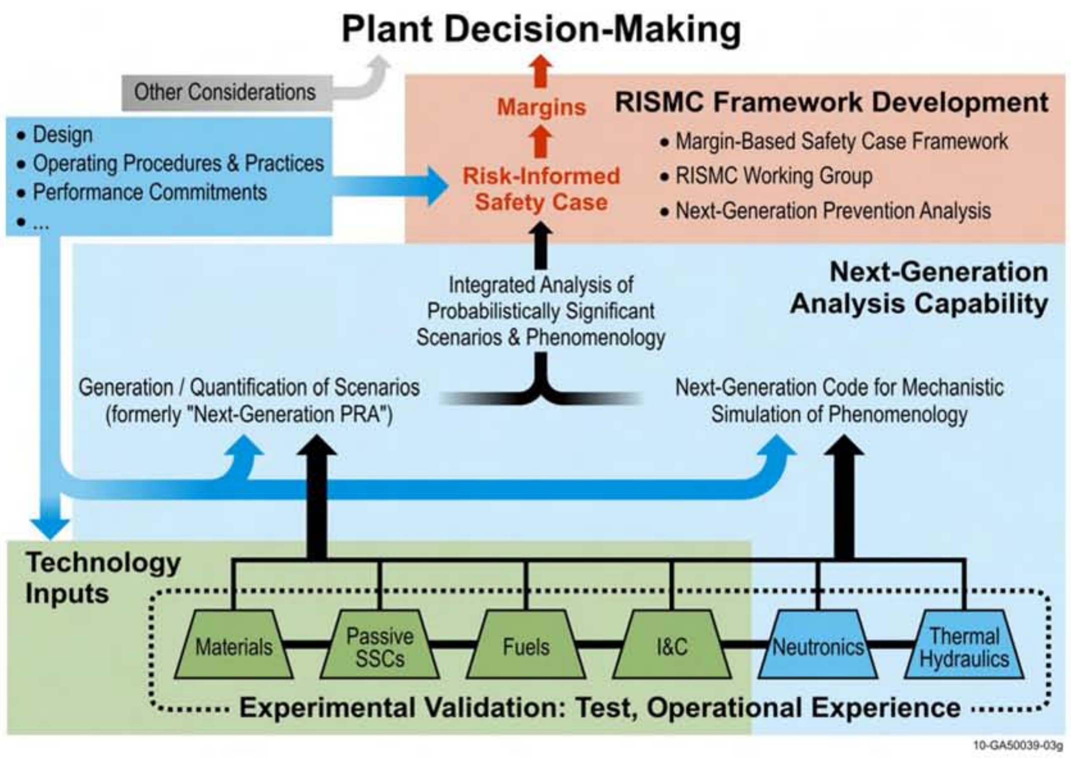
Plant Decision-Making
Other Considerations
• Design
• Operating Procedures & Practices
• Performance Commitments
• ...
Margins
Risk-Informed Safety Case
RISMC Framework Development
• Margin-Based Safety Case Framework
• RISMC Working Group
• Next-Generation Prevention Analysis
Next-Generation Analysis Capability
Integrated Analysis of Probabilistically Significant Scenarios & Phenomenology
Generation / Quantification of Scenarios (formerly "Next-Generation PRA")
Next-Generation Code for Mechanistic Simulation of Phenomenology
Technology Inputs
Materials
Passive SSCs
Fuels
I&C
Neutronics
Thermal Hydraulics
Experimental Validation: Test, Operational Experience
10-GA50039-03g

Fig. 1.2 : A Next Generation System Safety Simulation Software (RISMC code) as enabling tool for the RISMC-based "safety case". In here, "Risk-Informed Safety Case" is a documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment. The "Margins" is RISMC support to Decision Making based on implications of margins evaluated in a risk-informed paradigm (Figure adapted from R. Youngblood, "RISMC Framework", 2010.

chemical; Figure 1.3. A particular stressor or parameter is considered important because it represents a challenge to integrity or operability of the SSC under consideration.

Figure 1.3 shows modeling and simulation capabilities that the RISMC code needs to possess in order to compute the loading. This includes simulation of physical processes (left cluster) and operating factors (right cluster), and modeling of the plant systems which bring the physics and the operation into "loading".

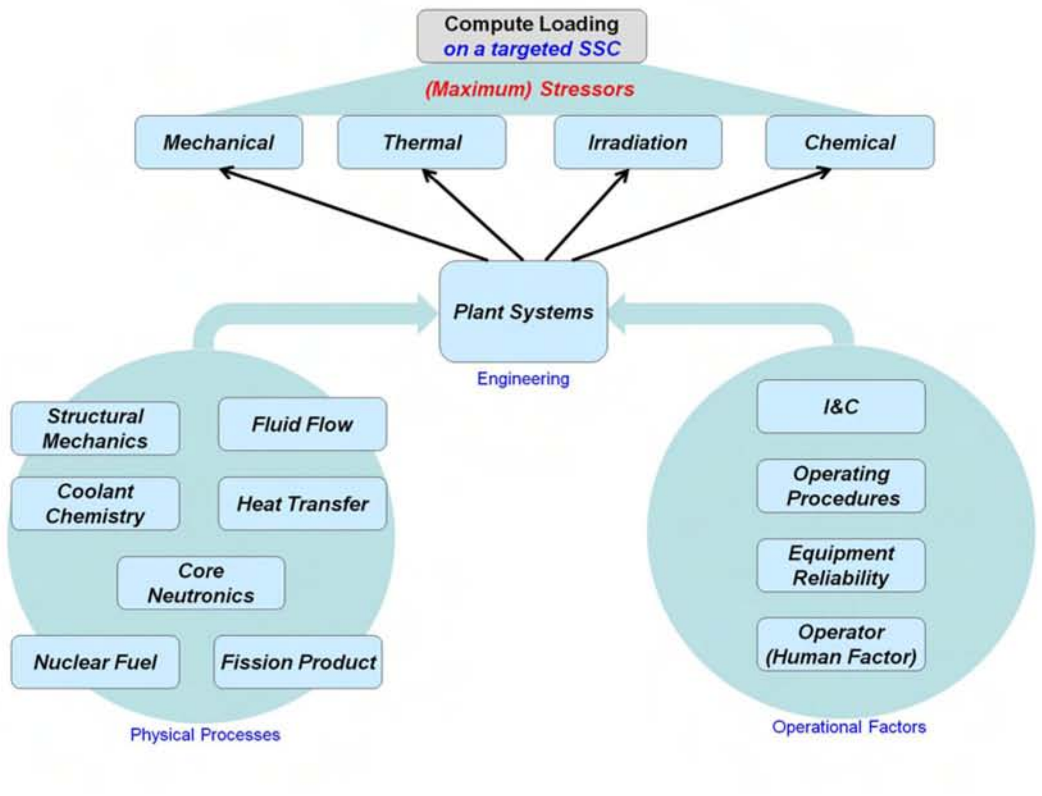

Fig. 1.3 : The RISMC code's basic function "Compute Loading" and related capabilities. The driving physics are abnormal transients and accidents over short time scale with potential for peak loading.

Similarly, Figure 1.4 shows the capabilities required to compute capacity. It can be seen that - in addition to simulation capabilities in support of loading computation - capacity computation requires

- (i) "damage calculation" capability, which translate stressors into materials damage, including the synergistic effect of multiple stressors on damage initiation and growth; and

- (ii) "degradation calculation" capability, which translates the materials damage measures into degradation of structural materials strength, including the synergistic effect of multiple types of damages on degradation of material physical / functional properties.

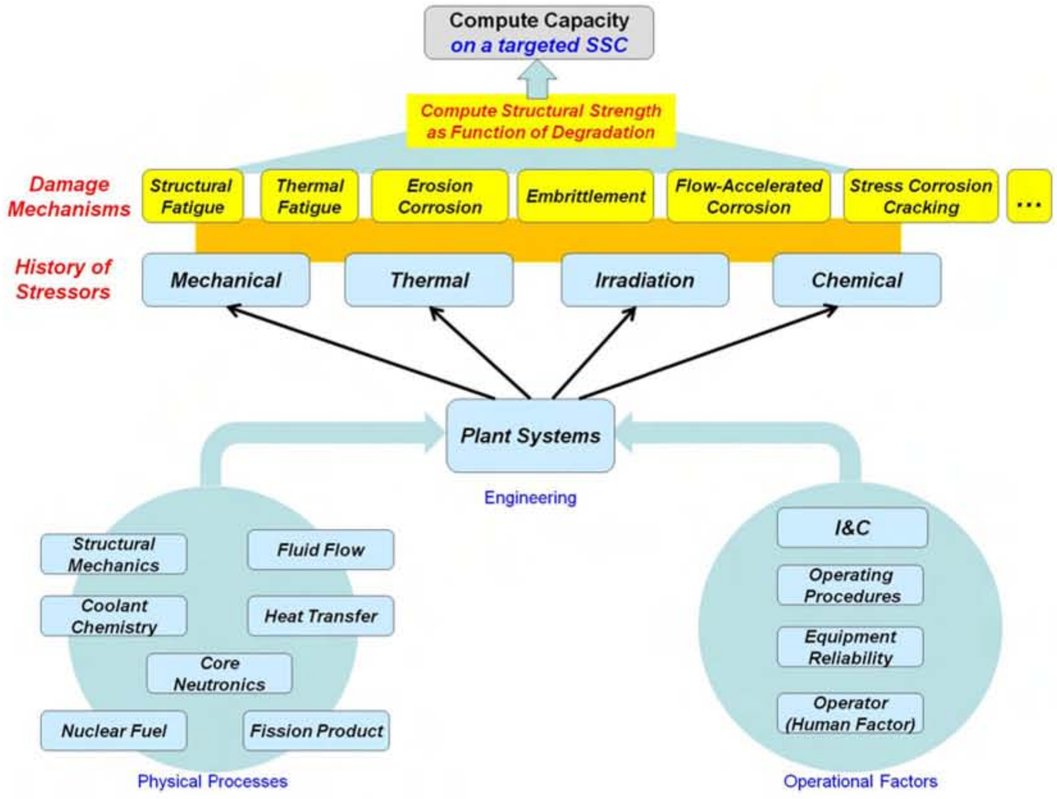

Fig. 1.4 : The RISMC code's basic function "Compute Capacity" and related capabilities. The driving physics is degradation over decades-long time scale.

It is noted that although the simulation capabilities under "physical process" and "operating factors" appear identical for loading computation and capacity computation functions, the actual emphasis and implementation of these capabilities differ for two functions. It is because the capacity computation function is concerned with long (years) time scale of degradation processes, whereas the loading computation function deals with short (minutes to hours) time scale of accident processes.

There are circumstances when the two functions largely overlap or tightly couple.

For example, the loading computation requires the long "aging" result ("plant health" capacity) as initial condition for transient analysis. In turn, the capacity computation for a real plant must also include transient loading analysis to capture operational, anticipated and abnormal (typically, mild) transients that have occurred or been postulated to occur during the plant operation [2]. Such abnormal transients are undesirable with respect to both plant safety and operability, as they often induce substantial stresses on the plant's SSC, and some time, these transients introduce stressors, which are not considered in the "normal aging" scenarios. Such periods of accelerated degradation have potential to physically age the plant much faster [3].

Another example of tight coupling between loading and capacity occurs when a rapid degradation (capacity erosion) leads to a failure that alters the sequence. This is the case when a static, pre-calculated notion of capacity (structural strength) ceases to be valid. It is noted that in such cases the term "capacity" refers to intermediate margins, not the ultimate margin (capacity) targeted in the analysis. When "loading exceeds capacity" situation occurs for a SSC (which is not the defined search objective) during a transient, it leads to SSC failure. Modeling of failure, often fast-time-scale thermo-fluid-mechanical coupled processes, pertains to the domain of "physical processes" in Figure 1.3.

[2]Such analysis belongs to category of multi-scale integration, that involves vastly different time scales, from minutes for transient to years for aging.

[3]Optimization of resource utilization has been referred to as "heartbeat" analysis, which accounts for how abnormal events ("fast beats" affect the life span (e.g. total number of heartbeats). It is noted that besides negative events with fast heartbeats, there are positive events e.g. plant maintenance, which are designed to increase the "life-given" number of heartbeats or slow down the plant's heartbeats during the normal operation.

1.2 Highlight of Requirements

In this section, we highlight major requirements, which drive selection of architecture and algorithms discussed in subsequent Chapters.

1.2.1 Multi-Scale / Multi-Physics Integration

THE set of system and physics models provides the backbone of the RISMC code. These models are given in the form of algebraic, algebro-differential, and integro-differential equations(transient 0D models), and partial differential equations (transient 1D to 3D models). These equations must be solved through integration in time and space.

In general, processes in nuclear power plants entail multiple physics of mechanistic / deterministic (e.g., core neutronics, thermal hydraulics, coolant chemistry, structural mechanics) and stochastic / probabilistic (e.g. reliability) nature, spaning over a broad range of time scales, from fraction of a second to decades, and length scales, from fraction of a millimeter to tens meters. The later manifest multi-scale nature of the problems addressed by the RISMC code. It is noted, however, that the RISMC method aims at engineering solution, so processes and effects are modeled at engineering scale. While there exist a number of approaches to multi-scale modeling and simulation[4], the RISMC code adopts universal continuum mechanics approximation over the engineering-system scales treated by the code. In other words, the "same-equation" formulation is used in the RISMC code to provide multi-scale treatment of the physical processes modeled.

A. Multiple Time Scales: "Gap-Tooth" Scheme

Dynamic time step control is instrumental to the computational efficiency needed to simulate physical processes, which involve periods of slow changes (e.g. aging) and fast changes (e.g. plant transients). Capability to dynamically control time step is also essential for the treatment of coupled physical processes with distinctive time scales. The RISMC code applies a so-called "gap-tooth" treatment, in which faster-time-scale phenomena are computed in short "tooth" periods, while

[4]Multi-scale methods applied to physics problems typically involve different physical models at different scales, e.g. continuum mechanics coupled with molecular dynamics or atomistic simulations.

characteristics derived from the "tooth" simulation are used (e.g. to determine source term) in simulations of longer-time-scale processes over a "gap" period between subsequent "teeth". In turn, numerical solution obtained in the "gap" problem provides information needed to dynamically update initial and boundary conditions for the "tooth" problem. The "derived" is underlined to emphasize two, relatively-independent processes whose interaction occurs at fast time scale (high-frequency phenomena); see Figure 1.5.

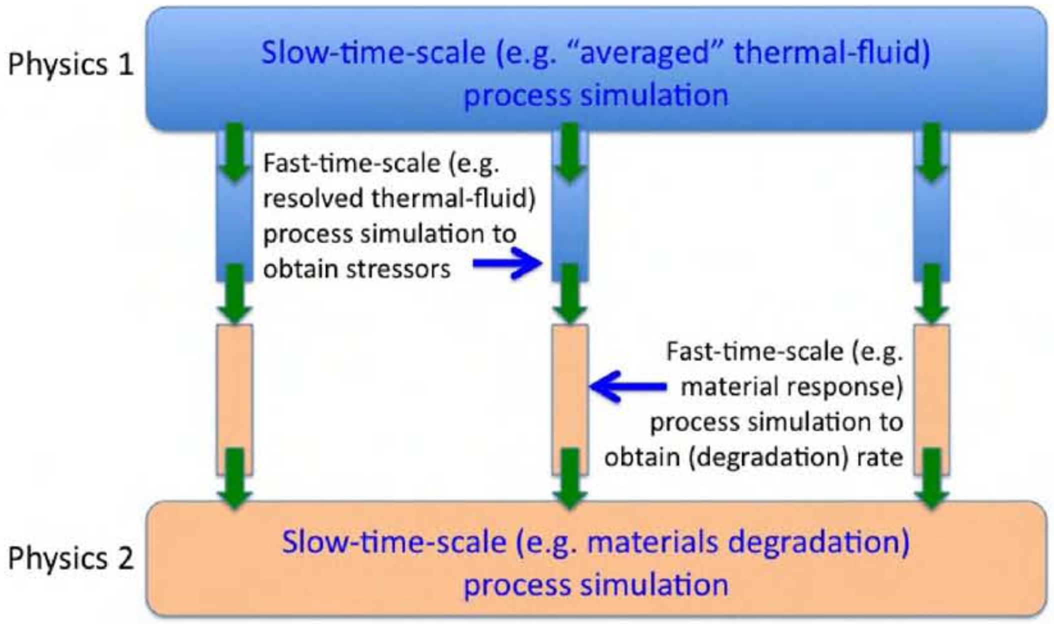
Physics 1
Slow-time-scale (e.g. "averaged" thermal-fluid) process simulation
Fast-time-scale (e.g. resolved thermal-fluid) process simulation to obtain stressors
Fast-time-scale (e.g. material response) process simulation to obtain (degradation) rate
Physics 2
Slow-time-scale (e.g. materials degradation) process simulation

Fig. 1.5 : The "gap-tooth" scheme.

The "gap-tooth" scheme appears well suited for the treatment of multiple time scales in the RISMC Use Cases. Example of high-frequency phenomena are thermal stripping and fluid-structure-interactions-induced vibration that govern thermal fatigue and structural fatigue over a long period of plant operation (aging). The fatigue degradation itself can be calculated by integrating an accumulative damage process model with large time steps, using the fatigue rate calculated by integrating a fine-grain thermal-fluid-structure model in the "tooth" periods. The later aims to capture stressors and material response in the SSC area of interest.

Beyond the "gap-tooth" treatment invoked in special cases, the RISMC code is designed to use a single time stepping for all physics and components, in order to facilitate engineering analysis and review. The time step is chosen, dynamically, to accommodate the fastest change in the system.

B. Multiple Length Scales: Domain Decomposition Scheme

Due to the nuclear power plant plant system complexity, for a risk-informed analysis, the RISMC code necessarily involves lumped-parameter models (0D) for a large number of SSCs, where higher resolution / higher dimensionality are neither necessary nor they reduce uncertainty. However, in certain Use Cases, especially at their later phases, three-dimensional (3D) modeling becomes necessary when model simplifications and discretization errors are found (or assessed) to dominate accuracy and uncertainty in the computation of key stressors. As a result, the RISMC code embodies a range of dimensionality of physical models, from 0D to 1D to 3D, in different SSCs.

It should be noted that - while coupling 3D solutions (e.g. CFD code) to 1D solutions (e.g. RELAP5 system thermal-hydraulics code) has been attempted in the past, the main distinction of the new 0D-1D-3D treatment in the RISMC code is that the models of different resolution are treated in the same code, i.e. subject to the same time and space discretization scheme, and solved by the same solution method. Such an approach aims to eliminate numerical errors brought by coupling of separate codes, with different discretization schemes and solution algorithms.

In the RISMC code, the 3D treatment can be either relatively isolated and stand-alone (e.g. structural mechanics of in area of reactor pressure vessel in the proximity of cold leg connection; or fuel pellet / cladding) or tightly coupled with adjacent domains with lower dimensionality (e.g. thermal-hydraulics in downcomer or lower plenum areas of the reactor pressure vessel coupled to 1D models of piping thermal-hydraulics). Most sensitive to the coupling of domains with varied dimensionality are thermal-hydraulics models due to their hyperbolicity and dynamic character. For this purpose, the Reconstructed Discontinuous Galerkin (rDG) method possesses unique features to meet the challenge.

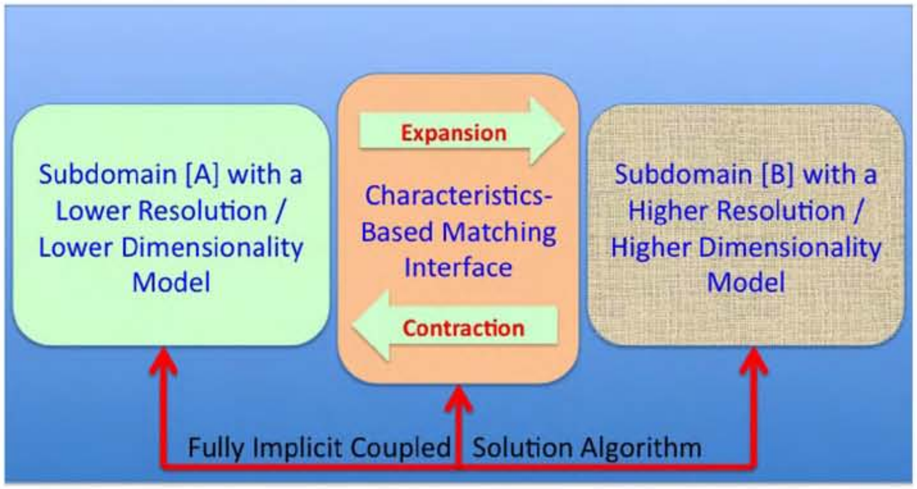
Subdomain [A] with a Lower Resolution / Lower Dimensionality Model
Expansion
Characteristics-Based Matching Interface
Contraction
Subdomain [B] with a Higher Resolution / Higher Dimensionality Model
Fully Implicit Coupled Solution Algorithm

Fig. 1.6 : Domain decomposition scheme requires a characteristics-based matching interface treatment and fully implicit nonlinearly coupled solution to ensure computational stability and effectiveness.

To achieve high robustness and traceability of the simulations, the RISMC code applies a domain decomposition scheme to handle multi-resolution models. In fact, in each sub-domain, there may be several models of different dimensionality and level of details (resolution). Selection of one over another model - for a physics within a domain - depends on a number of factors, including projection of benefits in uncertainty reduction over the added computational expenses in higher resolution models. Uncertainty reduction as a measure is most useful as it accounts for

- errors due to added uncertainty in constitutive relations / parameters required for closure of higher-resolution models;
- errors due to (in)consistency in the treatment of other physics in the same subdomain; and
- errors introduced by deficient treatment of interface (e.g. contraction / expansion) between adjacent subdomains.

While there exist a variety of algorithms for multi-scale domain decomposition treatment, the RISMC code should be designed to solve the coupled models on different domains simultaneously and implicitly, including their subdomain-to-subdomain interface contraction / expansion conditions. Such an implicit algorithm ensures the numerical solution stability, particularly important for bridging different resolutions in adjacent subdomains. Instead of "primitive-variables" coupling, a characteristics-based interface algorithm should be used to ensure two-way smooth information flow (waves) between adjacent subdomains[5] ; see Figure 1.6.

With respect to software implementation, domain decomposition is consistent with the object-oriented simulation environment required for the system analysis [6]. In making their meshing / nodalization decisions, the code users take into account experience (of various origins), analysis needs (varied at different phases of application), resource availability and guidelines from the code manual.

C. Multi-Physics Integration

By its design, the RISMC code eventually involves a large number of systems and physics, far larger than in methods currently used for plant safety analysis (i.e. thermal hydraulics, core neutronics). The more physics and systems become involved, the larger the spread of time and length scales of contributing phenomena. The interactions between different physics are further complicated by the effect of instrumentation and control, SSC reliability and operator actions. Such complications put severe challenges on multi-physics integration.

- First, and foremost, it reveals the limitations of the traditional approach which employs "loose coupling" of tools inherited from "divide-and-conquer" (operator split) era in nuclear reactor engineering.

[5]The characteristics-based matching (CBM) method developed by Nourgaliev, Dinh and Theofanous (2004) [NDT04] is an example.

[6]Adaptive mesh refinement (AMR), and more recently, adaptive model refinement (AMoR), are attractive concepts that can be used to address computational efficiency in a problem that involves multiple length scales. These concepts are however still under development for the types of problems of interest in reactor safety. More importantly, robustness and traceability are critical requirements for simulations whose results are part of a safety analysis report and license application submittal. Generation of computable meshes (nodalization) by the code users prior to simulation also help increase auditability and interrogability of the simulation results by other analysis tools.

  - Errors introduced by the operator split are shown to be significant in nonlinear problems; [KMCR05].
  - Furthermore, such numerical "biases" can accumulate over time, and become particularly detrimental for solution of evolutionary problems like plant aging simulations.

- Secondly, and also directly relevant to "aging", is the motivation to eliminate stability constraints that limit the time stepping in long transient simulations.

Enter fully implicit tightly coupled solution algorithms for multi-physics simulations, using so-called Jacobian-Free Newton Krylov (JFNK) method; see e.g. [KK04]. Most notably, with its "Jacobian-Free" quality, the JFNK method allows new physics be incorporated in the simulation without having to redo a substantial amount of - often impossible - analytical and programming work. The fully implicit simulation can be achieved (and has been achieved in the RISMC code developmental version) by using high-order-accurate time discretization schemes, which are variations of the Runga-Kutta method.

Efficiency of JFNK method, however, can deteriorate with the spread of physical time scales, which increase the problem's stiffness. The issue can be alleviated via development of advanced algorithms and by addressing the stiffness at the model formulation level.

- Algorithmically, the JFNK method convergence can be accelerated by using effective preconditioners. This includes

  - Mathematical (e.g. ILU) preconditioners, and
  - Consistent physics-based preconditioners; [MKR00, MKR04, RWMK03, RMWK05].

  Both types should be pursued.

- Fundamentally, it boils down to having homogenized models, whose time / length scales are consistent with the spatio-temporal resolution intended in the RISMC code simulations.

  - The "intended" resolution varies over the system domain and differently in different applications.

    * In system safety simulations, there is no need, and resources, to resolve fine-grain fluid motions.
    * However, it is important that the simulations capture flow patterns and other characteristics of importance for safety margins.
    * The premise is that the system dynamics and key safety parameters can be predicted with a reasonable accuracy without resolving the details of boundary layers and small-scale turbulent mixing, while the under-resolved flow features are effectively modeled with by subgrid scale closures in appropriately derived coarse-grain models.
  - The above suggests that the homogenization must take into account parameters of space/time discretization on which numerical solutions are to be obtained [7].
  - Homogenization must consistently treat both the transport terms (PDE model) and the constitutive relations (closure laws), which supplement structural / subgrid-scale information [8].

In summary, the "Integration" function requires:

System Modeling: To provide a decomposition/assemble-friendly library of SSCs.

Physics Modeling:

- To provide homogenized coarse-grain models with time/length scales consistent with level of spatio-temporal resolution intended in numerical simulations.
- To provide interfaces between models with different resolution and dimensionality.

[7]Example of achieving such homogenization at a fine scale is Large Eddy Simulation LES with its Sub-Grid Scale SGS model. Work is underway - by Professor S; Shkoller team - on developing a coarse-grain model for compressible Navier-Stokes equations and two-phase flow using the so-called the Lagrangian-Averaged approach. The resulting governing equations contains the third order terms. Numerical solutions to these equations necessitate the use of the higher-order methods.

[8]Modern one-dimensional channel-cross-section-averaged two-fluid model is a case where this consistency is violated. The closure relations are microphysics-based, being accurate at single bubble / droplet level; whereas they are used to model an averaged physics effect in a large control volume.

Code Architecture: Object-oriented.

Numerical Algorithms:

- Dynamic time step control.
- "Gap-tooth" scheme.
- Domain decomposition.
  - Characteristics-based matching interfaces
- Fully implicit nonlinearly coupled solution.
  - Preconditioning.
- Reconstructed Discontinuous Galerkin method.

Data Management: Accommodate multiple levels of resolution.

Verification and Validation: Manufactured solutions for multi-physics problems. Manufactured solutions for multi-scale problems.

1.2.2 Fluid Flow

MODELING and simulation of fluid flow are an area of critical importance for the RISMC code's capability and quality. In fact, the strategy for fluid flow modeling dominates over the whole implementation plan for the RISMC code development, validation and applications.

One apparent reason for this is a well-established view of thermal hydraulics as the major driver for system safety simulations; and this reason continues to be valid and applicable, particularly for a majority of use cases, when the RISMC code gets back to the traditional arena of severe plant transients, DBAs and beyond.

The other - new, life-extension-related, - reason for fluid flow to be the critical element in the RISMC code, already in the early developmental phase, is the role of fluid flow in determining stressors on risk-significant SSCs[9]. This "risk-informed" objective requires a tightly coupled thermal-fluid-structural analysis.

For Plant Aging analysis, fluid flow is primarily single-phase and quasi-steady-state. Two distinct areas of new capability are required to support this analysis.

Area FF-I: Fast-running thermal-fluid flow simulation

- Space-wise, a plant aging analysis must include a large piping network to enable assessment of their integrity and consequences of their failures.
  - This includes plant's primary coolant system, secondary system, safety systems, auxiliary systems, etc.
  - In some areas e.g. straight duct sections, a large node is sufficient, while in some other areas e.g. junctions, finer nodalization is required.
- Time-wise, to support simulations of core neutronics and plant system behavior over a long period of time, over a number of fuel reload cycles, it is required that the fluid flow simulation is fast-running, notably not be limited by the CFL limit as the case in the legacy system thermal hydraulics codes.

[9]The term "risk-significant" used here is meant to go beyond "safety" (e.g., core damage), to include plant operability and plant decision making.

- Even in one-dimensional formulation, the fluid flow system simulation task can thus become computationally demanding.
- High-order accurate schemes in both time and space are critical to achieve computational efficiency.
  - This has been demonstrated for compressible Euler and compressible Navier-Stokes equations models, using Reconstructed Discontinuous Galerkin methods in the RISMC code β_2 version.

Area FF-2: Time- and space-resolved thermal-fluid flow simulation

- Damage growth in structural materials of risk-significant SSCs is driven by the stressors, including those of thermal and mechanical origins. It is therefore important that thermal and mechanical stresses on these SSCs are accurately calculated. This requires high-resolution fluid flow simulations.
  - In a plant normal operation, i.e. quasi steady state, materials in VIS are subject to structural fluctuations and thermal stripping. Persistent nonuniformity in flow patterns (mixing / stratification) in the downcomer and lower plenum can lead to material fatigue.
  - During a plant transient, such as overcooling transients, large temperature gradients imposed on already long irradiated and embrittled structures can accelerate the damage growth. Fluid mixing upstream and in the vicinity of risk-significant SSCs must be resolved appropriately to capture the effect on structures.
- Since Direct Numerical Simulations (DNS) and Large-Eddy Simulations (LES) are not practical for engineering analysis - that involves large dimensions, complex geometry, and long transient processes, - the fluid flow simulation requires subgrid scale (turbulence) models as closure relations for the conservation-equations (PDE) model.
  - This "turbulence modeling" - even limited to regimes and conditions of interest to UC-1 and UC-2 - requires extensive experimental support for development and validation[10].

[10]Research to support validation of coarse-grain models in mixing / stratification flow regimes and transition from force to natural circulation of interest to plant transient analysis is underway at the Utah State University's Professor B. Smith group. Appendix should provide additional details on the research plan at USU.

- Highly desirable Coarse-Grain Models (CGM) need not be able to capture all flow details, but flow patterns and characteristics that influence uncertainty in predicting the stressors of interest for the application (Use Cases). This requires a new approach to fluid flow homogenization [11].

For Plant Transient analysis, fluid flow remains primarily single-phase in a majority of operational and mild transients. This analysis can be well served by capability developed for Areas FF-1 and FF-2 above. In addition, capability to simulate two-phase flow processes need to be considered, as they may emerge in certain set of plant transients, including those caused by degradation-induced failures e.g. pipe rupture.

Area FF-3: Consistent two-phase flow simulation

- (Single-phase and two-phase consistency) Two-phase flow must be modeled in a consistent formulation with single-phase flow, both mathematically and computationally.
  - This is to ensure seamless (preferably, unified) platform for fluid flow over single-phase and two-phase flow regimes.
  - This should also eliminate the issue of phase appearance and disappearance in two-phase flow simulations.
  - This suggests that high-order accurate schemes in time and space (RDG) should be extended to solve the two-phase flow equations.
  - The two-phase flow equations solution must be verifiable and associated numerical uncertainty (discretization errors, solution residuals) be quantified.
  - This requires the two-phase flow equations be hyperbolic (well-posed models) [12].

[11] Research to develop coarse-grain models in compressible Navier-Stokes equations of potential interest to UC-1 and UC-2 is underway at the University of California - Davis's Professor S. Shkoller group. In their work, the Lagrangian-Averaging concept is applied to Navier-Stokes equations, leading to Navier-Stokes-Fourier equations. Appendix should provide additional details on the research plan at UC-Davis.

[12] This can be achieved for two-fluid six-equations model in a variety of ways, including option for interfacial pressure term. For more discussion, see e.g. [DNT03].

    * Mathematical and computational requirements of consistency can be met by a two-phase flow mixture model (three equations with drift flux).

- (Self-consistency) Two-phase flow model should ensure consistency between transport model (PDE), its constitutive relations (closure laws) and PDE's discrete representation.

  - This requires homogenization of closure relations that reflect both internal structures / actions (flow regimes, interface interactions) and time / length scales of the volume for which the closures provide constitution.
  - The so-homogenized two-phase flow models ensure solution convergence, both on PDE part and closure part.
  - As an example and important area for applications, choked flow should be modelled under these constraints, to ensure that the RISMC code can simulate pipe breaks at any location in the piping network, and achieve converged solutions with grid refinement.

    * Predictive "choked flow" (Mach 1) was realized in the RISMC code for single-phase flow with RDG scheme.

- (Developmental consistency) Two-phase flow models used in the RISMC code should be amenable for model improvement, both physically and experimentally[13].

  - Assumptions used to formulate a two-phase flow model and its closure relations, must represent both the state of knowledge and degree of ignorance (lack of knowledge).

    * Such modeling framework uses the current knowledge / data as reference points, while providing "placeholders" for not-existent data;
    * Such placeholders point to data needs, allow sensitivity analysis to evaluate the new data's "value of information", and enable design of new experiments to maximize their value;

[13]This requirement is central to the dynamic concept of knowledge management, and advanced capability such as data assimilation.

    * Newly obtained data can be easily incorporated to reduce uncertainty, and the models are therefore improvable.

  - Static "flow regime maps" as used in the legacy codes are example of models which are built on "frozen" assumptions[14] and hence not amenable for improvement.

    * For inclined pipes, the flow regimes are weighted between those given by the "vertical channel" and "horizontal channel" maps. As a result, new data from inclined channels are not incorporable into the model to reduce its uncertainty.
    * The same remark applies for developing flow conditions and unsteady flow conditions.
    * The static maps create transitions in state space, rendering irrelevant data from experiments which measure time and length scales of the transitions.

  - Examples of improvable models are dynamic flow regimes and dynamic constitutive relations which use internal time- and length scales as parameters. Due to lack of related knowledge, these parameters must be given over an uncertainty range. As new data emerge, they help to narrow the range.

  - Model parameters which have high sensitivity on solutions of two-phase problem must be obtainable or inferable from measurements with high confidence.

    * This is to ensure that such uncertainty in these high-sensitivity parameters are reducible.
    * This also requires a projection of modern diagnostic capabilities, and a plan to develop advanced instruments to perform the necessary measurements.

The above-listed requirements manifest a formidable challenge in the two-phase flow modeling area. Other requirements on fluid flow simulation - for both single- and two-phase cases, such as coupling models of different resolution and dimensionality and coupling fluid flow model with other physics, are discussed as

[14]These modeling assumptions as a rule are derived from best knowledge of experts at the time of model formulation. They thus build on past experience and data but discount the possibility to assimilate new data.

part of ”Multi-Scale, Multi-Physics Integration” function in section 1.2.1.

As the project progresses into a later phase, two-phase boiling flow (and heat transfer) in the core (including highly nonequilibrium regimes as postdryout and rewetting) as well as nonequilibrium two-phase flow in some local areas (e.g. ECCS -RCS junction area) is expected to exhibit itself as a major source of uncertainty [15]. This requires interrogate foundations of two-phase flow modeling, including evaluating

- the merit of seven-equation two-fluid models, including its potential for uncertainty reduction;
- the potential of interfacial area transport equation and the maturity of database to support it, and
- the need to go to multi-field (four-field) models to represent mechanical and thermal nonequilibrium within liquid and vapor phases (e.g. wall bubbles vs vapor core; wall film vs droplets).

This analysis suggests a critical need for the Project to consolidate efforts in two-phase flow and heat transfer areas and establish a modeling framework for two-phase flow and heat transfer that

- allows various sources of uncertainty be represented and evaluated for their significance; this include but are not limited to
  - dynamic flow regimes (in transient and developing two-phase flow);
  - materials / surface / chemistry / irradiation effect on near-wall two-phase flow behavior (bubble nucleation, film rupture);
  - interactions between near-wall / boundary-layer flow and core / bulk flow;
  - multi-dimensional and geometry effects.

[15] Research is underway in the Oregon State University, Ohio State University and MIT to (i) quantify uncertainties in two-phase flow and heat transfer models, and (ii) evaluate capability of modern diagnostics and infrastructure to obtain new data to support model development, validation and uncertainty reduction. Appendix should provide more details on teh planned research in these topical areas.

- serves as platform to integrate results (data, insights, models) from diverse - and currently diverged - experimental and computational research activities in this area, and
- enables obtaining coarse-grain models, which are computationally effective, while conserving key measures of importance to the applications.

This Page is Intentionally Left Blank

Chapter 2

Software Architecture

THIS Chapter describes the architecture of the RISMC β-2 code. We start with outline of the code design concepts (components, interfaces, structure of the vector of unknowns and Jacobian matrices), in Section 2.1, followed by the definition of the code developmental model and brief descriptions of external packages and their use in the RISMC β-2 code. Organization and contents of this chapter is oriented on advanced users, who intend to understand details of the code implementation and to contribute to the libraries of components and interfaces.

2.1 Architecture Design

The RISMC β-2 code must be viewed as a library/suite of utilities for performing nuclear reactor system analysis. The main code driver is implemented as the C++ Class `R7_Driver`. Figure 2.1 outlines the major ingredients of the code design:

★ In the core of the code are the libraries of *components* and *interfaces*, Sections 2.1.1 and 2.1.6.

  – *Components* are designed to represent the components of the modeled reactor system, such as piping, pumps, elbows, T-junctions, downcomer, pressurizer, etc. They encapsulate the mathematical and physical models of the components in their discretized representation. We will discuss component generic design features in section 2.1.1.

  – *Interfaces* are designed to "glue" components together. All components are developed with the capability to be coupled with some other

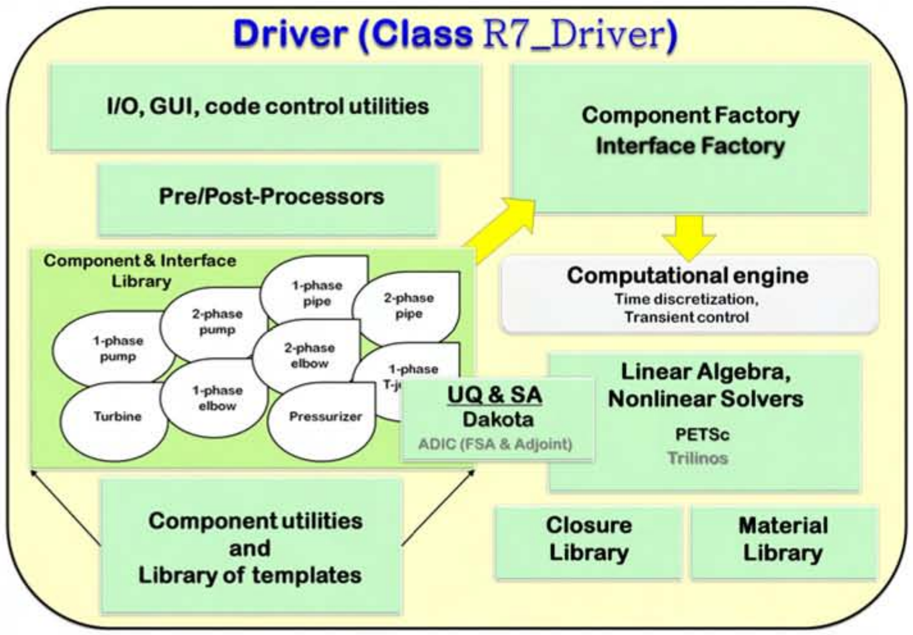

Fig. 2.1 : On design of the RISMC β-2 code.

components. Currently, we utilize the commonly used in computational science "ghost storage" concept. As such, components do not know about existence of other components, and operate presuming that the "ghost storage" is populated in some manner. Interfaces are basically the "traffic cops", which are designed to take certain information from one component and pass it to the "ghost storage" of another, in the appropriate manner. We will discuss interface concepts in Section 2.1.6.

★ *Component and Interface Factories* are C++ utilities for initializing and putting components and interfaces together, in the modeled reactor system.

★ *Computational Engine* is a driving force for transient simulations. It includes initiation of time discretization classes, control of time stepping, including dynamic time step control and handling of abnormal situations (crash/disconvergence), restart control, interaction with and re-initialization of linear and non-linear algebra solvers.

★ *Linear and Non-Linear Algebra*. Currently, the RISMC β-2 code is based on PETSc suite of data structures and routines, Section 2.3.1. PETSc also provides the basic linear and non-linear algebra routines (KSP and SNES, respectively). The details will be discussed in Chapter 3.4.

★ *Material Library*. Another important ingredient of the RISMC β-2 code design is the material library, which includes properties of simple and multi-phase fluids, solids, and neutron properties of fuels. The detail description is given in Section 3.6.

★ *Closure Library*. Mathematical models for most of the components are based on homogenized set of governing equations, requiring physical closure (or constitutive) laws. These closure laws are designed as C++ Classes, and collected in the *Closure Library*.

★ *UQ & SA*. RISMC β-2 code is developed with built-in capabilities for Uncertainty Quantification (UQ) and Sensitivity Analysis (SA). Currently, our main support for UQ & SA is coming from the DAKOTA toolkit, Section 2.3.4.

★ *I/O, GUI, code control utilities*.

★ *Pre- & Post-processors*. Currently, we utilize EXODUS II based data model (Section 2.3.3) for pre- and post-processing. The EXODUS meshes can be generated using SNL's CUBIT software (see Section 2.3.6), and the results can be post-processed (visualized) using LLNL's VISIT toolkit (see Section 2.3.5).

The resulting features for the RISMC β-2 code design are:

- Object-oriented (C++).
- Parallel (MPICH2).
- High-order space discretization (discontinuous finite-element based).
- Fully-implicit, L-stable, high-order time discretization (though, explicit time discretization schemes are also available).
- All-speed capabilities.
- Uncertainty Quantification (UQ) and Sensitivity Analysis (SA) enabled.

- State-of-the-art linear algebra (Krylov-based, Jacobian-free Newton-based algorithms).
- "Born-assessed", with subversion control, extensive verification and documentation, 3D visualization, thorough regression testing.

2.1.1 Components

One of the most important concepts in the RISMC β-2 architectural design is the concept of ***component***. Component is a discrete representation of particular elements/components in the reactor system, such as pipes, elbows, T-junctions, pressurizer, pumps, valves, lower plenum, downcomer, control rods, fuel pins, etc. Snapshot for the hierarchy of currently available component library is shown in Figure 2.2.

Components are based on certain mathematical and physical models. In terms of space representation, we have 0D, 1D, 2D and 3D components, Figure 2.3. Similarly, components can be sorted based on the dominant physics, such as *thermalhydraulics*, *neutronics*, *thermal-structural*, *control system*, etc., see Figure 2.4.

2.1.2 Visualization

Very important concept in designing a component – is its visualization. Regardless of the actual component topology, all components should have their three-dimensional "avatars", which can be used for post-processing. For example, 1D pipes are actually rendered in 3D. In the absence of native 3D mesh data structures, component's "avatar" can be generated using CUBIT, exporting data structure in Exodus II format. An example of a simple multi-component reactor system is shown in Figure 2.5. In this case, 0D pressurizer is rendered as a 3D tank, while 0D control rod system is represented as a solid cylindrical tube inserted into the core.

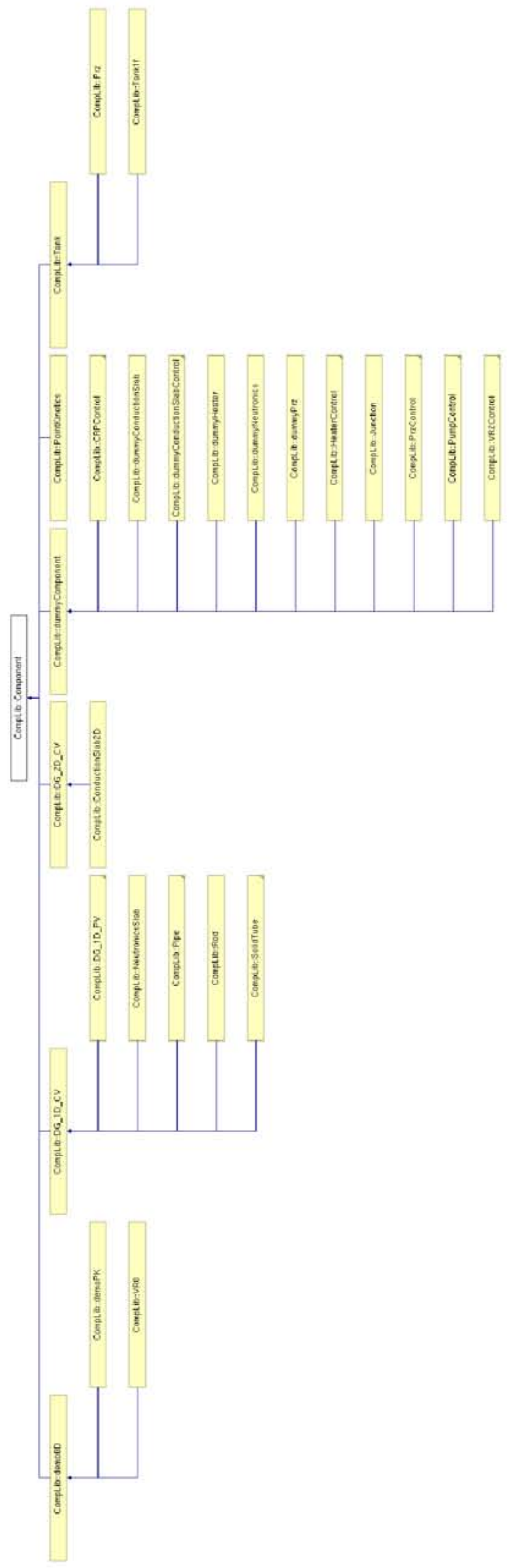

Fig. 2.2 : Hierarchical tree of the Class `Component`.

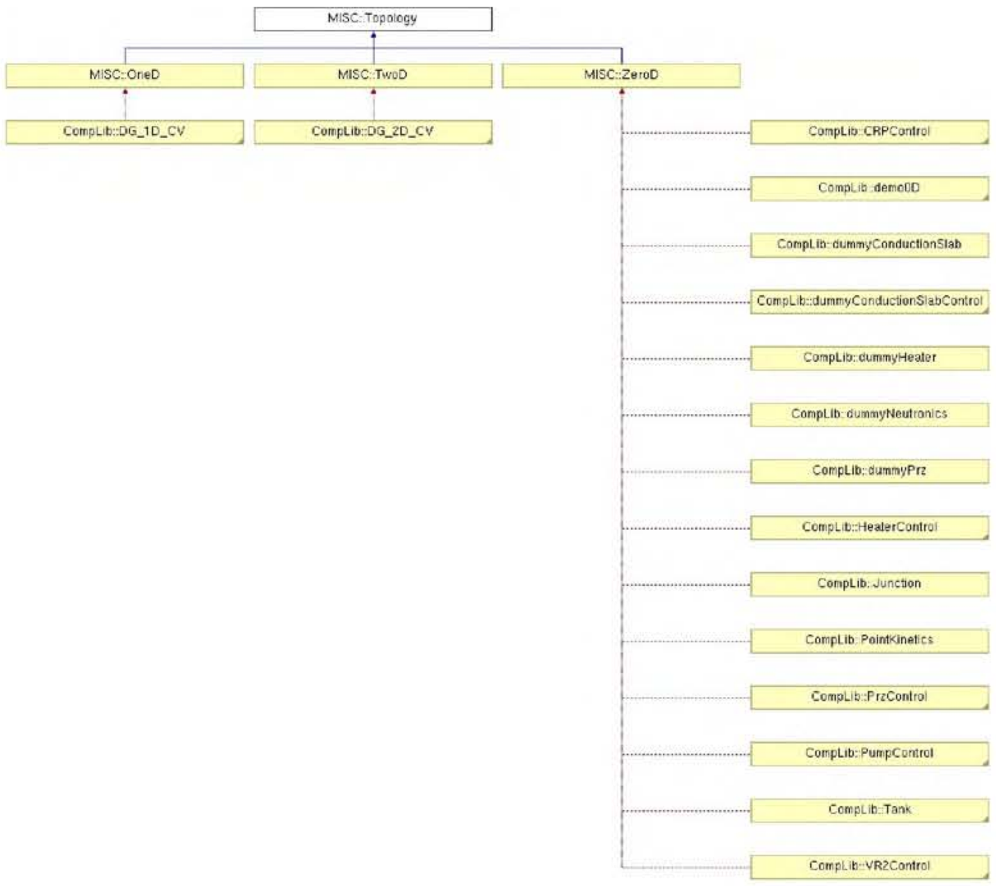

Fig. 2.3 : Hierarchical tree of the Class `Topology`.

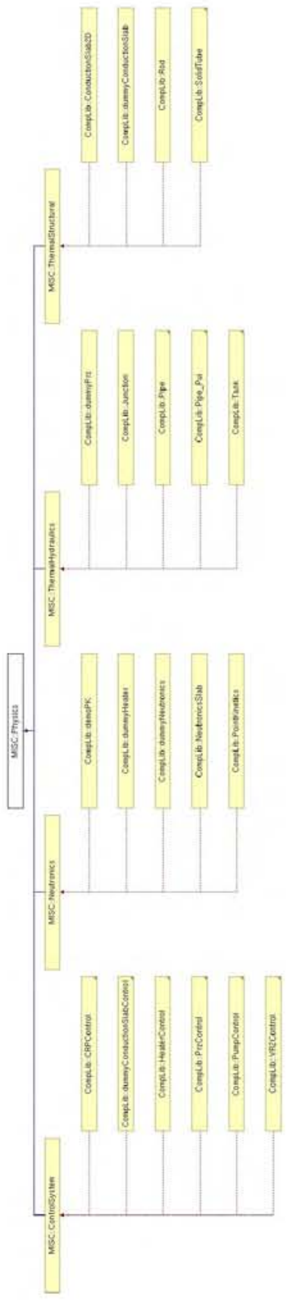

Fig. 2.4 : Hierarchical tree of the Class `Physics`.

2.1.3 Data structures

The main ingredient of the component data structure is the *vector of unknowns*. This is the part which is exposed/represented in the *global solution vector* for the RISMC β-2 non-linear algebra solver. When developing/designing a particular component, the main task is to compute the *vector of residuals*, corresponding to the component's vector of unknowns. Similarly to the vector of unknowns, the local vector of residuals is exposed/represented in the *global vector of residuals*. In addition to these two important component's data concepts, one can have additional data, which are necessary for existence and functioning of a component. Examples of these auxiliary data structures are computational meshes, classes for space discretization, material properties, closures, I/O data streaming, etc.

Another very important concept in the RISMC β-2 design is the *"ghost storage"*. When the component is being developed, it is presumed to be "autonomous", that is it does not know about possible existence of other components. The *"ghost storage"* is a data structure necessary for component's independent functionality. In other words, it is presumed that the *"ghost storage"* is always appropriately filled-in from outside (in fact, this is the task of "interfaces", see Section 2.1.6). The *"ghost storage"* is component-specific, and organized in such a way, so the component's residual vector can be computed seamlessly, without the knowledge of data structures outside of the component.

2.1.4 Orderings

It is important to understand the *ordering* conventions used for the *component's solution/residual vectors* and their relative place in the *global solution vector/residual*. To understand these, one must look at it in the light of the RISMC β-2 global non-linear solver. While details of the used here JFNK solution procedure are revealed in Chapter 3.4, the main piece of the global solution vector non-linear update is captured by the following equation:

$$\vec{\mathcal{X}}^{a+1} = \vec{\mathcal{X}}^{a} + \underbrace{\left(\mathbb{J}^{a}\right)^{-1}\left(-r\vec{e}s\left(\vec{\mathcal{X}}^{a}\right)\right)}_{\text{Linear stage}} \tag{2.1}$$

where $\vec{\mathcal{X}}^{a}$ is a global solution vector at a non-linear iteration a, $r\vec{e}s$ is the corresponding global residual vector, and $\mathbb{J}^{a}$ is a Jacobian matrix. The ordering of unknowns is very important for efficient treatment at the linear stage, corresponding

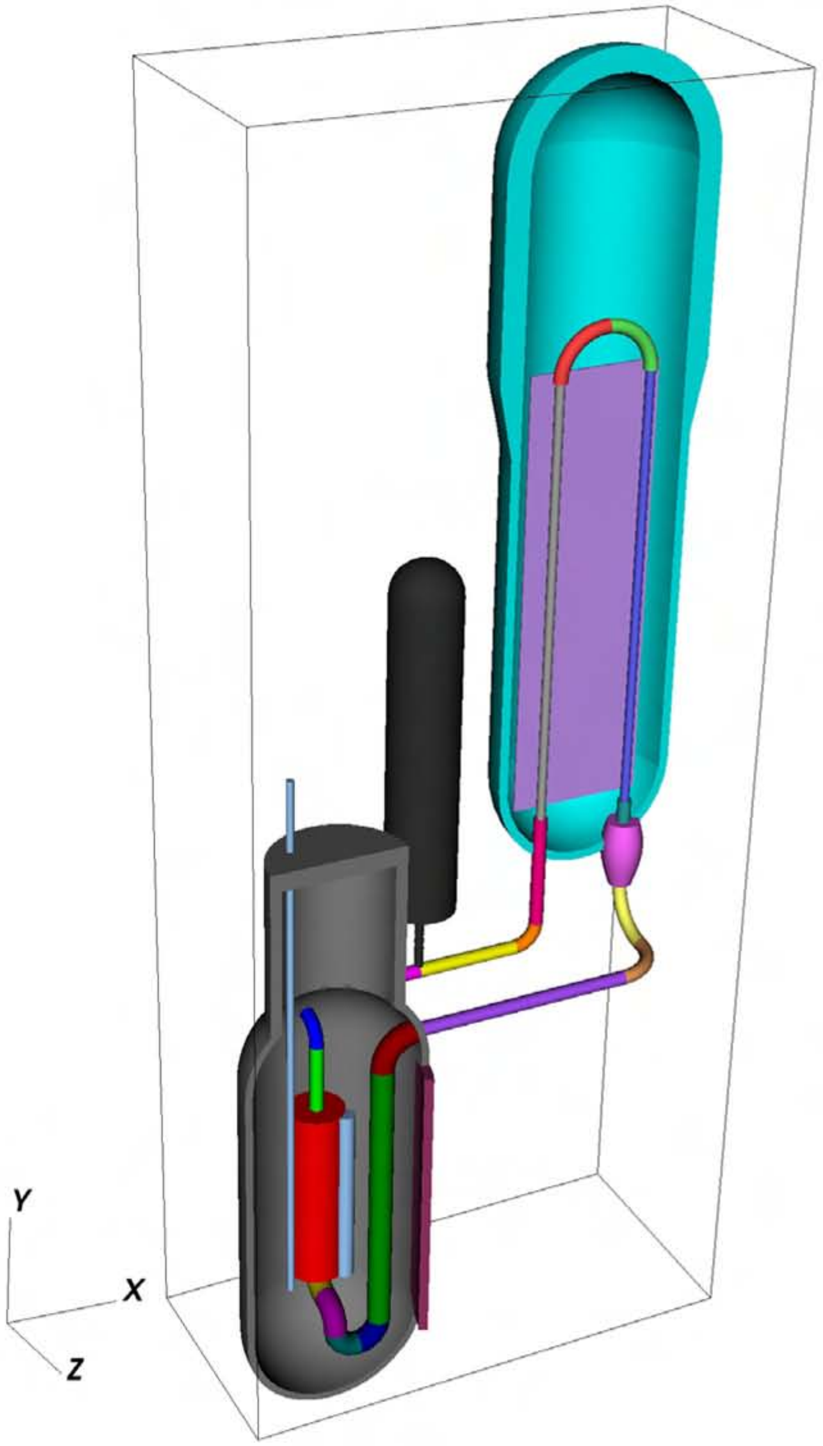

Fig. 2.5 : An example of multi-component system in RISMC β-2 code. 32 components are shown in different colors.

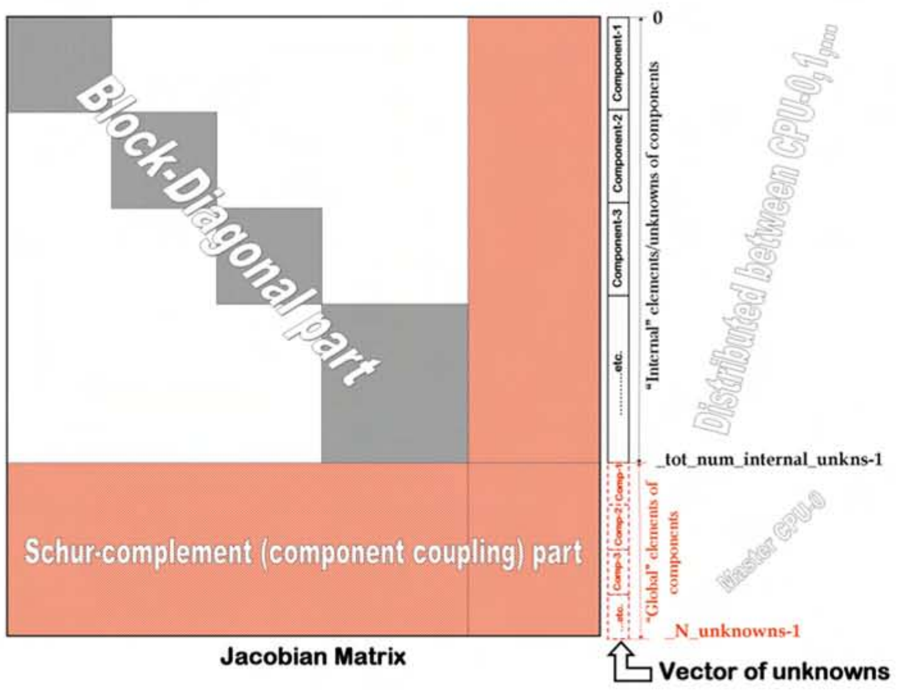
Block-Diagonal part
Schur-complement (component coupling) part
Jacobian Matrix
Component-1
Component-2
Component-3
..........etc.
Comp-1
Comp-2
Comp-3
...etc.
0
"Internal" elements/unknowns of components
Distributed between CPU-0,1,....
_tot_num_internal_unkns-1
"Global" elements of components
Master CPU-0
_N_unknowns-1
Vector of unknowns

Fig. 2.6 : On the structure of the Jacobian matrix and solution vector in RISMC β-2 .

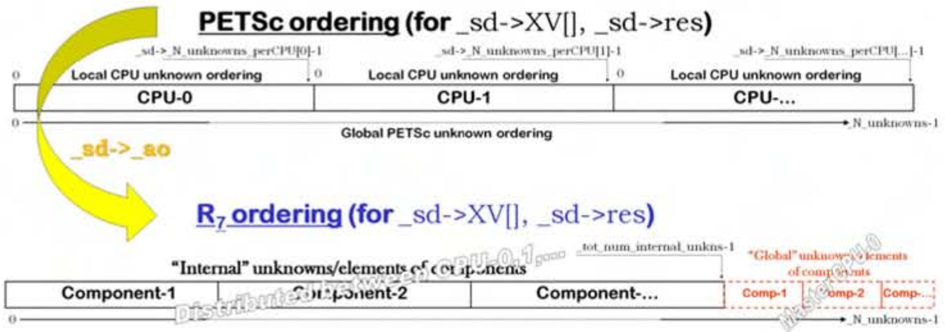
PETSc ordering (for _sd->XV[], _sd->res)
_sd->_N_unknowns_perCPU[0]-1
_sd->_N_unknowns_perCPU[1]-1
_sd->_N_unknowns_perCPU[...]-1
0
Local CPU unknown ordering
CPU-0
CPU-1
CPU-...
Global PETSc unknown ordering
_N_unknowns-1
_sd->_ao
R7 ordering (for _sd->XV[], _sd->res)
"Internal" unknowns/elements of components
Distributed between CPU-0,1,....
_tot_num_internal_unkns-1
"Global" unknowns/elements of components
Master CPU-0
Component-1
Component-2
Component-...
Comp-1
Comp-2
Comp-...

Fig. 2.7 : On the PETSc and RISMC β-2 ordering of unknowns in the RISMC β-2 code.

to the matrix inversion of the Jacobian matrix (generally, treated "matrix-free"). It is necessary to keep the Jacobian matrix structured, so that effective Krylov preconditioning techniques can be applied (see Chapter 3.5).

For that purpose, at the component initiation stage, we tag local (component's) solution vector on "internal" and "global" elements/unknowns. This can be explained in terms of elements of the Jacobian matrix as follows.

Each $_{(ij)}$ element of the Jacobian matrix represents coupling/dependence of the $_{(i)^{\text{th}}}$ unknown to the $_{(j)^{\text{th}}}$ unknown. We define the $_{(i)^{\text{th}}}$ unknown of a component as *internal*, if $\mathbb{J}^{a}_{ij} = 0$ for all js (unknowns) which belong to other components. In other words, the $_{(i)^{\text{th}}}$ unknown is *linearly coupled* to unknowns of its component only, and *linearly independent* of unknowns from outside of the component[1]. Correspondingly, all component's unknowns which linearly-dependent on the solution in another components are called *global*.

Next, we distinguish three kinds of orderings in the RISMC β-2 code:

1. RISMC β-2 ordering,
2. PETSc ordering, and
3. Local component's ordering.

RISMC β-2 ordering and its parallel structure is explained in Figure 2.6. Suppose we have in total `_N_unknowns` unknowns. The RISMC β-2 ordering is started from zero, numbering first *internal* unknowns, on the component-by-component basis, finishing at `_tot_num_internal_unkns`-1. This piece of the global solution vector is parallelized, distributing component data between available CPUs (see **Parallelization strategy** below). Next, we continue ordering, starting from `_tot_num_internal_unkns`, and numbering all *global* unknowns, on the component-by-component basis, finishing at `_N_unknowns`-1. This piece of the global solution vector is shown in red in Figure 2.6, and it is placed on the "master" CPU-0.

As a result of this solution vector ordering and parallelization, the Jacobian matrix is structured as conceptualized in Figure 2.6. There is a *block-diagonal* part, with each block representing the Jacobian sub-matrix, describing linear coupling of unknowns inside each component. In general, each sub-matrix is not necessary full, but it can be structured in some convenient way, facilitating effective

[1]In terms of "ghost storage" defined above, computation of the $_{(i)^{\text{th}}}$ residual elements does not involve data from the "ghost storage".

physics-based preconditioning on the component-by-component basis, see Chapter 3.5. The red part of the Jacobian matrix represents the *component coupling* part. Desirably, the block-diagonal part is significantly larger than the component coupling part. If the physics-based preconditioning strategy were chosen, the component coupling part (lower-right corner block in Figure 2.6) would be directly solved. Thus, the smaller the red lower-right corner block is, the more efficient the general solution strategy ought to be.

PETSc ordering. We base our solution/residual data structures on the PETSc (see Section 2.3.1). PETSc ordering is different from our RISMC β-2 ordering, as explained in Figure 2.7. The PETSc global solution vector is distributed between CPUs. We keep track of both *global PETSc unknown ordering* (from 0 to `_N_unknowns`-1), and *local PETSc unknown ordering* (from 0 to `_sd->_N_unknowns_perCPU[`p`]`-1, for each p^{th} CPU)[2]. Mapping from PETSc to RISMC β-2 ordering and back is implemented using PETSc's "Application ordering" concept, `_sd->ao` of the PETSc type `AO`.

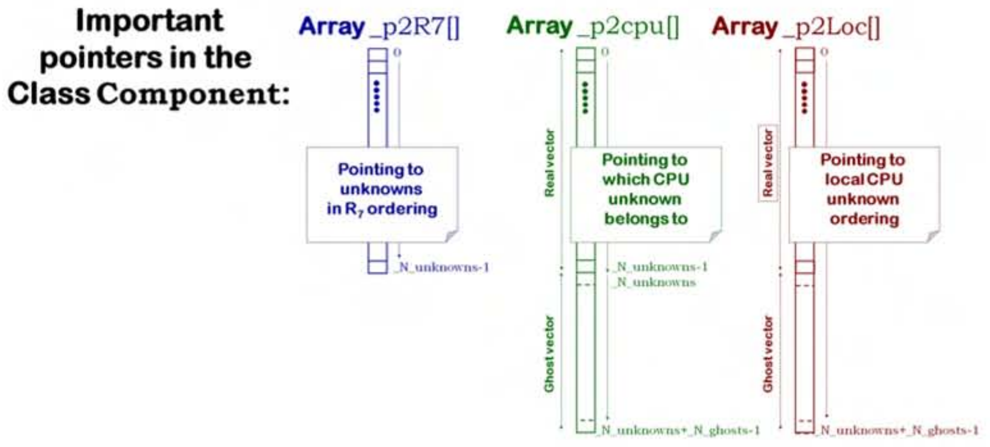
Important pointers in the Class Component:
Array _p2R7[]
Array _p2cpu[]
Array _p2Loc[]
0
Pointing to unknowns in R$_7$ ordering
Pointing to which CPU unknown belongs to
Pointing to local CPU unknown ordering
Real vector
Ghost vector
_N_unknowns-1
_N_unknowns
_N_unknowns+_N_ghosts-1

Fig. 2.8 : On the local component ordering of unknowns in the RISMC β-2 code.

Local component's ordering. When developing a new component, it is very useful to understand the following local component's ordering and mapping arrays, Figure 2.8:

- `_N_unknowns` is a total number of unknowns in the component.

[2]Class `_sd` is of type `R7sdata`, containing all relevant solution data information.

- `_p2R7`[] is the array, pointing from the local component's unknown ordering (set from 0 to `_N_unknowns`-1) to the RISMC β-2 ordering.
- `_p2cpu`[] is the array, pointing to which CPU the unknown belongs to.
- `_p2Loc`[] is the array, pointing to local PETSc unknown ordering (see Figure 2.7).

2.1.5 Parallelization strategy

We use component-based partitioning strategy, that is the CPUs are initially distributed between components, based on the size of the local component's unknown vector, ignoring the actual graph structure of the global Jacobian matrix and solution vector. There is some way the user can affect the partitioning, by specifying the following input parameters (or these can be hard-coded) in the component's auxiliary data structure:

- `_1cpu`: set to `true` if this is a 1-CPU component. This is generally for components which you do not want to partition for the sake of simplicity.
- `_1cpu_no_share`: If 1-CPU component, specify if you want to share CPU with other components (set to `true`). This is generally for components which you do not want to partition, and which are rather "heavy-weight" in terms of computing residual vectors, preconditioning and memory.
- `_live_on_master`: set to `true` if it is 1-CPU component which you want to "live" on the master CPU-0. This is generally for components which are very "light-weight", something like 0D components – very cheap in computation and small memory-wise.

With this, the parallelization is implemented in the following manner:

1. "Master" CPU-0 is reserved for the Schur-complement part of the solution vector.
2. All the rest CPUs are split between components based on the size of the component's solution vector and the following partitioning rules:
   (a) If the component is multi-CPU (its `_1cpu=false`), it is assigned with some CPUs.

(b) If the component is 1-CPU (its `_1cpu=true`) and declared "share" (`_1cpu_no_share=false`), it does not "straddle" over a few CPUs, but it can share CPUs with other 1-CPU components.

(c) If the component is 1-CPU (its `_1cpu=true`) and declared "no share" (`_1cpu_no_share=true`), it "owns" a CPU without share with other components.

Each multi-CPU component on the other hand should have it's own partitioning strategy/algorithm. For example, for 2D/3D components, one can utilize PARMETIS package (Section 2.3.2), for effective data partitioning inside a component.

In general, parallelization of the RISMC β-2 code is cost-effective when there are some very "heavy-weight" (3D) components, and when the block-diagonal part of the Jacobian matrix significantly larger than the component-coupling part, see Figure 2.6.

2.1.6 Interfaces

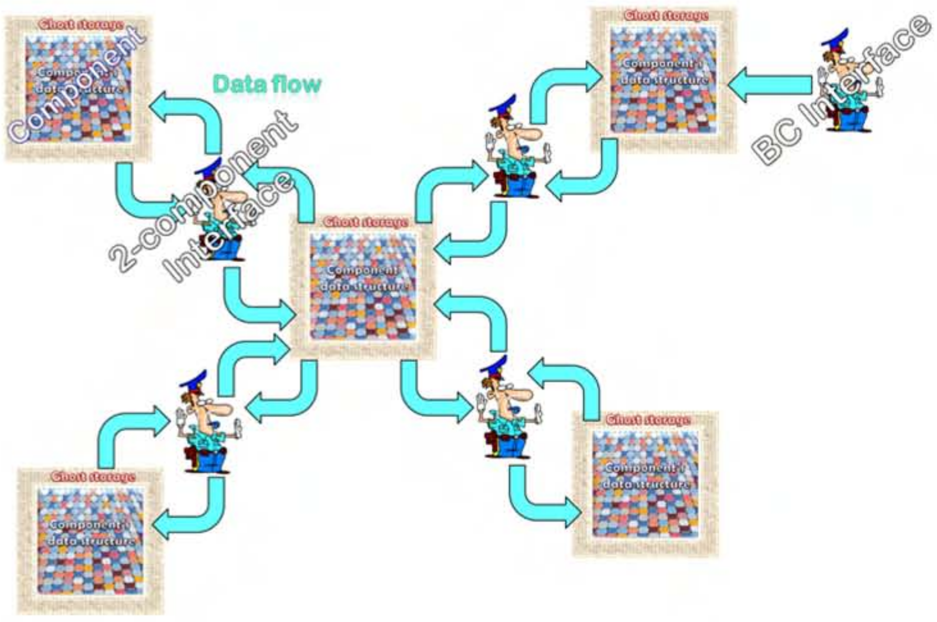

Fig. 2.9 : On the concept of interfaces.

The concept of component's ***interface*** is of great importance for understanding RISMC β-2 architecture. It is closely related to the concept of component's *autonomous functionality*, contributing to the *code development scalability* and general *design flexibility* in building a RISMC β-2 model for specific reactor system.

The foundation of the component library is based on the ability to develop and test (verify/validate) a component independently (autonomously) of other components. For this purpose, we use a concept of "ghost storage". "Ghost storage" is a component-specific data structure, which is not explicitly represented in the global solution/residual vectors, but which is rather created/populated for the purpose of component independent functionality. It is the task of interfaces to properly "populate" component's "ghost storage". Thus, the primary design purpose of interfaces is to be able access data in components and transfer data in appropriate manner from one component data structure to another component "ghost storage", serving a role of "traffic cops", see Figure 2.9.

Each component might have multiple interfaces. An interface might connect several components. In the case of 1-component interface, it serves the role of *boundary condition*.

Interfaces are created using *Interface Factory* – a C++ Class `InterfaceFactory`, designed to automatically identify component connectivity and construct/initialize appropriate interface, adding it to the list of interfaces.

Design of a particular interface is rather difficult task, requiring not only intimate knowledge of underlying mathematics/physics and "ghost storage" for all relevant components, but also the knowledge of how the components interact and integrate into the global solution vectors/procedures. Therefore, only very experienced code developers are assumed the task of creating interfaces.

To understand the function of interfaces, we need to understand how the calls to interfaces are incorporated throughout RISMC β-2 solution algorithms. The building block for RISMC β-2 solution algorithm and the cornerstone for understanding interface machinery is the call for computation of residual vectors. It is invoked from the code's non-linear solvers in implicit algorithms, as well as at the time update of explicit algorithms.

In a nutshell, before any actual computation of the given residual vector $\vec{res}$ using a given solution vector $\vec{\mathcal{X}}$, we call all interfaces in the interface array `_InterfaceArray` (possibly, in a multi-stage manner), and asking to perform "traffic cop" tasks, taking some appropriate data from $\vec{\mathcal{X}}$ and stuffing it into the appropriate "ghost storage". Meta-code for typical interface action is given below:

```
std::vector<Interface *>::iterator i_iterator;

  Apply pre-control functions:

       for(i_iterator=_InterfaceArray->begin();i_iterator!=_InterfaceArray->end();i_iterator++)
       {Interface *interface = *i_iterator; interface->Start_pre_Control(X,0,time);}

       for(i_iterator=_InterfaceArray->begin();i_iterator!=_InterfaceArray->end();i_iterator++)
       {Interface *interface = *i_iterator; interface->Stop_pre_Control(X,0,time);}

  ...Apply component/system control actions...

  Apply post-control functions:

       for(i_iterator=_InterfaceArray->begin();i_iterator!=_InterfaceArray->end();i_iterator++)
       {Interface *interface = *i_iterator; interface->Start_Control(X,0,time);}

       for(i_iterator=_InterfaceArray->begin();i_iterator!=_InterfaceArray->end();i_iterator++)
       {Interface *interface = *i_iterator; interface->Stop_Control(X,0,time);}

  Ask interfaces to populate "ghost storage":

       for(i_iterator=_InterfaceArray->begin();i_iterator!=_InterfaceArray->end();i_iterator++)
       {Interface *interface = *i_iterator; interface->Start_GhstPopulation(X,0,time);}

       for(int stage=0;stage<num_stages;stage++) {

            for(i_iterator=_InterfaceArray->begin();i_iterator!=_InterfaceArray->end();i_iterator++)
            {Interface *interface = *i_iterator; interface->Stage_GhstPopulation(stage,X,0,time);}

       }

       for(i_iterator=_InterfaceArray->begin();i_iterator!=_InterfaceArray->end();i_iterator++)
       {Interface *interface = *i_iterator; interface->Stop_GhstPopulation(X,0,time);}
```

In the above meta-code, interfaces performed three functions: 1) do everything necessary preparing to invoke the control system components; 2) do everything necessary after the control system has been applied; and 3) populate "ghost storage", in `num_stages` manner. Multi-stage ghost population is introduced to increase flexibility of the interfaces to deal with their possibly numerous components.

2.2 Development model

The projected code development model is sketched in Figure 2.10. We visualize 2.5 layers of the code development. At the bottom is the core group of INL researchers (software engineers, computer scientists, nuclear engineers) developing and testing the crucial components of the code infrastructure and user support group. One of the main task is to advance the math and computer science behind the RISMC β-2 computational analysis vehicle, providing component, material and closure templates and specializing component interfaces for the outside user and code development community. The second layer is the outside community

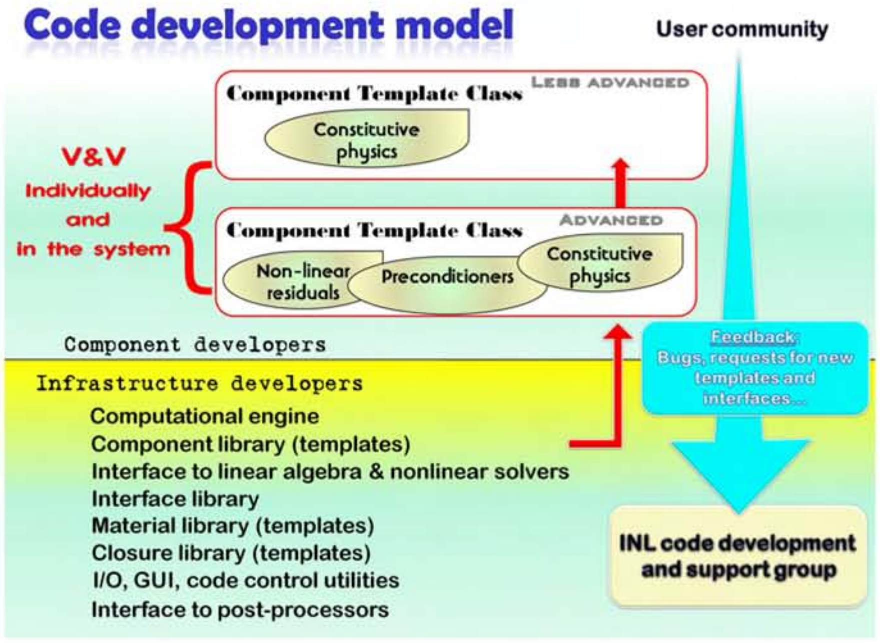
Code development model
User community
Component Template Class
LESS ADVANCED
Constitutive physics
V&V
Individually and in the system
Component Template Class
ADVANCED
Non-linear residuals
Preconditioners
Constitutive physics
Component developers
Feedback: Bugs, requests for new templates and interfaces...
Infrastructure developers
Computational engine
Component library (templates)
Interface to linear algebra & nonlinear solvers
Interface library
Material library (templates)
Closure library (templates)
I/O, GUI, code control utilities
Interface to post-processors
INL code development and support group

Fig. 2.10 : On the RISMC β-2 code development model.

of advanced code users/developers. This group of people with advanced knowledge of the code infrastructure should be able to develop application-specific components out of available component templates. The development activity involves adding closure laws, writing/modifying ("specializing") computation of non-linear residuals, component-wise preconditioning strategy and developing new material properties (if needed), using material templates. This group should provide the major feedback to the infrastructure developing group at the bottom, in terms of bug reports and requests for additional templates and specializing component interfaces. The final "half-layer" of the users/developers is less advanced code testing and application community. In terms of developments, the main contribution of this group is to add application-specific closures, using templates available through the development by the "advanced" users group.

2.3 External packages

There are a number of external packages used in RISMC β-2 , for code functionality, pre/post-processing and documentation. The most important packages are described below.

2.3.1 PETSc

The **P**ortable, **E**xtensible **T**oolkit for **S**cientific **C**omputation (PETSc) [BBG+01, BBE+04] is a suite of data structures and routines, developed at the *Mathematics and Computer Science* Division of Argonne National Laboratory (ANL). PETSc provides the building blocks for the implementation of large-scale application codes written in Fortran, C and C++. It also provides many of the mechanisms needed within parallel application codes, such as parallel matrix and vector assembly routines. The library is organized hierarchically, enabling the level of abstraction that is most appropriate for a particular problem. By using techniques of object-oriented programming, PETSc provides enormous flexibility for its users.

In the RISMC β-2 , PETSc is incorporated on three levels:

1. **Architecture.** We use PETSc as a foundation for our data structures, in particular, for vectors, matrices and distributing parallel data.
2. **Nonlinear solver**. PETSc's nonlinear solver package `SNES` is one of the main options for implementation of JFNK.
3. **Linear solver**. PETSc's linear solver package `KSP` is available for developing preconditioners, on both component- and global-solution levels.

2.3.2 ParMeTiS

ParMeTiS [KSK02, KSK03] is an MPI-based parallel library that implements a variety of algorithms for partitioning and repartitioning unstructured graphs and for computing fill-reducing orderings of sparse matrices. ParMeTiS is particularly suited for parallel numerical simulations involving large unstructured meshes. In this type of computation, ParMeTiS dramatically reduces the time spent in communication by computing mesh decompositions such that the number of interface elements are minimized.

The algorithms in ParMeTiS are based on the multilevel partitioning and fill-reducing ordering algorithms that are implemented in the widely-used serial package MeTiS [KK98]. ParMeTiS provides the following additional functionalities and routines, that are especially suited for parallel computations and large-scale simulations:

- Partition unstructured graphs and meshes.
- Partition graphs that correspond to adaptively refined meshes. Partition graphs for multi-phase and multi-physics simulations.
- Improve the quality of existing partitions.
- Compute fill-reducing orderings for sparse direct factorizations.
- Construct the dual graphs of meshes.

In the RISMC β-2 , ParMeTiS is used through the PETSc (it is a part of the PETSc installation package). ParMeTiS is used to partition both global-component-level (coarse-grain) graphs, as well as for partitioning in-component data structures.

2.3.3 Exodus II

EXODUS II is a model developed in Sandia National Laboratory (SNL) to store and retrieve data for finite element analyses. It is used for preprocessing (problem definition), postprocessing (results visualization), as well as code to code data transfer. An EXODUS II data file is a random access, machine independent, binary file that is written and read via C, C++, or Fortran library routines which comprise the Application Programming Interface (API). In RISMC β-2 , EXODUS II is used for post- and pre-processing computational results.

2.3.4 Dakota

The DAKOTA (**D**esign **A**nalysis **K**it for **O**ptimization and **T**erascale **A**pplications) toolkit provides a flexible, extensible interface between analysis codes and iterative systems analysis methods. It is developed at Sandia National Laboratory (SNL). DAKOTA contains algorithms for optimization with gradient and nongradient-based methods; uncertainty quantification with sampling, reliability, stochastic expansion, and epistemic methods; parameter estimation with nonlinear least squares methods; and sensitivity/variance analysis with design of experiments and parameter study methods. These capabilities may be used on their own or as components within advanced strategies such as hybrid optimization, surrogate-based optimization, mixed integer nonlinear programming, or optimization under uncertainty. By employing object-oriented design to implement abstractions of the key components required for iterative systems analyses, the DAKOTA toolkit provides a flexible and extensible problem-solving environment for design and performance analysis of computational models on high performance computers.

2.3.5 VisIT

The VISIT is a free interactive parallel visualization and graphical analysis tool for viewing scientific data on Unix and PC platforms. It is developed at Lawrence Livermore National Laboratory (LLNL). Users can quickly generate visualizations from their data, animate them through time, manipulate them, and save the resulting images for presentations. VISIT contains a rich set of visualization features so that you can view your data in a variety of ways. It can be used to visualize scalar and vector fields defined on two- and three-dimensional (2D and 3D) structured and unstructured meshes. VISIT was designed to handle very large data set sizes in the terascale range and yet can also handle small data sets in the kilobyte range.

RISMC β-2 code generates outputs for visualization of simulation frames, which are compatible with VISIT. To read RISMC β-2 's visualization files into the VISIT, open the files under

```
/RESULT.X/*.visit
```

2.3.6 Cubit

The CUBIT is a full-featured software toolkit for robust generation of two- and three-dimensional finite element meshes (grids) and geometry preparation. Its main goal is to reduce the time to generate meshes, particularly large hex meshes of complicated, interlocking assemblies. It is a solid-modeler based preprocessor that meshes volumes and surfaces for finite element analysis. Mesh generation algorithms include quadrilateral and triangular paving, 2D and 3D mapping, hex sweeping and multi-sweeping, tet meshing, and various special purpose primitives. CUBIT contains many algorithms for controlling and automating much of the meshing process, such as automatic scheme selection, interval matching, sweep grouping and sweep verification, and also includes state-of-the-art smoothing algorithms.

2.3.7 Doxygen (documentation)

The DOXYGEN is a documentation system for C++, C, Java, Objective-C, Python, IDL (Corba and Microsoft flavors), Fortran, VHDL, PHP, C#, and to some extent D, developed by Dimitri van Heesch [vH97].

In the RISMC β-2 , DOXYGEN is used to generate an online documentation browser (in HTML) and/or an off-line reference manual (in LaTeX) from a set of documented source files. HTML documentation is compiled by executing

```
> doxme
```

in the command shell. To view documentation, open

```
file://PATH/doc/html/index.html
```

in your browser, where `PATH` is the path to your RISMC β-2 installation.

To build full documentation (HTML, LaTeX and including this manual), execute

```
> ./build_doc
```

in the command shell.

2.4 Input and Output

THIS section discusses technical content of Input and Output (I/O) of the RISMC Software.

Input: Input data needed to operate the RISMC code vary with the Use Cases as well as the stage, in which the Use Case is being implemented. As the RISMC code evolves, and the Use Cases enter maturation phases, the input becomes more encompassing. In general, the following categories of input are sought:

- General specification of the system configuration to be analyzed,
  - A set of SSCs that constitute the system;
    - Identify each SSC in the RISMC code's database for the SSC's type, function(s), operational characteristics, reliability and failure modes;
    - Describe each SSC's interface to other SSCs;
  - Assumptions about interfacing SSCs (whose behavior is not modeled / simulated);
- Geometrical characteristics of the modeled SSCs rendered in three-dimensional arrangement;
- Material and fluid characteristics of the modeled SSCs;
- Initial conditions; initialization parameters;
- Nodalization parameters and meshing;
- Numerical integration control parameters (solution algorithms; time step control; convergence criteria);
- Model control parameters (types of models when model selection is required);
- Uncertainty range for a set of scenario, algorithmic and modeling parameters (selected for study);
- Simulation control parameters (including simulation modes e.g. calibration, probabilistic loading; serach);

- Simulation execution monitoring parameters;
- Output parameters and output control parameters.

Output: In practice, specification of the RISMC code output also varies with different Use Cases and User Classes. In general, the following categories of output are sought:

- Input information, including mesh, model / solution choices, etc.;
- Numerical solution control parameters (e.g. time step, residuals, models used);
- System dynamics parameters, e.g. component-average variables;
- Dynamic transient / accident progression trees (identifying times and modes of major events);
- Dynamic 1D profiles (of the plant system);
- Dynamic 3D fiels (e.g. temperature, velocity, strain);
- On-line processing results (e.g. safety margins; correlation coefficients).

The main challenge for the Output is to balance between

- (a) a massive amount of data (particularly, three-dimensional field parameters) generated by transient simulations and combinatorial explosion of scenarios; therefore requiring effective ways to process and collapse the data,
  - by scenario aggregation;
  - by large-time-step output discretization;
  - by coarse-mesh / component-wide characterization (e.g. representing component-average parameters).
- (b) a potential loss of information due to the coarsened / aggregated storage, which is not recoverable in the post-processing step.

Thus, an intelligent Output requires developing and embedding ”on-line” data processing algorithms with the numerical solution that analyze a set of pre-defined control parameters, to identify scenario or process peculiarities (typically, fast-time-scale events) so to enable a ”dynamic printing-time-step control” for data output (printing to files).

This Page is Intentionally Left Blank

Chapter 3

Solution Techniques

3.1 Governing Equations

WE are concerned with the solution of the general class of PDEs (ODEs), expressed in the following form

$$\partial_t \vec{\mathcal{U}} = \underbrace{-\nabla\left(\vec{\mathcal{H}} + \vec{\mathcal{J}}\right) + \vec{\mathcal{R}}}_{\vec{\mathcal{S}}} \tag{3.1}$$

where $\vec{\mathcal{U}}, \vec{\mathcal{H}}, \vec{\mathcal{J}} = -D\nabla\vec{\mathcal{U}}$ and $\vec{\mathcal{R}}$ are the vectors of variables, hyperbolic flux, diffusion flux and local reaction rate, respectively, and D are the diffusion coefficients. These types of PDE describe numerous physical problems, including high- and low-speed fluid dynamics, combustion, radiation/neutron diffusion and transport, thermo-structural mechanics, to name but a few. In *nuclear reactor safety*, these type of equations are relevant for analysis of nuclear reactor transients, involving natural convection of coolant (in passive safety systems) and coupling of fluid dynamics/thermal hydraulics with neutronics, structural mechanics and coolant chemistry. When $\nabla\left(\vec{\mathcal{H}} + \vec{\mathcal{J}}\right) = 0$, eq.(3.1) reduces to ODEs, which can describe many 0D components in RISMC β-2 , such as tanks, point-kinetics neutronics, some elements of control system, etc.

3.1.1 Non-Dimensional Forms

The following basic scaling parameters (pressure, density and temperature and length scale) are used:

$$(P_0, \rho_0, T_0, L)$$

Length scale is set always set to $L = 1$. These scaling parameters are defined as

```
[Scaling]
  PRE_scale = ...
  DEN_scale = ...
  TEM_scale = ...
[]
```

in the file "`INPUT/ConfigR7.inp`".

Scaling can be turn off by setting all the above parameters to 1. Scaling of different physics in RISMC β-2 is defined below.

3.1.2 Thermal-Hydraulics: Acoustic Scaling

In the case of 1D single-phase fluid,

$$\vec{\mathcal{U}} = \begin{bmatrix} \rho \\ m \\ E \end{bmatrix}; \quad \vec{\mathcal{H}} = \begin{bmatrix} m \\ m^2/\rho + P \\ u\,(E+P) \end{bmatrix}; \quad \vec{\mathcal{J}} = \begin{bmatrix} 0 \\ \tau \\ u\tau + \kappa\partial_{\xi} T \end{bmatrix}$$
$$\vec{\mathcal{R}} = \begin{bmatrix} \mathcal{R}_{\rho} \\ \mathcal{R}_{m} + \rho\vec{g}\cdot\vec{\xi} \\ \mathcal{R}_{E} + m\vec{g}\cdot\vec{\xi} \end{bmatrix} \tag{3.2}$$

where ρ, u, $m = \rho u$, P, $E = \rho\left(i + \frac{u^2}{2}\right)$, $\tau = \eta\partial_{\xi} u$, T, η, κ, $\vec{g}$, $\vec{\xi}$ are density, velocity, momentum, pressure, total energy, viscous stress, temperature, dynamic viscosity, thermal conductivity, gravity vector and channel's unit normal vector, respectively. $\mathcal{R}_{\rho}$, $\mathcal{R}_{m}$ and $\mathcal{R}_{E}$ are source (reaction) terms for mass, momentum and total energy, correspondingly. Internal energy is defined as $i = i_{\text{ref}} + C_v\,(T - T_{\text{ref}})$, where C_v, i_{ref} and T_{ref} are specific heat and reference internal energy and temperature, respectively.

For this set of equations, state variables are scaled as[1]:

$$\begin{gathered}
\rho = \tilde{\rho}\rho_0, \quad P = \tilde{P}P_0, \quad u = \tilde{u}\sqrt{\frac{P_0}{\rho_0}}, \qquad T = \tilde{T}T_0 \\
E = \tilde{E}P_0, \quad i = \tilde{i}\frac{P_0}{\rho_0}, \quad C_v = \tilde{C}_v\frac{P_0}{\rho_0 T_0}, \quad \kappa = \tilde{\kappa}\frac{LP_0\sqrt{\frac{P_0}{\rho_0}}}{T_0} = \frac{\tilde{\kappa}P_0\sqrt{\frac{P_0}{\rho_0}}}{T_0} \\
\eta = \tilde{\eta}L\rho_0\sqrt{\frac{P_0}{\rho_0}} = \tilde{\eta}\rho_0\sqrt{\frac{P_0}{\rho_0}}, \qquad g = \tilde{g}\frac{P_0}{L\rho_0} = \frac{\tilde{g}P_0}{\rho_0} \\
L = \tilde{L}, \quad t = \tilde{t}\frac{L}{\sqrt{\frac{P_0}{\rho_0}}} = \frac{\tilde{t}}{\sqrt{\frac{P_0}{\rho_0}}}
\end{gathered} \tag{3.3}$$

[1]Note that $L = 1$.

3.2 Time Discretization

THIS chapter describes time discretization of eq.(3.1). Explicit time schemes are summarized in Section 3.2.1. Implicit schemes are described in Section 3.2.2. The hierarchical tree of the time discretization Class `TDiscr` is shown in Figure 3.1.

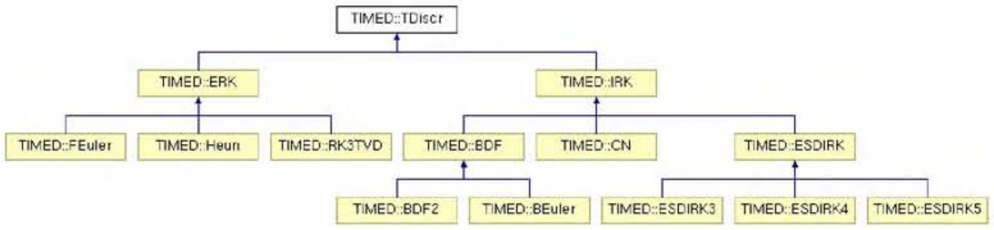

Fig. 3.1 : Hierarchical tree of the Class `TDiscr`.

3.2.1 Explicit schemes (Class `ERK`)

For general time integration of the system eq.(3.1), a number of *Strong-Stability-Preserving (SSP)* explicit time discretization methods are available [Got05]. The *Total Variation Diminishing (TVD) Runge-Kutta* methods of Shu and Osher [SO89] (a subclass of SSP) are particularly suited for this purpose. In addition to the simplicity of the Runge-Kutta methods, they are specially designed for time integration of hyperbolic conservation laws in a way that does not create spurious oscillation in the solution. The explicit Runge-Kutta schemes implemented in R$_7$ code are given by

First-order Forward Euler (Class `FEuler`):

$$\vec{\mathcal{U}}^{(n+1)} = \vec{\mathcal{U}}^{(n)} + \Delta t \cdot \vec{\mathcal{S}}\left(\vec{\mathcal{U}}^{(n)}\right) \tag{3.4}$$

Second-order Heun (Class `Heun`):

$$\left\{ \begin{array}{l} \vec{\mathcal{U}}^{(1)} = \vec{\mathcal{U}}^{(n)} + \Delta t \cdot \vec{\mathcal{S}}\left(\vec{\mathcal{U}}^{(n)}\right) \\ \vec{\mathcal{U}}^{(n+1)} = \frac{1}{2}\vec{\mathcal{U}}^{(n)} + \frac{1}{2}\left(\vec{\mathcal{U}}^{(1)} + \Delta t \cdot \vec{\mathcal{S}}\left(\vec{\mathcal{U}}^{(1)}\right)\right) \end{array} \right. \left| \begin{array}{l} t^{(1)} = t^{(n+1)} \\ \hline \end{array} \right. \tag{3.5}$$

Third-order RK-TVD (Class RK3TVD):

$$\left\{\begin{array}{l} \vec{\mathcal{U}}^{(1)} = \vec{\mathcal{U}}^{(n)} + \Delta t \cdot \vec{\mathcal{S}}\left(\vec{\mathcal{U}}^{(n)}\right) \\ \vec{\mathcal{U}}^{(2)} = \frac{3}{4}\vec{\mathcal{U}}^{(n)} + \frac{1}{4}\left(\vec{\mathcal{U}}^{(1)} + \Delta t \cdot \vec{\mathcal{S}}\left(\vec{\mathcal{U}}^{(1)}\right)\right) \\ \vec{\mathcal{U}}^{(n+1)} = \frac{1}{3}\vec{\mathcal{U}}^{(n)} + \frac{2}{3}\left(\vec{\mathcal{U}}^{(2)} + \Delta t \cdot \vec{\mathcal{S}}\left(\vec{\mathcal{U}}^{(2)}\right)\right) \end{array}\right. \left|\begin{array}{l} t^{(1)} = t^{(n+1)} \\ \hline t^{(2)} = t^{(n+\frac{1}{2})} \\ \hline \\ \end{array}\right. \tag{3.6}$$

where Δt is a time step.

The third order *TVD* method is generally recommended, since it has the greatest accuracy and the largest time step stability region of the *TVD* schemes. Due to its large stability region (which includes a segment of purely imaginary linear growth rates), for a sufficiently small time step, it is guaranteed to be linearly stable for the wide range of problems. In contrast, the first and the second order methods both require some spatial diffusion terms in order to be stable. Without that, no matter how small the time step is, they may be mildly unstable. For this reason, they should not be used unless there is substantial spatial diffusion in the problem [FMDO98].

3.2.2 Implicit schemes (Class IRK)

An implicit time discretization of Eq.(3.1) can be written as

$$\begin{array}{rcl} \vec{\mathcal{U}}^{[k]} & = & \vec{\mathcal{U}}^{[n]} + \Delta t \sum\limits_{r=1}^{k} \mathsf{a}_{kr} \vec{\mathcal{S}}\left(\vec{\mathcal{U}}^{[r]}\right), \quad k = 1, ..., s-1 \\ \vec{\mathcal{U}}^{[n+1]} & = & \alpha\vec{\mathcal{U}}^{[n-1]} + \beta\vec{\mathcal{U}}^{[n]} + \Delta t \left(\mathsf{b}_0 \vec{\mathcal{S}}\left(\vec{\mathcal{U}}^{[n]}\right) + \sum\limits_{r=1}^{s} \mathsf{b}_r \vec{\mathcal{S}}\left(\vec{\mathcal{U}}^{[r]}\right)\right) \end{array} \tag{3.7}$$

where s is the total number of implicit Runge-Kutta (IRK) stages, while a_{kr} and b_r are the stage and the main scheme weights, respectively. We dropped all sub/superscripts associated with spatial discretization, for brevity. Superscripts "$[\times]$" denote the stages of IRK iteration.

Class BEuler. In the case of $\alpha = 0$, $\beta = 1$, $s = 1$, $\mathsf{b}_0 = 0$ and $\mathsf{b}_1 = 1$, Eq.(3.7) reduces to the first-order *Backward Euler* (BE_1) discretization.

Class BDF2. In the case of $\alpha = -\frac{\Delta t^2}{\Delta t_{n-1}\left(2\Delta t + \Delta t_{n-1}\right)}$, $\beta = \frac{\Delta t}{\Delta t_{n-1}} \frac{\Delta t + \Delta t_{n-1}}{2\Delta t + \Delta t_{n-1}}$, $s = 1$, $\mathsf{b}_0 = 0$ and $\mathsf{b}_1 = 1$, Eq.(3.7) reduces to the second-order *Backward Difference* (BDF_2) discretization.

Class CN. In the case of $\alpha = 0$, $\beta = 1$, $s = 1$, $\mathsf{b}_0 = \frac{1}{2}$ and $\mathsf{b}_1 = \frac{1}{2}$, it is the second-order *Crank-Nicholson* (CN$_2$) scheme.

Class ESDIRK$_{3,4,5}$. A family of high-order IRK schemes recently developed by Carpenter et al. [BCVK02, CKB+05] is particularly useful for multiphysics problems, since these schemes not only do not amplify any left-half-plane-(LHP)-scaled eigenvalues (*A-stability*), but also provide a complete damping of all eigenvalues including those at the limit $||z \to \infty||$ (*L-stability*). These IRK schemes are prescribed by $\mathsf{b}_0 = 0$ and the Butcher tableau of the following form:

$$
\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & \dots & 0 \\
\mathsf{c}_2 & \mathsf{a}_{21} & \gamma & 0 & 0 & \dots & 0 \\
\mathsf{c}_3 & \mathsf{a}_{31} & \mathsf{a}_{32} & \gamma & 0 & \dots & 0 \\
\dots & & & \dots & & & \\
\mathsf{c}_{s-1} & \mathsf{a}_{(s-1)1} & \mathsf{a}_{(s-1)2} & \dots & \dots & \gamma & 0 \\
1 & \mathsf{b}_1 & \mathsf{b}_2 & \mathsf{b}_3 & \dots & \mathsf{b}_{(s-1)} & \gamma \\
\hline
 & \mathsf{b}_1 & \mathsf{b}_2 & \mathsf{b}_3 & \dots & \mathsf{b}_{(s-1)} & \gamma \\
\hline
 & \hat{\mathsf{b}}_1 & \hat{\mathsf{b}}_2 & \hat{\mathsf{b}}_3 & \dots & \hat{\mathsf{b}}_{(s-1)} & \hat{\mathsf{b}}_{(s)}
\end{array}
\tag{3.8}
$$

where c_r denotes the point in time of the r^{th}-stage, $t^{[n]} + \mathsf{c}_r \Delta t$. Note that the first stage is explicit, and the diagonal elements for all stages $r > 1$ are the same, $\mathsf{a}_{rr} = \gamma$, which is why this family is called *"Explicit, Singly Diagonal Implicit Runge-Kutta" (ESDIRK)* in the literature. Note that the p^{th}-order ESDIRK_p schemes allow to compute $(p-1)^{\text{th}}$-order solution, as

$$
\vec{\mathcal{U}}^{[n+1]} = \vec{\mathcal{U}}^{[n]} + \Delta t \sum_{r=1}^{s} \hat{\mathsf{b}}_r \vec{\mathcal{S}} \left(\vec{\mathcal{U}}^{[r]} \right) \tag{3.9}
$$

The coefficients for $\mathsf{ESDIRK}_{3,4,5}$ are summarized below.

ESDIRK$_3$

$$
\begin{array}{c|cccc}
0 & 0 & 0 & 0 & 0 \\
\frac{1767732205903}{2027836641118} & \frac{1767732205903}{4055673282236} & \frac{1767732205903}{4055673282236} & 0 & 0 \\
\frac{3}{5} & \frac{2746238789719}{10658868560708} & -\frac{640167445237}{6845629431997} & \frac{1767732205903}{4055673282236} & 0 \\
1 & \frac{1471266399579}{7840856788654} & -\frac{4482444167858}{7529755066697} & \frac{11266239266428}{11593286722821} & \frac{1767732205903}{4055673282236} \\
\hline
\mathsf{b}_r & \frac{1471266399579}{7840856788654} & -\frac{4482444167858}{7529755066697} & \frac{11266239266428}{11593286722821} & \frac{1767732205903}{4055673282236} \\
\hline
\hat{\mathsf{b}}_r & \frac{2756265671327}{12835298489170} & -\frac{10771552573575}{22201958757719} & \frac{9247589265047}{10645013368117} & \frac{2193209047091}{5459859503100}
\end{array}
\tag{3.10}
$$

ESDIRK$_4$

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \\
\frac{83}{250} & \frac{8611}{62500} & -\frac{1743}{31250} & \frac{1}{4} & 0 & 0 & 0 \\
\frac{31}{50} & \frac{5012029}{34652500} & -\frac{654441}{2922500} & \frac{174375}{388108} & \frac{1}{4} & 0 & 0 \\
\frac{17}{20} & \frac{15267082809}{155376265600} & -\frac{71443401}{120774400} & \frac{730878875}{902184768} & \frac{2285395}{8070912} & \frac{1}{4} & 0 \\
1 & \frac{82889}{524892} & 0 & \frac{15625}{83664} & \frac{69875}{102672} & -\frac{2260}{8211} & \frac{1}{4} \\
\hline
\mathsf{b}_r & \frac{82889}{524892} & 0 & \frac{15625}{83664} & \frac{69875}{102672} & -\frac{2260}{8211} & \frac{1}{4} \\
\hline
\hat{\mathsf{b}}_r & \frac{4586570599}{29645900160} & 0 & \frac{178811875}{945068544} & \frac{814220225}{1159782912} & -\frac{3700637}{11593932} & \frac{61727}{225920}
\end{array} \tag{3.11}$$

ESDIRK$_5$

$$
\begin{array}{c|ccccc}
0 & 0 & 0 & 0 & 0 & . \\
\frac{41}{100} & \frac{41}{200} & \frac{41}{200} & 0 & 0 & . \\
\frac{2935347310677}{11292855782101} & \frac{41}{400} & \frac{-567603406766}{11931857230679} & \frac{41}{200} & 0 & . \\
\frac{1426016391358}{7196633302097} & \frac{683785636431}{9252920307686} & 0 & \frac{-110385047103}{1367015193373} & \frac{41}{200} & . \\
\frac{92}{100} & \frac{3016520224154}{10081342136671} & 0 & \frac{30586259806659}{12414158314087} & \frac{-22760509404356}{11113319521817} & \mathbb{B} \\
\frac{24}{100} & \frac{218866479029}{1489978393911} & 0 & \frac{638256894668}{5436446318841} & \frac{-1179710474555}{5321154724896} & . \\
\frac{3}{5} & \frac{1020004230633}{5715676835656} & 0 & \frac{25762820946817}{25263940353407} & -\frac{2161375909145}{9755907335909} & . \\
1 & \frac{-872700587467}{9133579230613} & 0 & 0 & \frac{22348218063261}{9555858737531} & . \\
\hline
\mathsf{b}_r & \frac{-872700587467}{9133579230613} & 0 & 0 & \frac{22348218063261}{9555858737531} & .
\end{array}
\tag{3.12}
$$

$$
\mathbb{B} = \left[\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\frac{41}{200} & 0 & 0 & 0 \\
\frac{-60928119172}{8023461067671} & \frac{41}{200} & 0 & 0 \\
\frac{-211217309593}{5846859502534} & \frac{-4269925059573}{7827059040749} & \frac{41}{200} & 0 \\
\frac{-1143369518992}{8141816002931} & \frac{-39379526789629}{19018526304540} & \frac{32727382324388}{42900044865799} & \frac{41}{200} \\
\hline
\frac{-1143369518992}{8141816002931} & \frac{-39379526789629}{19018526304540} & \frac{32727382324388}{42900044865799} & \frac{41}{200}
\end{array}\right]
$$

3.2.3 Input options

Time discretization is defined in the file

INPUT/ConfigR7.inp"

The following options are currently available:

-
  ```
  [TimeDiscr]
    time_discr = ...
  []
  ```
  , time discretization scheme:

  (0) Backward Euler, class `BEuler`.

  (1) Crank-Nicholson, class `CN`.

  (2) BDF2, class `BDF2`.

  (3) ESDIRK3, class `ESDIRK3`.

  (4) ESDIRK4, class `ESDIRK4`.

  (11) Forward Euler, class `FEuler`.

  (12) Heun, class `Heun`.

  (13) RK3-TVD, class `RK3TVD`.

-
  ```
  [TimeDiscr]
    steady_state = ...
  []
  ```
  , option to switch to steady-state:

  (0) Always transient (Default).

  (1) At the last step, switch to steady-state.

  (2) Always steady-state.

  In computations, switching to steady-state means ignoring transient terms in residual evaluations[2].

-
  ```
  [TimeDiscr]
    timeBEG = ...
    timeEND = ...
  []
  ```
  : starting and ending time of the transient.

[2]If a component solves for $\frac{d\vec{\mathcal{U}}}{dt} = \vec{\mathcal{S}}$, and $\vec{\mathcal{S}}$ is not a function of $\vec{\mathcal{U}}$, linear algebra will break.

-
```
[TimeDiscr]
  NtstepsMAX = ...
[]
```
: maximum number of time steps.

-
```
[TimeDiscr]
  dt_strategy = ...
[]
```
, strategy (name of the Class) for time stepping, see Section 3.2.4.

-
```
[TimeDiscr]
  check_steady_state = true/false
[]
```
: If `true`, check whether the steady-state is achieved (activated only for implicit time discretizations, Classes of the `IRK` family).

-
```
[TimeDiscr]
  check_steady_stateL1 = true/false
  check_steady_stateL2 = true/false
  check_steady_stateLI = true/false
[]
```
: If `check_steady_state`=$true$, use $\mathcal{L}_1$-, $\mathcal{L}_2$- or $\mathcal{L}_\infty$-norms of steady-state residuals for check of steady-state.

  At the end of each time step, steady-state residuals for each solution variable are scaled by maxima of these residuals (traced during a whole transient). If the scaled $\mathcal{L}_\times < \mathrm{tol}_{ss}$ for the last n_{ss} time steps, the simulation is declared to reach steady-state.

-
```
[TimeDiscr]
  steady_state_tol =→ tol_ss
[]
```
: Tolerance for steady-state.

-
```
[TimeDiscr]
  num_steps_to_satisfy =→ n_ss
[]
```
: The number of time steps to satisfy steady-state conditions, before stopping.

3.2.4 Dynamic time step strategies

There are a number of time stepping algorithms available in the RISMC β-2 , all summarized in Figure 3.2. A particular time stepping strategy is chosen by setting

```
[TimeDiscr]
  dt_strategy = ...
[]
```

in the file

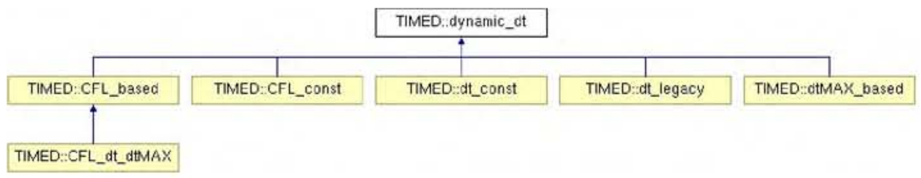

Fig. 3.2 : Hierarchical tree of the Class `dynamic_dt`.

`INPUT/ConfigR7.inp`"

Input settings for each of the time stepping algorithms are specified in file

`INPUT/dt.inp`"

Dynamic time estimate

Dynamic time can be estimated using asymptotic analysis of the time update. Expanding to the third-order:

$$\underbrace{\mathcal{U}^{(k)} - \mathcal{U}^{(m)}}_{\text{Total change}} = \underbrace{\left(\overbrace{t^{(m)} - t^{(k)}}^{\Delta t^{(m)}}\right) \partial_t \mathcal{U}^{(m)}}_{\text{Gradient, } \mathfrak{G}} + \underbrace{+\frac{\left(t^{(m)} - t^{(k)}\right)^2}{2} \partial_{tt} \mathcal{U}^{(m)}}_{\text{Curvature, } \mathfrak{C}} + O\left(\Delta t^4\right) \tag{3.13}$$

at $t^{(m=n+1)}$, the time gradient and curvature terms are

$$\begin{aligned} \mathfrak{G} &\equiv \Delta t^{(n+1)} \partial_t \mathcal{U}^{(n+1)} &&= \frac{\alpha^{(n+1)}\left(2+\alpha^{(n+1)}\right)\Delta^+ - \Delta^-}{\alpha^{(n+1)}\left(1+\alpha^{(n+1)}\right)} \\ \mathfrak{C} &\equiv \frac{\left(\Delta t^{(n+1)}\right)^2}{2} \partial_{tt} \mathcal{U}^{(n+1)} &&= \frac{\alpha^{(n+1)}\Delta^+ - \Delta^-}{\alpha^{(n+1)}\left(1+\alpha^{(n+1)}\right)} \end{aligned} \tag{3.14}$$

where $\alpha^{(m)} \equiv \frac{\Delta t^{(m)}}{\Delta t^{(m-1)}}$, $\Delta^+ = \left(\mathcal{U}^{(n+1)} - \mathcal{U}^{(n)}\right)$ and $\Delta^- = \left(\mathcal{U}^{(n)} - \mathcal{U}^{(n-1)}\right)$.

The next time update of the variable can be estimated as

$$\Delta^{++} = \alpha^{(n+2)}\mathfrak{G} + \left(\alpha^{(n+2)}\right)^2 \mathfrak{C} \tag{3.15}$$

Scaling eq.(3.15) with

$$\mathcal{U}_0 = \begin{cases} \bar{\mathcal{U}}_0 = \frac{1}{3}\left(\left|\mathcal{U}^{(n+1)}\right|, \left|\mathcal{U}^{(n)}\right|, \left|\mathcal{U}^{(n-1)}\right|\right) & \text{if } \bar{\mathcal{U}}_0 > \varepsilon_{\text{scaling}} \\ 1 & \text{otherwise} \end{cases} \tag{3.16}$$

as

$$\sigma = \frac{\Delta^{++}}{\mathcal{U}_0} = \alpha^{(n+2)}\frac{\mathfrak{G}}{\mathcal{U}_0} + \left(\alpha^{(n+2)}\right)^2 \frac{\mathfrak{C}}{\mathcal{U}_0} \tag{3.17}$$

eq.(3.17) gives an estimate of $\alpha^{(n+2)}$ which results in the new time change of variable $\mathcal{U}$ which is less or equal to the specified "safety" parameter σ. Solving eq.(3.17) for $\alpha^{(n+2)}$ gives dynamic time estimate:

$$\Delta t_{\text{dyn}} = \alpha^{(n+2)}\Delta t^{(n+1)} \tag{3.18}$$

We actually use the following algorithm:

$$\begin{array}{lll} \text{if } \left|\frac{\mathfrak{G}}{\mathcal{U}_0}\right| < \varepsilon_\Delta & \text{then} & \alpha^{(n+2)} = \beta \\ \text{else if } \left|\frac{\mathfrak{C}}{\mathcal{U}_0}\right| < \varepsilon_\Delta & \text{then} & \alpha^{(n+2)} = \left|\frac{\sigma\mathcal{U}_0}{\mathfrak{G}}\right| \\ \text{otherwise} & & \text{The smallest positive root of eq.(3.17)} \end{array} \tag{3.19}$$

The above-described scheme for dynamic time estimate is defined by setting

```
[dynamic_time]
  sch_dyn_time = 2
[]
```

in the file

INPUT/dt.inp"

The other necessary input parameters defined in this file are:

-
```
[dynamic_time]
  dyn_tm_sfty = ...
[]
```
: safety factor σ in eq.(3.19).

-
  ```
  [dynamic_time]
    dt_ratioUP = ...
  []
  ```
  : safety factor β in eq.(3.19).

-
  ```
  [dynamic_time]
    epsSC_dyn_tm = ...
  []
  ```
  : threshold for scaling in dynamic time estimate, $\varepsilon_{\text{scaling}}$ in eq.(3.16).

-
  ```
  [dynamic_time]
    epsDR_dyn_tm = ...
  []
  ```
  : threshold for mode of derivatives in dynamic time estimate, ε_{Δ} in eq.(3.19).

Class `dt_const`

Constant time stepping is defined by setting

```
[TimeDiscr]
  dt_strategy = dt_const
[]
```

in the file

INPUT/ConfigR7.inp"

Time step is specified in the file

INPUT/dt.inp"

as

```
[constant_dt]
  dt0 = ...
[]
```

Class `CFL_const`

Using the fastest normal-mode time scale, one can dynamically change Δt of the simulation as

$$\Delta t_{\text{CFL}} = \text{CFL} \cdot \tau_{\text{fastest}} \tag{3.20}$$

where the CFL is an input Courant limit, and the normal-mode times scales are discussed in the description of each component.

This time stepping is defined by setting

```
[TimeDiscr]
  dt_strategy = CFL_const
[]
```

in the file

INPUT/ConfigR7.inp”

CFL is specified in the file

INPUT/dt.inp”

as

```
[constant_CFL]
  CFL0 = ...
[]
```

Class `CFL_based`

The new time step can be chosen as

$$\Delta t^{(n+2)}_{\mathrm{CFL\_based}} = \begin{cases} \frac{\Delta t_{\mathrm{CFL}_{\mathrm{min}}}}{2} & \text{if } N_{\text{time step}} < N_{\mathrm{Init}} \\ \min\left(\max\left(\min\left(\Delta t_{\mathrm{dyn}}, \Delta t_{\mathrm{CFL}_{\mathrm{max}}}\right), \Delta t_{\mathrm{CFL}_{\mathrm{min}}}\right), \beta\Delta t^{(n+1)}\right) & \text{otherwise} \end{cases} \tag{3.21}$$

where Δt_{dyn} is computed from eqs.(3.18) and (3.19), while $\Delta t_{\mathrm{CFL}_{\mathrm{min}}}$ and $\Delta t_{\mathrm{CFL}_{\mathrm{max}}}$ are defined by eq.(3.20).

This time stepping is defined by setting

```
[TimeDiscr]
  dt_strategy = CFL_based
[]
```

in the file

INPUT/ConfigR7.inp”

There are three input parameters for this strategy, defined in the file

INPUT/dt.inp”

as

-
  ```
  [CFL_based]
    CFL0 = ...
  []
  ```
  : $\mathrm{CFL}_{\mathrm{min}}$ in computing $\Delta t_{\mathrm{CFL}_{\mathrm{min}}}$.

-
  ```
  [CFL_based]
    CFLmax = ...
  []
  ```
  : $\mathrm{CFL}_{\mathrm{max}}$ in computing $\Delta t_{\mathrm{CFL}_{\mathrm{max}}}$.

-
  ```
  [dynamic_time]
    n_steps_start = ...
  []
  ```
  : N_{Init} in eq.(3.21). Default is set to 2.

Class **`dtMAX_based`**

This is a variation of the dynamic time strategy in Section 3.2.4, replacing eq.(3.21) with

$$\Delta t^{(n+2)}_{\mathrm{dtMAX\_based}} = \begin{cases} \Delta t_{\mathrm{min}} & \text{if } N_{\text{time step}} < N_{\mathrm{Init}} \\ \min\left(\max\left(\min\left(\Delta t_{\mathrm{dyn}}, \Delta t_{\mathrm{max}}\right), \Delta t_{\mathrm{min}}\right), \beta\Delta t^{(n+1)}\right) & \text{otherwise} \end{cases} \tag{3.22}$$

where $N_{\text{time step}}$ is current time step number.

This time stepping is defined by setting

```
[TimeDiscr]
  dt_strategy = dtMAX_based
[]
```

in the file

`INPUT/ConfigR7.inp`"

There are three input parameters for this strategy, defined in the file

`INPUT/dt.inp`"

as

-
  ```
  [dtMAX_based]
    dt0 = ...
  []
  ```
  : Δt_{min}.

-
  ```
  [dtMAX_based]
    dt_max = ...
  []
  ```
  : Δt_{max}.

-
  ```
  [dynamic_time]
    n_steps_start = ...
  []
  ```
  : N_{Init} in eq.(3.22). Default is set to 2.

Class `CFL_dt_dtMAX`

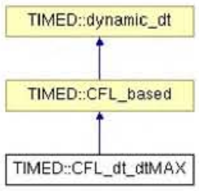

Fig. 3.3 : Hierarchical tree of the Class `CFL_dt_dtMAX`.

This 3-stage dynamic time stepping is a combination of classes `CFL_based`, `dt_const` and `dtMAX_based`. It is defined by setting

```
[TimeDiscr]
  dt_strategy = CFL_dt_dtMAX
[]
```

in the file

`INPUT/ConfigR7.inp`”

Each stage is defined in the file

`INPUT/dt.inp`”

as

Stage I. For $t = [0 \,...\, t_1]$:

-
  ```
  [CFL_dt_dtMAX]
    CFL0 = ...
  []
  ```
  : $\mathrm{CFL}_{\mathrm{min}}$ in computing $\Delta t_{\mathrm{CFL}_{\mathrm{min}}}$.

-
  ```
  [CFL_dt_dtMAX]
    CFLmax = ...
  []
  ```
  : $\mathrm{CFL}_{\mathrm{max}}$ in computing $\Delta t_{\mathrm{CFL}_{\mathrm{max}}}$.

-
  ```
  [CFL_dt_dtMAX]
    n_steps_start = ...
  []
  ```
  : N_{Init} in eq.(3.21). Default is set to 2.

Stage II. For $t = [t_1 \,...\, t_2]$:

-
  ```
  [CFL_dt_dtMAX]
    time2 = ...
  []
  ```
  : t_1.

-
  ```
  [CFL_dt_dtMAX]
    dt2 = ...
  []
  ```
  : constant time step at this stage.

Stage III. For $t \geq t_2$:

-
  ```
  [CFL_dt_dtMAX]
    time3 = ...
  []
  ```
  : t_2.

-
  ```
  [CFL_dt_dtMAX]
    dt3 = ...
  []
  ```
  : $\Delta t_{\min}$ for this stage.

-
  ```
  [CFL_dt_dtMAX]
    dt3_max = ...
  []
  ```
  : $\Delta t_{\max}$ for this stage.

3.3 Space Discretization

THIS chapter describes spatial discretization for hyperbolic, diffusion and reaction operators of eq.(3.1), in 1D (Section 3.3.1). Extensions to 2D and 3D are discussed in Section 3.3.2. The hierarchical tree of the space discretization Class SDiscr is shown in Figure 3.4.

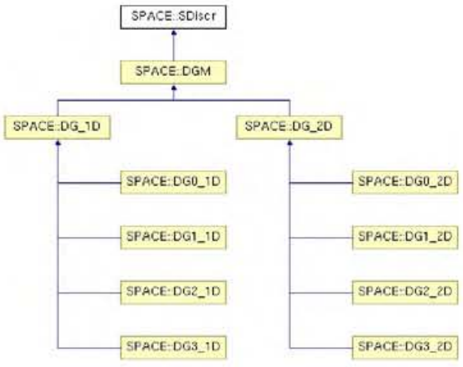

Fig. 3.4 : Hierarchical tree of the Class SDiscr.

3.3.1 One-dimension

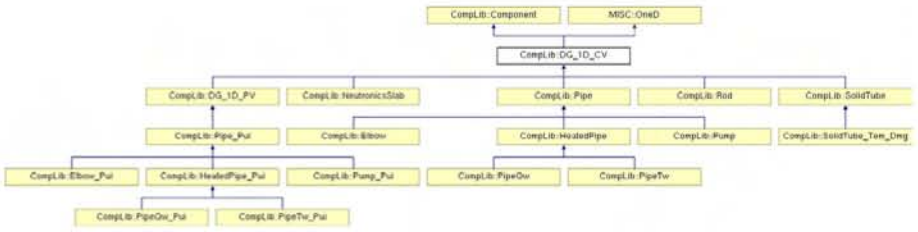

Fig. 3.5 : Hierarchical tree of the Class DG_1D_CV.

In this section we describe 1D discontinuous Galerkin based spatial discretization, as implemented in components derived from DG_1D_CV, Figure 3.5.

Discontinuous Galerkin (DG)

A DG method [RH73, Coc89, CS89, CLS89, CHS90] of the p^{th}-order solves for $(p+1)$ degrees of freedom (DoFs) in each cell, $\vec{\mathcal{U}}^{(n=0,\ldots,p)}_{\text{cell}}$, representing a solution inside each cell $\mathcal{I}_{\text{cell}}$ as:

$$\vec{\mathcal{U}}_{\mathcal{I}_{\text{cell}}}(x) = \sum_{n=0}^{p} \vec{\mathcal{U}}_{\text{cell}}^{(n)} \, \mathfrak{L}_{(n)}(\xi_{\text{cell}}) \tag{3.23}$$

where $\mathfrak{L}_{(n)}(\xi)$ is a basis function of n^{th}-order; typically, the scaled $\left(\xi_{\text{cell}} = \frac{2(x-x_{\text{cell}})}{\Delta x_{\text{cell}}}\right)$ Legendre polynomials. The limiting case of $p = 0$ corresponds to the first-order finite-volume method, with $\vec{\mathcal{U}}_{\text{cell}}^{(0)}$ being the cell-average quantities.

The evolution equations for each degree of freedom in a cell $_{(\mathrm{j})}$ can be written in a semi-discrete form as

$$\frac{d}{dt}\vec{\mathcal{U}}_{\mathrm{j}}^{(n)} = \vec{\mathcal{S}}_{\mathrm{j}}^{(n)} \tag{3.24}$$

Time discretization is described in Chapter 3.2. Implicit schemes are implemented using Jacobian-free Newton Krylov (JFNK) algorithm (see Chapter 3.4), which requires formulation of residuals. In the case of CN scheme, these residuals are computed as:

$$r\vec{e}s_{\mathrm{j}}^{(m)} = \vec{\mathcal{U}}_{\mathrm{j}}^{(m),[n+1]} - \vec{\mathcal{U}}_{\mathrm{j}}^{(m),[n]} - \frac{\Delta t}{2}\left(\vec{\mathcal{S}}_{\mathrm{j}}^{(m)[n+1]} + \vec{\mathcal{S}}_{\mathrm{j}}^{(m),[n]}\right) \tag{3.25}$$

used in JFNK's non-linear solver of type

$$\begin{gathered} \vec{\mathcal{X}}^{(a)} = \vec{\mathcal{X}}^{(a-1)} - \mathbb{J}^{-1} r\vec{e}s^{(a-1)} \\ a = 0, 1, ... \end{gathered} \tag{3.26}$$

Source terms are discretized as

$$\vec{\mathcal{S}}_{\mathrm{j}}^{(n)} = \frac{\left(\mathbf{H}_{\mathrm{j}-\frac{1}{2}} + \mathbf{D}_{\mathrm{j}-\frac{1}{2}}\right)\mathfrak{L}_{(n)}(-1) - \left(\mathbf{H}_{\mathrm{j}+\frac{1}{2}} + \mathbf{D}_{\mathrm{j}+\frac{1}{2}}\right)\mathfrak{L}_{(n)}(1) + \int_{\mathcal{I}_{\mathrm{j}}} \vec{\Upsilon}^{(n)}(x)\,dx}{C_{n}} \tag{3.27}$$

where $\vec{\mathbf{H}}_{\mathrm{j}\pm\frac{1}{2}}$ are numerical hyperbolic fluxes at cell edges, computed with one of the hyperbolic flux treatment schemes (Section 3.3.1), and $\mathbf{D}_{\mathrm{j}\pm\frac{1}{2}}$ are numerical diffusion fluxes[3] (Section 3.3.1). The vector $\vec{\Upsilon}^{(n)}(x)$ is defined as

$$\vec{\Upsilon}^{(n)}(x) \equiv \left[\vec{\mathcal{H}}(x) + \vec{\mathcal{J}}(x)\right]\partial_{x}\mathfrak{L}_{(n)}(x) + \vec{\mathcal{R}}(x)\,\mathfrak{L}_{(n)}(x) \tag{3.28}$$

[3]We treat hyperbolic and diffusion operators similarly. This is different from [vLN05], where the integration-by-parts of the diffusion operator is done twice. This double integration is meaningful only if the diffusion coefficient is constant, which is not the case considered here.

Integration over the cell $\mathcal{I}_{\mathrm{j}}$ is done using N-point Gaussian quadrature formula, where N is varied from 4 to 16 (i.e., the higher the order of the discretization, the more accurate integration is necessary). The parameter C_{n} is a normalization constant for the Legendre polynomials,

$$C_{n} = \frac{\Delta x_j}{2n+1} \tag{3.29}$$

The derivation of the weak form eqs.(3.24)-(3.28) is given by Cockburn [CS89].

In-Cell Recovery (cRDG)

Using $3(p+1)$ DoFs in the cell $_{(\mathrm{j})}$ and its immediate ("von Neumann") neighbors, $_{(\mathrm{j}\pm1)}$, one can either *reconstruct* or *recover* additional degrees of freedom, $\vec{\mathcal{U}}_{\mathrm{j}}^{(n=p+1,\ldots,R),\mathcal{R}}$, where $R = 3p+2$. In the present study, we will follow Lo & van Leer [LvL09], who refer to ***recovery*** as a *weak* interpolation technique for locally building a polynomial from the given on a computational net piecewise-discontinuous polynomials in such a way, so the *$\mathcal{L}_2$-projections* of the "recovered" and original polynomials coincide. This is to be contrasted to the ***reconstruction***, which uses *strong* interpolation for building a polynomial, whose functions and/or its derivatives are matched in *points*.

The "recovered" $(R+1)^{\mathrm{th}}$-order-accurate in-cell polynomial is

$$\vec{\mathcal{U}}_{\mathcal{I}_{\mathrm{j}}}^{\mathcal{R}}(x) = \sum_{n=0}^{R} \vec{\mathcal{U}}_{\mathrm{j}}^{(n),\mathcal{R}} \, \mathfrak{L}_{(n)}(\xi_{\mathrm{j}}) = \underbrace{\sum_{n=0}^{R} \vec{f}^{(n)} \, \mathfrak{H}_{(n)}(\zeta_{\mathrm{j}})}_{\vec{f}(x)} \tag{3.30}$$

where $\mathfrak{H}_{(n)}(\zeta)$ is the Hermite polynomial of the n^{th} order and $\zeta_{\mathrm{j}} = x - x_{\mathrm{j}}$. The *recovered Hermite-based* DoFs $\vec{f}^{(n)}$ are computed using the following "weak statements":

$$\int\limits_{\mathcal{I}_{\mathrm{m}}} \mathfrak{L}_{(n)}(\xi_{\mathrm{m}}) \vec{\mathcal{U}}_{\mathcal{I}_{\mathrm{m}}}(x)\, dx = \int\limits_{\mathcal{I}_{\mathrm{m}}} \mathfrak{L}_{(n)}(\xi_{\mathrm{m}}) \, \vec{f}(x) dx \qquad \text{where } \mathrm{m=j,j\pm1} \text{ and } n=0,\ldots,p \tag{3.31}$$

which corresponds to $(R+1)$ equations solved for unknowns $\vec{f}^{(n=0,\ldots,R)}$. Equations (3.31) enforce the recovered polynomial $\vec{f}(x)$ to be indistinguishable from the DG solutions in cells $_{\mathrm{j,j\pm1}}$, from the broken Sobolev space point of view. Once all $\vec{f}^{(n)}$ are found, the *recovered Legendre-based* DoFs are computed as

$$\vec{\mathcal{U}}_{\mathrm{j}}^{(n),\mathcal{R}} = \frac{1+2n}{\Delta x_{\mathrm{j}}} \int\limits_{\mathcal{I}_{\mathrm{j}}} \mathfrak{L}_{(n)}\left(\xi_{\mathrm{j}}\right) \left(\sum_{n=0}^{R} \vec{f}^{(n)} \, \mathfrak{H}_{(n)}\left(\zeta_{\mathrm{j}}\right) \right) dx \tag{3.32}$$

Equations (3.31) and (3.32) can be straightforwardly solved with any symbolic manipulation software (e.g., Mathematica or MatLab).

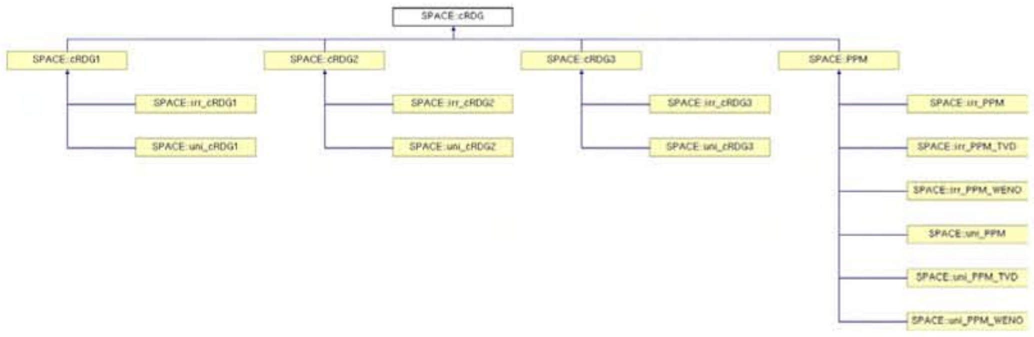

Fig. 3.6 : Hierarchical tree of the Class `cRDG`.

The first $(p+1)$ DoFs of the unlimited cRDG coincide with DG's DoFs, $\vec{\mathcal{U}}_{\mathrm{j}}^{(n),\mathcal{R}} = \vec{\mathcal{U}}_{\mathrm{j}}^{(n)}$, $n = 0, ..., p$, implying that the conservation of the first $(p+1)$ moments is not affected by the recovery. The rest DoFs $(n = p+1, ..., R)$ provide high-order terms, boosting the accuracy of the DGM. They are given below[4,5] (see also Figure 3.6).

PIECEWISE-CONSTANT, $\mathsf{cRDG}_{0}^{(3)}$ (Figure 3.7):

Regular grid, Class `uni_PPM`:

$$\begin{bmatrix} \vec{\mathcal{U}}_{\mathrm{j}}^{(1),\mathcal{R}} \\[2ex] \vec{\mathcal{U}}_{\mathrm{j}}^{(2),\mathcal{R}} \end{bmatrix} = \begin{bmatrix} \frac{\vec{\mathcal{U}}_{\mathrm{j+1}}^{(0)} - \vec{\mathcal{U}}_{\mathrm{j-1}}^{(0)}}{4} \\[2ex] \frac{\vec{\mathcal{U}}_{\mathrm{j+1}}^{(0)} - 2\vec{\mathcal{U}}_{\mathrm{j}}^{(0)} + \vec{\mathcal{U}}_{\mathrm{j-1}}^{(0)}}{12} \end{bmatrix} \tag{3.33}$$

[4]In this manuscript, the schemes are denoted as $\mathrm{cRDG}_{p}^{(\mathrm{o})}$, where p is the order of basis functions for the DG, while (o) corresponds to the order of spatial accuracy.

[5]For irregular grids, we show here only the *piecewise-constant* and *piecewise-linear* discretizations.

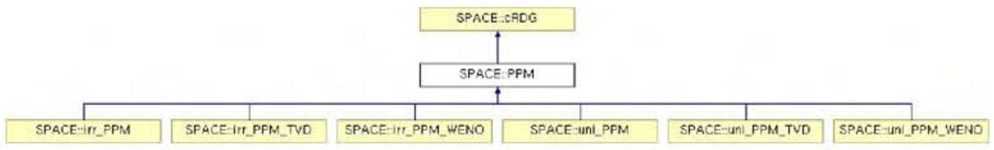

Fig. 3.7 : Hierarchical tree of the Class `PPM` (cRDG$_0$).

Irregular grid, Class `irr_PPM`:

$$\begin{bmatrix} \vec{\mathcal{U}}_{\mathrm{j}}^{(1),\mathcal{R}} \\ \\ \vec{\mathcal{U}}_{\mathrm{j}}^{(2),\mathcal{R}} \end{bmatrix} = \begin{bmatrix} \dfrac{\begin{pmatrix} \Delta x_{\mathrm{j}} \left((\Delta x_{\mathrm{j}+1} - \Delta x_{\mathrm{j}-1}) (3\Delta x_{\mathrm{j}} + 2(\Delta x_{\mathrm{j}-1} + \Delta x_{\mathrm{j}+1})) \vec{\mathcal{U}}_{\mathrm{j}}^{(0)} - \right. \\ - (\Delta x_{\mathrm{j}} + \Delta x_{\mathrm{j}+1}) (\Delta x_{\mathrm{j}} + 2\Delta x_{\mathrm{j}+1}) \vec{\mathcal{U}}_{\mathrm{j}-1}^{(0)} + \\ \left. + (\Delta x_{\mathrm{j}} + \Delta x_{\mathrm{j}-1}) (\Delta x_{\mathrm{j}} + 2\Delta x_{\mathrm{j}-1}) \vec{\mathcal{U}}_{\mathrm{j}+1}^{(0)} \right) \end{pmatrix}}{2\left(\Delta x_{\mathrm{j}} + \Delta x_{\mathrm{j}-1}\right)\left(\Delta x_{\mathrm{j}} + \Delta x_{\mathrm{j}+1}\right)\left(\Delta x_{\mathrm{j}} + \Delta x_{\mathrm{j}-1} + \Delta x_{\mathrm{j}+1}\right)} \\ \\ \dfrac{\Delta x_{\mathrm{j}}^2 \begin{pmatrix} \Delta x_{\mathrm{j}+1} \left(\vec{\mathcal{U}}_{\mathrm{j}-1}^{(0)} - \vec{\mathcal{U}}_{\mathrm{j}}^{(0)} \right) + \Delta x_{\mathrm{j}-1} \left(\vec{\mathcal{U}}_{\mathrm{j}+1}^{(0)} - \vec{\mathcal{U}}_{\mathrm{j}}^{(0)} \right) + \\ + \Delta x_{\mathrm{j}} \left(-2\vec{\mathcal{U}}_{\mathrm{j}}^{(0)} + \vec{\mathcal{U}}_{\mathrm{j}-1}^{(0)} + \vec{\mathcal{U}}_{\mathrm{j}+1}^{(0)} \right) \end{pmatrix}}{2\left(\Delta x_{\mathrm{j}} + \Delta x_{\mathrm{j}-1}\right)\left(\Delta x_{\mathrm{j}} + \Delta x_{\mathrm{j}+1}\right)\left(\Delta x_{\mathrm{j}} + \Delta x_{\mathrm{j}-1} + \Delta x_{\mathrm{j}+1}\right)} \end{bmatrix} \tag{3.34}$$

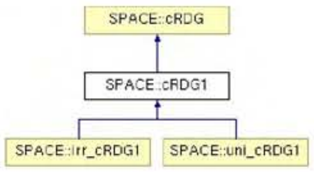

Fig. 3.8 : Hierarchical tree of the Class `cRDG1`.

<u>PIECEWISE-LINEAR</u>, $\mathsf{cRDG}_1^{(6)}$ (Figure 3.8):

Regular grid, Class `uni_cRDG1`:

$$\begin{bmatrix} \vec{\mathcal{U}}_j^{(2),\mathcal{R}} \\ \vec{\mathcal{U}}_j^{(3),\mathcal{R}} \\ \vec{\mathcal{U}}_j^{(4),\mathcal{R}} \\ \vec{\mathcal{U}}_j^{(5),\mathcal{R}} \end{bmatrix} = \begin{bmatrix} \dfrac{73\left(\vec{\mathcal{U}}_{j+1}^{(0)}+\vec{\mathcal{U}}_{j-1}^{(0)}\right)-45\left(\vec{\mathcal{U}}_{j+1}^{(1)}-\vec{\mathcal{U}}_{j-1}^{(1)}\right)-146\vec{\mathcal{U}}_j^{(0)}}{336} \\ \dfrac{357\left(\vec{\mathcal{U}}_{j+1}^{(0)}-\vec{\mathcal{U}}_{j-1}^{(0)}\right)-197\left(\vec{\mathcal{U}}_{j+1}^{(1)}+\vec{\mathcal{U}}_{j-1}^{(1)}\right)-1034\vec{\mathcal{U}}_j^{(1)}}{3888} \\ \dfrac{2\vec{\mathcal{U}}_j^{(0)}-\vec{\mathcal{U}}_{j-1}^{(0)}-\vec{\mathcal{U}}_{j+1}^{(0)}-\vec{\mathcal{U}}_{j-1}^{(1)}+\vec{\mathcal{U}}_{j+1}^{(1)}}{112} \\ \dfrac{15\left(\vec{\mathcal{U}}_{j-1}^{(0)}-\vec{\mathcal{U}}_{j+1}^{(0)}\right)+11\left(\vec{\mathcal{U}}_{j-1}^{(1)}+\vec{\mathcal{U}}_{j+1}^{(1)}\right)+38\vec{\mathcal{U}}_j^{(1)}}{3888} \end{bmatrix} \tag{3.35}$$

Irregular grid, Class `irr_cRDG1`:

$$\vec{\mathcal{U}}_{\mathrm{j}}^{(2),\mathcal{R}} = \frac{\begin{pmatrix} \vec{\mathcal{U}}_{\mathrm{j+1}}^{(1)}\xi_{\mathrm{j-1}} \left(-14(5\xi_{\mathrm{j+1}}(\xi_{\mathrm{j+1}}+1)+1)\xi_{\mathrm{j-1}}^3 - 2(\xi_{\mathrm{j+1}}(7\xi_{\mathrm{j+1}}(7\xi_{\mathrm{j+1}}+18)+82)+13)\xi_{\mathrm{j-1}}^2\right. \\ -(2\xi_{\mathrm{j+1}}(49\xi_{\mathrm{j+1}}(\xi_{\mathrm{j+1}}+2)+54)+15)\xi_{\mathrm{j-1}} - (\xi_{\mathrm{j+1}}+1)(7\xi_{\mathrm{j+1}}(4\xi_{\mathrm{j+1}}+3)+3)) \\ (\xi_{\mathrm{j-1}}+1)^3 + \xi_{\mathrm{j+1}} \left(\vec{\mathcal{U}}_{\mathrm{j-1}}^{(1)} \left(14(5\xi_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}+1)+1)\xi_{\mathrm{j+1}}^3 + 2(\xi_{\mathrm{j-1}}(7\xi_{\mathrm{j-1}}(7\xi_{\mathrm{j-1}}+18)\right.\right. \\ +82)+13)\xi_{\mathrm{j+1}}^2 + 2\xi_{\mathrm{j-1}}(49\xi_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}+2)+54)\xi_{\mathrm{j+1}} + 15\xi_{\mathrm{j+1}} + \xi_{\mathrm{j-1}}(7\xi_{\mathrm{j-1}}(4\xi_{\mathrm{j-1}}+7) \\ +24)+3)\,(\xi_{\mathrm{j+1}}+1)^3 + 30(\vec{\mathcal{U}}_{\mathrm{j-1}}^{(0)} - 2\vec{\mathcal{U}}_{\mathrm{j}}^{(0)})\xi_{\mathrm{j-1}} + \xi_{\mathrm{j-1}} \left(14(9\xi_{\mathrm{j-1}}\vec{\mathcal{U}}_{\mathrm{j-1}}^{(0)} + 6\vec{\mathcal{U}}_{\mathrm{j-1}}^{(0)}\right. \\ -3\vec{\mathcal{U}}_{\mathrm{j}}^{(0)}(3\xi_{\mathrm{j-1}}+2) + \vec{\mathcal{U}}_{\mathrm{j}}^{(1)}(2\xi_{\mathrm{j-1}}(2\xi_{\mathrm{j-1}}+5)+5))\xi_{\mathrm{j+1}}^6 - 2(3\vec{\mathcal{U}}_{\mathrm{j}}^{(0)}(35\xi_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}+4) \\ +76) - 3\vec{\mathcal{U}}_{\mathrm{j-1}}^{(0)}(35\xi_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}+4)+76) - \vec{\mathcal{U}}_{\mathrm{j}}^{(1)}(\xi_{\mathrm{j-1}}+2)(14\xi_{\mathrm{j-1}}(4\xi_{\mathrm{j-1}}+13)+97))\xi_{\mathrm{j+1}}^5 \\ -(3\vec{\mathcal{U}}_{\mathrm{j}}^{(0)}(4\xi_{\mathrm{j-1}}(70\xi_{\mathrm{j-1}}+177)+333) - 3\vec{\mathcal{U}}_{\mathrm{j-1}}^{(0)}(4\xi_{\mathrm{j-1}}(70\xi_{\mathrm{j-1}}+177)+333) \\ -\vec{\mathcal{U}}_{\mathrm{j}}^{(1)}(\xi_{\mathrm{j-1}}(2\xi_{\mathrm{j-1}}(224\xi_{\mathrm{j-1}}+901)+2313)+864))\xi_{\mathrm{j+1}}^4 - \left(3\vec{\mathcal{U}}_{\mathrm{j}}^{(0)}\left(440\xi_{\mathrm{j-1}}^2 + 894\xi_{\mathrm{j-1}} + 377\right)\right. \\ -3\vec{\mathcal{U}}_{\mathrm{j-1}}^{(0)}\left(440\xi_{\mathrm{j-1}}^2 + 894\xi_{\mathrm{j-1}} + 377\right) + 7\vec{\mathcal{U}}_{\mathrm{j}}^{(1)}\left(\xi_{\mathrm{j-1}}\left(\xi_{\mathrm{j-1}}\left(16\xi_{\mathrm{j-1}}^2(\xi_{\mathrm{j-1}}+4)-285\right)-385\right)\right. \\ -137))\,\xi_{\mathrm{j+1}}^3 - \left(-3\vec{\mathcal{U}}_{\mathrm{j-1}}^{(0)}(4\xi_{\mathrm{j-1}}(85\xi_{\mathrm{j-1}}+151)+233) + 3\vec{\mathcal{U}}_{\mathrm{j}}^{(0)}(2\xi_{\mathrm{j-1}}(5\xi_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}(7\xi_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}\right. \\ +4)+44)+68)+367)+253) + \vec{\mathcal{U}}_{\mathrm{j}}^{(1)}\left(\xi_{\mathrm{j-1}}\left(\xi_{\mathrm{j-1}}^2(2\xi_{\mathrm{j-1}}(14\xi_{\mathrm{j-1}}(2\xi_{\mathrm{j-1}}+21)+901)\right.\right. \\ +1995)-1274)-518))\,\xi_{\mathrm{j+1}}^2 - \left(-15\vec{\mathcal{U}}_{\mathrm{j-1}}^{(0)}\left(26\xi_{\mathrm{j-1}}^2 + 42\xi_{\mathrm{j-1}} + 15\right) + \vec{\mathcal{U}}_{\mathrm{j}}^{(1)}\left(\xi_{\mathrm{j-1}}^2(\xi_{\mathrm{j-1}}\right.\right. \\ +2)(\xi_{\mathrm{j-1}}(2\xi_{\mathrm{j-1}}(70\xi_{\mathrm{j-1}}+321)+1029)+637)-105) + 3\vec{\mathcal{U}}_{\mathrm{j}}^{(0)}(2\xi_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}+2)(\xi_{\mathrm{j-1}} \\ (\xi_{\mathrm{j-1}}(7\xi_{\mathrm{j-1}}(3\xi_{\mathrm{j-1}}+14)+158)+131)+105)+105))\,\xi_{\mathrm{j+1}} - 15(21\vec{\mathcal{U}}_{\mathrm{j}}^{(0)} + 7\vec{\mathcal{U}}_{\mathrm{j}}^{(1)} \\ -6\vec{\mathcal{U}}_{\mathrm{j-1}}^{(0)})\xi_{\mathrm{j-1}} - \xi_{\mathrm{j-1}}^2(-60\vec{\mathcal{U}}_{\mathrm{j-1}}^{(0)} + 3\vec{\mathcal{U}}_{\mathrm{j}}^{(0)}(\xi_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}(4\xi_{\mathrm{j-1}}(7\xi_{\mathrm{j-1}}+38)+333)+377) \\ +253) + \vec{\mathcal{U}}_{\mathrm{j}}^{(1)}(\xi_{\mathrm{j-1}}(2\xi_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}(35\xi_{\mathrm{j-1}}+194)+432)+959)+518)) + 3\vec{\mathcal{U}}_{\mathrm{j+1}}^{(0)}(\xi_{\mathrm{j-1}} \\ +1)^3\left(10(7\xi_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}+1)+2)\xi_{\mathrm{j+1}}^2 + 2\xi_{\mathrm{j-1}}(7\xi_{\mathrm{j-1}}(3\xi_{\mathrm{j-1}}+11)+60)\xi_{\mathrm{j+1}} + 30\xi_{\mathrm{j+1}}\right. \\ +\xi_{\mathrm{j-1}}(4\xi_{\mathrm{j-1}}(7\xi_{\mathrm{j-1}}+17)+45)+10))) \end{pmatrix}}{42\xi_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}+1)^3\xi_{\mathrm{j+1}}(\xi_{\mathrm{j+1}}+1)^3(\xi_{\mathrm{j-1}}+\xi_{\mathrm{j+1}}+1)^3} \tag{3.36}$$

$$\vec{\mathcal{U}}_{\mathrm{j}}^{(3),\mathcal{R}} = \frac{\begin{pmatrix} \vec{\mathcal{U}}_{\mathrm{j+1}}^{(1)}\xi_{\mathrm{j-1}}\left(6(2\xi_{\mathrm{j+1}}+1)\xi_{\mathrm{j-1}}^3 + 6\left(-2\xi_{\mathrm{j+1}}^2 + \xi_{\mathrm{j+1}} + 1\right)\xi_{\mathrm{j-1}}^2 - (2\xi_{\mathrm{j+1}}+1)(3\xi_{\mathrm{j+1}}(7\xi_{\mathrm{j+1}}+9)\right. \\ +1)\xi_{\mathrm{j-1}} - (\xi_{\mathrm{j+1}}+1)(3\xi_{\mathrm{j+1}}(7\xi_{\mathrm{j+1}}+4)+1))\,(\xi_{\mathrm{j-1}}+1)^3 + \xi_{\mathrm{j+1}}\left(-\vec{\mathcal{U}}_{\mathrm{j-1}}^{(1)}\left(21(2\xi_{\mathrm{j+1}}+1)\xi_{\mathrm{j-1}}^3\right.\right. \\ +3(\xi_{\mathrm{j+1}}(4\xi_{\mathrm{j+1}}+25)+11)\xi_{\mathrm{j-1}}^2 + (\xi_{\mathrm{j+1}}(29-6\xi_{\mathrm{j+1}}(2\xi_{\mathrm{j+1}}+1))+13)\xi_{\mathrm{j-1}} + \xi_{\mathrm{j+1}} \\ \left.-6\xi_{\mathrm{j+1}}^2(\xi_{\mathrm{j+1}}+1)+1\right)(\xi_{\mathrm{j+1}}+1)^3 - \xi_{\mathrm{j-1}}\left(3\left(\vec{\mathcal{U}}_{\mathrm{j+1}}^{(0)}\left(6\xi_{\mathrm{j-1}}^3 - 6(\xi_{\mathrm{j+1}}-1)\xi_{\mathrm{j-1}}^2\right.\right.\right. \\ \left.-5\left(6\xi_{\mathrm{j+1}}^2 + 9\xi_{\mathrm{j+1}} + 2\right)\xi_{\mathrm{j-1}} - 5(\xi_{\mathrm{j+1}}+1)(3\xi_{\mathrm{j+1}}+1)\right)(\xi_{\mathrm{j-1}}+1)^3 + \vec{\mathcal{U}}_{\mathrm{j-1}}^{(0)}(\xi_{\mathrm{j+1}}+1)^3 \\ \left(-6\xi_{\mathrm{j+1}}^3 + 6(\xi_{\mathrm{j-1}}-1)\xi_{\mathrm{j+1}}^2 + 5\left(6\xi_{\mathrm{j-1}}^2 + 9\xi_{\mathrm{j-1}} + 2\right)\xi_{\mathrm{j+1}} + 5(\xi_{\mathrm{j-1}}+1)(3\xi_{\mathrm{j-1}}+1)\right) \\ -\vec{\mathcal{U}}_{\mathrm{j}}^{(0)}(\xi_{\mathrm{j-1}}-\xi_{\mathrm{j+1}})\left(6\xi_{\mathrm{j-1}}^5 + 24\xi_{\mathrm{j-1}}^4 + (26-3\xi_{\mathrm{j+1}}(10\xi_{\mathrm{j+1}}+13))\xi_{\mathrm{j-1}}^3 - (3\xi_{\mathrm{j+1}}(2\xi_{\mathrm{j+1}}(5\xi_{\mathrm{j+1}}\right. \\ +24)+49)+11)\xi_{\mathrm{j-1}}^2 - (\xi_{\mathrm{j+1}}(3\xi_{\mathrm{j+1}}(13\xi_{\mathrm{j+1}}+49)+137)+24)\xi_{\mathrm{j-1}} + (\xi_{\mathrm{j+1}}+1)(\xi_{\mathrm{j+1}} \\ (2\xi_{\mathrm{j+1}}(3\xi_{\mathrm{j+1}}(\xi_{\mathrm{j+1}}+3)+4)-19)-5))) + \vec{\mathcal{U}}_{\mathrm{j}}^{(1)}\left(-6(\xi_{\mathrm{j+1}}+2)\xi_{\mathrm{j-1}}^6 + 6(\xi_{\mathrm{j+1}}-4)(\xi_{\mathrm{j+1}}\right. \\ +2)\xi_{\mathrm{j-1}}^5 + (8\xi_{\mathrm{j+1}}(3\xi_{\mathrm{j+1}}(2\xi_{\mathrm{j+1}}+7)+17)-43)\xi_{\mathrm{j-1}}^4 + (2\xi_{\mathrm{j+1}}(6\xi_{\mathrm{j+1}}(4\xi_{\mathrm{j+1}}(\xi_{\mathrm{j+1}}+7)+63) \\ +301)+77)\xi_{\mathrm{j-1}}^3 + 3(2\xi_{\mathrm{j+1}}(\xi_{\mathrm{j+1}}(\xi_{\mathrm{j+1}}(\xi_{\mathrm{j+1}}(\xi_{\mathrm{j+1}}+28)+126)+224)+161)+63)\xi_{\mathrm{j-1}}^2 \\ +(2\xi_{\mathrm{j+1}}(\xi_{\mathrm{j+1}}(\xi_{\mathrm{j+1}}(\xi_{\mathrm{j+1}}(68-3\xi_{\mathrm{j+1}}(\xi_{\mathrm{j+1}}+2))+301)+483)+322)+133)\xi_{\mathrm{j-1}} \\ -(\xi_{\mathrm{j+1}}+1)(\xi_{\mathrm{j+1}}(\xi_{\mathrm{j+1}}(\xi_{\mathrm{j+1}}(12\xi_{\mathrm{j+1}}(\xi_{\mathrm{j+1}}+3)+7)-84)-105)-28)))) \end{pmatrix}}{18\xi_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}+1)^3\xi_{\mathrm{j+1}}(\xi_{\mathrm{j+1}}+1)^3(\xi_{\mathrm{j-1}}+\xi_{\mathrm{j+1}}+1)^3} \tag{3.37}$$

$$\vec{\mathcal{U}}_{j}^{(4),\mathcal{R}} = \frac{\left(\begin{array}{c}
\vec{\mathcal{U}}^{(1)}_{j+1}\xi_{j-1}\big((8\xi_{j+1}+4)\xi^2_{j-1}+(\xi_{j+1}(7\xi_{j+1}+15)+5)\xi_{j-1}+\xi_{j+1}-7\xi^2_{j+1}(\xi_{j+1}+1)+1\big) \\
(\xi_{j-1}+1)^3+\xi_{j+1}\big(-\vec{\mathcal{U}}^{(1)}_{j-1}\big(-7\xi^3_{j-1}+7(\xi_{j+1}-1)\xi^2_{j-1}+(\xi_{j+1}(8\xi_{j+1}+15)+1)\xi_{j-1}+ \\
(\xi_{j+1}+1)(4\xi_{j+1}+1)\big)(\xi_{j+1}+1)^3+6\vec{\mathcal{U}}^{(0)}_{j}\xi_{j-1}-3\vec{\mathcal{U}}^{(0)}_{j-1}\xi_{j-1}+\xi_{j-1}\big(4(3\vec{\mathcal{U}}^{(0)}_{j}-3\vec{\mathcal{U}}^{(0)}_{j-1} \\
-\vec{\mathcal{U}}^{(1)}_{j}(\xi_{j-1}+2))\xi^5_{j+1}+(15\vec{\mathcal{U}}^{(0)}_{j}(\xi_{j-1}+4)-15\vec{\mathcal{U}}^{(0)}_{j-1}(\xi_{j-1}+4)-\vec{\mathcal{U}}^{(1)}_{j}(4\xi_{j-1}(2\xi_{j-1} \\
+9)+43))\xi^4_{j+1}-(3\vec{\mathcal{U}}^{(0)}_{j-1}((11-5\xi_{j-1})\xi_{j-1}+37)+7\vec{\mathcal{U}}^{(1)}_{j}(2\xi_{j-1}(2\xi_{j-1}+7)+13) \\
+3\vec{\mathcal{U}}^{(0)}_{j}(\xi_{j-1}(5\xi_{j-1}-11)-37))\xi^3_{j+1}-\big(9(1-5\xi_{j-1})\xi_{j-1}\vec{\mathcal{U}}^{(0)}_{j-1}+93\vec{\mathcal{U}}^{(0)}_{j-1}+\vec{\mathcal{U}}^{(1)}_{j} \\
\big(-8\xi^4_{j-1}-28\xi^3_{j-1}+98\xi_{j-1}+91\big)+3\vec{\mathcal{U}}^{(0)}_{j}(\xi_{j-1}(5\xi_{j-1}(\xi_{j-1}+6)+12)-26)\big)\xi^2_{j+1}+ \\
\big(3\vec{\mathcal{U}}^{(0)}_{j-1}(\xi_{j-1}(15\xi_{j-1}+7)-11)+3\vec{\mathcal{U}}^{(0)}_{j}(\xi_{j-1}(\xi_{j-1}(\xi_{j-1}+3)(5\xi_{j-1}-4)-14)+7)+ \\
\vec{\mathcal{U}}^{(1)}_{j}\big(2\xi^2_{j-1}(\xi_{j-1}(2\xi_{j-1}(\xi_{j-1}+9)+49)+49)-35\big)\big)\xi_{j+1}+\xi_{j-1}(3\vec{\mathcal{U}}^{(0)}_{j-1}(5\xi_{j-1}+4) \\
+3\vec{\mathcal{U}}^{(0)}_{j}(\xi_{j-1}(\xi_{j-1}+2)(4\xi_{j-1}(\xi_{j-1}+3)+13)+7)+\vec{\mathcal{U}}^{(1)}_{j}(\xi_{j-1}+1)(\xi_{j-1}(\xi_{j-1}(8\xi_{j-1} \\
+35)+56)+35))-3\vec{\mathcal{U}}^{(0)}_{j+1}(\xi_{j-1}+1)^3\big(-5\xi^2_{j+1}+5\xi_{j-1}\xi_{j+1}-4\xi_{j+1} \\
+4\xi_{j-1}(\xi_{j-1}+2)+1)))
\end{array}\right)}{14\xi_{j-1}(\xi_{j-1}+1)^3\xi_{j+1}(\xi_{j+1}+1)^3(\xi_{j-1}+\xi_{j+1}+1)^3} \tag{3.38}$$

$$\vec{\mathcal{U}}_{j}^{(5),\mathcal{R}} = \frac{\left(\begin{array}{c}
\vec{\mathcal{U}}^{(1)}_{j+1}\xi_{j-1}\big(3\xi^2_{j+1}+2(\xi_{j-1}+2)\xi_{j+1}+\xi_{j-1}+1\big)(\xi_{j-1}+1)^3+\xi_{j+1}\big(\vec{\mathcal{U}}^{(1)}_{j-1} \\
(\xi_{j+1}+\xi_{j-1}(3\xi_{j-1}+2\xi_{j+1}+4)+1)(\xi_{j+1}+1)^3+\xi_{j-1}\big(3\big(-\vec{\mathcal{U}}^{(0)}_{j+1}(\xi_{j-1} \\
+2\xi_{j+1}+2)(\xi_{j-1}+1)^3+\vec{\mathcal{U}}^{(0)}_{j-1}(\xi_{j+1}+1)^3(2\xi_{j-1}+\xi_{j+1}+2)+\vec{\mathcal{U}}^{(0)}_{j} \\
(\xi_{j-1}-\xi_{j+1})\big(\xi^3_{j+1}+(3\xi_{j-1}+5)\xi^2_{j+1}+\xi_{j-1}(3\xi_{j-1}+11)\xi_{j+1}+9\xi_{j+1}+ \\
\xi_{j-1}(\xi_{j-1}(\xi_{j-1}+5)+9)+5))+\vec{\mathcal{U}}^{(1)}_{j}(\xi_{j-1}+\xi_{j+1}+2)\big((\xi_{j+1}+2)\xi^3_{j-1} \\
+(\xi_{j+1}(2\xi_{j+1}+7)+7)\xi^2_{j-1}+(\xi_{j+1}+2)(\xi_{j+1}(\xi_{j+1}+5)+5)\xi_{j-1}+ \\
(\xi_{j+1}+1)(\xi_{j+1}(2\xi_{j+1}+5)+5))))
\end{array}\right)}{18\xi_{j-1}(\xi_{j-1}+1)^3\xi_{j+1}(\xi_{j+1}+1)^3(\xi_{j-1}+\xi_{j+1}+1)^3} \tag{3.39}$$

where $\xi_{j+1} = \frac{\Delta x_{j+1}}{\Delta x_{j}}$ and $\xi_{j-1} = \frac{\Delta x_{j-1}}{\Delta x_{j}}$.

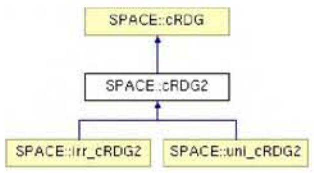

Fig. 3.9 : Hierarchical tree of the Class `cRDG2`.

PIECEWISE-QUADRATIC, $\mathsf{cRDG}_2^{(9)}$, (Figure 3.9).

(Regular grid), Class `uni_cRDG2`:

$$\begin{bmatrix} \vec{\mathcal{U}}_{j}^{(3),\mathcal{R}} \\ \vec{\mathcal{U}}_{j}^{(4),\mathcal{R}} \\ \vec{\mathcal{U}}_{j}^{(5),\mathcal{R}} \\ \vec{\mathcal{U}}_{j}^{(6),\mathcal{R}} \\ \vec{\mathcal{U}}_{j}^{(7),\mathcal{R}} \\ \vec{\mathcal{U}}_{j}^{(8),\mathcal{R}} \end{bmatrix} = \begin{bmatrix} \frac{-28082\vec{\mathcal{U}}_{j}^{(1)}+11436(\vec{\mathcal{U}}_{j+1}^{(0)}-\vec{\mathcal{U}}_{j-1}^{(0)})-8831(\vec{\mathcal{U}}_{j-1}^{(1)}+\vec{\mathcal{U}}_{j+1}^{(1)})+4185(\vec{\mathcal{U}}_{j+1}^{(2)}-\vec{\mathcal{U}}_{j-1}^{(2)})}{57024} \\ \frac{-232480\vec{\mathcal{U}}_{j}^{(0)}-452074\vec{\mathcal{U}}_{j}^{(2)}+116240(\vec{\mathcal{U}}_{j-1}^{(0)}+\vec{\mathcal{U}}_{j+1}^{(0)})+84505(\vec{\mathcal{U}}_{j-1}^{(1)}-\vec{\mathcal{U}}_{j+1}^{(1)})+35627(\vec{\mathcal{U}}_{j-1}^{(2)}+\vec{\mathcal{U}}_{j+1}^{(2)})}{1235520} \\ \frac{748\vec{\mathcal{U}}_{j}^{(1)}+345(\vec{\mathcal{U}}_{j-1}^{(0)}-\vec{\mathcal{U}}_{j+1}^{(0)})+316(\vec{\mathcal{U}}_{j-1}^{(1)}+\vec{\mathcal{U}}_{j+1}^{(1)})+189(\vec{\mathcal{U}}_{j-1}^{(2)}-\vec{\mathcal{U}}_{j+1}^{(2)})}{16848} \\ \frac{4030\vec{\mathcal{U}}_{j}^{(0)}+6238\vec{\mathcal{U}}_{j}^{(2)}-2015(\vec{\mathcal{U}}_{j-1}^{(0)}+\vec{\mathcal{U}}_{j+1}^{(0)})+1630(\vec{\mathcal{U}}_{j+1}^{(1)}-\vec{\mathcal{U}}_{j-1}^{(1)})-809(\vec{\mathcal{U}}_{j-1}^{(2)}+\vec{\mathcal{U}}_{j+1}^{(2)})}{213840} \\ \frac{-250\vec{\mathcal{U}}_{j}^{(1)}+120(\vec{\mathcal{U}}_{j+1}^{(0)}-\vec{\mathcal{U}}_{j-1}^{(0)})-115(\vec{\mathcal{U}}_{j-1}^{(1)}+\vec{\mathcal{U}}_{j+1}^{(1)})+81(\vec{\mathcal{U}}_{j+1}^{(2)}-\vec{\mathcal{U}}_{j-1}^{(2)})}{247104} \\ \frac{-440\vec{\mathcal{U}}_{j}^{(0)}-626\vec{\mathcal{U}}_{j}^{(2)}+220(\vec{\mathcal{U}}_{j-1}^{(0)}+\vec{\mathcal{U}}_{j+1}^{(0)})+185(\vec{\mathcal{U}}_{j-1}^{(1)}-\vec{\mathcal{U}}_{j+1}^{(1)})+103(\vec{\mathcal{U}}_{j-1}^{(2)}+\vec{\mathcal{U}}_{j+1}^{(2)})}{1010880} \end{bmatrix} \tag{3.40}$$

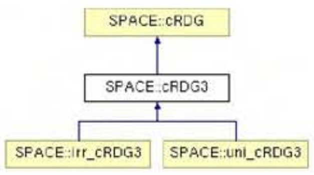

Fig. 3.10 : Hierarchical tree of the Class `cRDG3`.

PIECEWISE-CUBIC, $\mathsf{cRDG}_3^{(12)}$, (Figure 3.10).

(Regular grid)[6], Class `uni_cRDG3`:

$$\begin{bmatrix} \vec{\mathcal{U}}_j^{(4),\mathcal{R}} \\ \vec{\mathcal{U}}_j^{(5),\mathcal{R}} \\ \vec{\mathcal{U}}_j^{(6),\mathcal{R}} \\ \vec{\mathcal{U}}_j^{(7),\mathcal{R}} \\ \vec{\mathcal{U}}_j^{(8),\mathcal{R}} \\ \vec{\mathcal{U}}_j^{(9),\mathcal{R}} \\ \vec{\mathcal{U}}_j^{(10),\mathcal{R}} \\ \vec{\mathcal{U}}_j^{(11),\mathcal{R}} \end{bmatrix} = \begin{bmatrix} \dfrac{\begin{pmatrix} -17008750\vec{\mathcal{U}}_j^{(0)} - 25538674\vec{\mathcal{U}}_j^{(2)} + 8504375(\vec{\mathcal{U}}_{j-1}^{(0)} + \vec{\mathcal{U}}_{j+1}^{(0)}) + 7153735(\vec{\mathcal{U}}_{j-1}^{(1)} - \\ -\vec{\mathcal{U}}_{j+1}^{(1)}) + 4665497(\vec{\mathcal{U}}_{j-1}^{(2)} + \vec{\mathcal{U}}_{j+1}^{(2)}) + 1806705(\vec{\mathcal{U}}_{j-1}^{(3)} - \vec{\mathcal{U}}_{j+1}^{(3)}) \end{pmatrix}}{44478720} \\ \dfrac{\begin{pmatrix} -196469378\vec{\mathcal{U}}_j^{(1)} - 403694182\vec{\mathcal{U}}_j^{(3)} + 82503365(\vec{\mathcal{U}}_{j+1}^{(0)} - \vec{\mathcal{U}}_{j-1}^{(0)}) - 66772041\times \\ (\vec{\mathcal{U}}_{j-1}^{(1)} + \vec{\mathcal{U}}_{j+1}^{(1)}) + 40397287(\vec{\mathcal{U}}_{j+1}^{(2)} - \vec{\mathcal{U}}_{j-1}^{(2)}) - 14038579(\vec{\mathcal{U}}_{j-1}^{(3)} + \vec{\mathcal{U}}_{j+1}^{(3)}) \end{pmatrix}}{866121984} \\ \dfrac{\begin{pmatrix} 1293050\vec{\mathcal{U}}_j^{(0)} + 1560302\vec{\mathcal{U}}_j^{(2)} - 646525(\vec{\mathcal{U}}_{j-1}^{(0)} + \vec{\mathcal{U}}_{j+1}^{(0)}) + 589265(\vec{\mathcal{U}}_{j+1}^{(1)} - \vec{\mathcal{U}}_{j-1}^{(1)}) - \\ -436591(\vec{\mathcal{U}}_{j-1}^{(2)} + \vec{\mathcal{U}}_{j+1}^{(2)}) + 193995(\vec{\mathcal{U}}_{j+1}^{(3)} - \vec{\mathcal{U}}_{j-1}^{(3)}) \end{pmatrix}}{19388160} \\ \dfrac{\begin{pmatrix} 201983810\vec{\mathcal{U}}_j^{(1)} + 320885230\vec{\mathcal{U}}_j^{(3)} + 88115825(\vec{\mathcal{U}}_{j-1}^{(0)} - \vec{\mathcal{U}}_{j+1}^{(0)}) + \\ +75239745(\vec{\mathcal{U}}_{j-1}^{(1)} + \vec{\mathcal{U}}_{j+1}^{(1)}) + 49833007(\vec{\mathcal{U}}_{j-1}^{(2)} - \vec{\mathcal{U}}_{j+1}^{(2)}) + 19243255(\vec{\mathcal{U}}_{j-1}^{(3)} + \vec{\mathcal{U}}_{j+1}^{(3)}) \end{pmatrix}}{5485439232} \\ \dfrac{\begin{pmatrix} -90550\vec{\mathcal{U}}_j^{(0)} - 101002\vec{\mathcal{U}}_j^{(2)} + 45275(\vec{\mathcal{U}}_{j-1}^{(0)} + \vec{\mathcal{U}}_{j+1}^{(0)}) + 42475(\vec{\mathcal{U}}_{j-1}^{(1)} - \vec{\mathcal{U}}_{j+1}^{(1)}) + \\ +33701(\vec{\mathcal{U}}_{j-1}^{(2)} + \vec{\mathcal{U}}_{j+1}^{(2)}) + 16605(\vec{\mathcal{U}}_{j-1}^{(3)} - \vec{\mathcal{U}}_{j+1}^{(3)}) \end{pmatrix}}{25608960} \\ \dfrac{\begin{pmatrix} -123718\vec{\mathcal{U}}_j^{(1)} - 178034\vec{\mathcal{U}}_j^{(3)} + 54943(\vec{\mathcal{U}}_{j+1}^{(0)} - \vec{\mathcal{U}}_{j-1}^{(0)}) - 48027(\vec{\mathcal{U}}_{j-1}^{(1)} + \vec{\mathcal{U}}_{j+1}^{(1)}) + \\ +33509(\vec{\mathcal{U}}_{j+1}^{(2)} - \vec{\mathcal{U}}_{j-1}^{(2)}) - 14009(\vec{\mathcal{U}}_{j-1}^{(3)} + \vec{\mathcal{U}}_{j+1}^{(3)}) \end{pmatrix}}{66624768} \\ \dfrac{\begin{pmatrix} 1162\vec{\mathcal{U}}_j^{(0)} + 1246\vec{\mathcal{U}}_j^{(2)} - 581(\vec{\mathcal{U}}_{j-1}^{(0)} + \vec{\mathcal{U}}_{j+1}^{(0)}) + 553(\vec{\mathcal{U}}_{j+1}^{(1)} - \vec{\mathcal{U}}_{j-1}^{(1)}) - \\ -455(\vec{\mathcal{U}}_{j-1}^{(2)} + \vec{\mathcal{U}}_{j+1}^{(2)}) + 243(\vec{\mathcal{U}}_{j+1}^{(3)} - \vec{\mathcal{U}}_{j-1}^{(3)}) \end{pmatrix}}{20093184} \\ \dfrac{\begin{pmatrix} 5294\vec{\mathcal{U}}_j^{(1)} + 7234\vec{\mathcal{U}}_j^{(3)} + 2375(\vec{\mathcal{U}}_{j-1}^{(0)} - \vec{\mathcal{U}}_{j+1}^{(0)}) + 2103(\vec{\mathcal{U}}_{j-1}^{(1)} + \vec{\mathcal{U}}_{j+1}^{(1)}) + \\ +1513(\vec{\mathcal{U}}_{j-1}^{(2)} - \vec{\mathcal{U}}_{j+1}^{(2)}) + 673(\vec{\mathcal{U}}_{j-1}^{(3)} + \vec{\mathcal{U}}_{j+1}^{(3)}) \end{pmatrix}}{180838656} \end{bmatrix} \tag{3.41}$$

[6]On irregular grid, we dropped the accuracy of recovery to the 8th order, i.e. $\vec{\mathcal{U}}_j^{(n),R} = 0,\ n = 8, ..., 11.$

Remarks:

1. Eq.(3.33) is exactly unlimited FV-PPM [CW84].

2. The stencil of cRDG of any order is compact and exactly the same as that of FV-PPM (i.e., involving only von Neumann neighbors).

3. Recovery operation provides a mean for local spatial error estimator, which can be utilized in uncertainty quantifications and useful in Adaptive Model Refinement (AMoR) and tagging for adaptive hp-refinements of meshes .

4. Recovered DoFs is the auxiliary information, which is utilized to compute residuals in each cell. No extra memory is necessary to store $\vec{\mathcal{U}}_{\mathrm{j}}^{(n),\mathcal{R}}$, $n = p+1, ..., R$, as they are computed locally, used for computation of local numerical hyperbolic fluxes, integral term, and inter-cell (interface) recovery iRDG of the diffusion operator, and then disposed.

5. The cost of the in-cell recovery is very insignificant, compared to the most CPU-consuming computation of numerical hyperbolic fluxes.

6. In some boundary conditions, it is necessary to utilize "sided" recovery, i.e. dropping some weak statements either from the left or from right cell. In this case, the order of accuracy is dropped. For example, using "recovery-from-the-left", the following DoFs can be constructed on uniform mesh:

<u>cRDG$_0$</u>, $O\left(\Delta x^2\right)$:

$$\left[\begin{array}{c} \vec{\mathcal{U}}_{\mathrm{j}}^{(1),\mathcal{R}} \end{array}\right] = \left[\begin{array}{c} \frac{\vec{\mathcal{U}}_{\mathrm{j}}^{(0)} - \vec{\mathcal{U}}_{\mathrm{j}-1}^{(0)}}{2} \end{array}\right] \tag{3.42}$$

<u>cRDG$_1$</u>, $O\left(\Delta x^4\right)$:

$$\left[\begin{array}{c} \vec{\mathcal{U}}_{\mathrm{j}}^{(2),\mathcal{R}} \\ \\ \vec{\mathcal{U}}_{\mathrm{j}}^{(3),\mathcal{R}} \end{array}\right] = \left[\begin{array}{c} \frac{-15\vec{\mathcal{U}}_{\mathrm{j}}^{(0)} + 15\vec{\mathcal{U}}_{\mathrm{j}-1}^{(0)} + 19\vec{\mathcal{U}}_{\mathrm{j}}^{(1)} + 11\vec{\mathcal{U}}_{\mathrm{j}-1}^{(1)}}{24} \\ \\ \frac{-\vec{\mathcal{U}}_{\mathrm{j}}^{(0)} + \vec{\mathcal{U}}_{\mathrm{j}-1}^{(0)} + \vec{\mathcal{U}}_{\mathrm{j}}^{(1)} + \vec{\mathcal{U}}_{\mathrm{j}-1}^{(1)}}{8} \end{array}\right] \tag{3.43}$$

<u>cRDG$_2$</u>, $O\left(\Delta x^6\right)$:

$$\begin{bmatrix} \vec{\mathcal{U}}_{\mathrm{j}}^{(3),\mathcal{R}} \\ \\ \vec{\mathcal{U}}_{\mathrm{j}}^{(4),\mathcal{R}} \\ \\ \vec{\mathcal{U}}_{\mathrm{j}}^{(5),\mathcal{R}} \end{bmatrix} = \begin{bmatrix} \frac{800\vec{\mathcal{U}}_{\mathrm{j}}^{(0)} - 800\vec{\mathcal{U}}_{\mathrm{j}-1}^{(0)} - 905\vec{\mathcal{U}}_{\mathrm{j}}^{(1)} - 695\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)} + 1051\vec{\mathcal{U}}_{\mathrm{j}}^{(2)} - 421\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)}}{960} \\ \\ \frac{120\vec{\mathcal{U}}_{\mathrm{j}}^{(0)} - 120\vec{\mathcal{U}}_{\mathrm{j}-1}^{(0)} - 125\vec{\mathcal{U}}_{\mathrm{j}}^{(1)} - 115\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)} + 111\vec{\mathcal{U}}_{\mathrm{j}}^{(2)} - 81\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)}}{320} \\ \\ \frac{5\vec{\mathcal{U}}_{\mathrm{j}}^{(0)} - 5\vec{\mathcal{U}}_{\mathrm{j}-1}^{(0)} - 5\vec{\mathcal{U}}_{\mathrm{j}}^{(1)} - 5\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)} + 4\vec{\mathcal{U}}_{\mathrm{j}}^{(2)} - 4\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)}}{120} \end{bmatrix} \tag{3.44}$$

<u>cRDG$_3$</u>, $O\left(\Delta x^6\right)$:

$$\begin{bmatrix} \vec{\mathcal{U}}_{\mathrm{j}}^{(4),\mathcal{R}} \\ \\ \vec{\mathcal{U}}_{\mathrm{j}}^{(5),\mathcal{R}} \\ \\ \vec{\mathcal{U}}_{\mathrm{j}}^{(6),\mathcal{R}} \\ \\ \vec{\mathcal{U}}_{\mathrm{j}}^{(7),\mathcal{R}} \end{bmatrix} = \begin{bmatrix} \frac{\begin{pmatrix} -11319\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(0)} - \vec{\mathcal{U}}_{\mathrm{j}-1}^{(0)}\right) + 12215\vec{\mathcal{U}}_{\mathrm{j}}^{(1)} + 10423\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)} - \\ -13545\vec{\mathcal{U}}_{\mathrm{j}}^{(2)} + 8169\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)} + 13809\vec{\mathcal{U}}_{\mathrm{j}}^{(3)} + 4209\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)} \end{pmatrix}}{9856} \\ \\ \frac{\begin{pmatrix} -9653\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(0)} - \vec{\mathcal{U}}_{\mathrm{j}-1}^{(0)}\right) + 10017\vec{\mathcal{U}}_{\mathrm{j}}^{(1)} + 9289\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)} - \\ -10031\vec{\mathcal{U}}_{\mathrm{j}}^{(2)} + 7847\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)} + 7747\vec{\mathcal{U}}_{\mathrm{j}}^{(3)} + 4419\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)} \end{pmatrix}}{11648} \\ \\ \frac{\begin{pmatrix} -1925\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(0)} - \vec{\mathcal{U}}_{\mathrm{j}-1}^{(0)}\right) + 1953\vec{\mathcal{U}}_{\mathrm{j}}^{(1)} + 1897\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)} - \\ -1855\vec{\mathcal{U}}_{\mathrm{j}}^{(2)} + 1687\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)} + 1283\vec{\mathcal{U}}_{\mathrm{j}}^{(3)} + 1027\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)} \end{pmatrix}}{9856} \\ \\ \frac{\begin{pmatrix} -25\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(0)} - \vec{\mathcal{U}}_{\mathrm{j}-1}^{(0)}\right) + 25\vec{\mathcal{U}}_{\mathrm{j}}^{(1)} + 25\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)} - \\ -23\vec{\mathcal{U}}_{\mathrm{j}}^{(2)} + 23\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)} + 15\vec{\mathcal{U}}_{\mathrm{j}}^{(3)} + 15\vec{\mathcal{U}}_{\mathrm{j}-1}^{2)} \end{pmatrix}}{1664} \end{bmatrix} \tag{3.45}$$

<u>**Note:**</u>

In the code, the minimum (default) number of Gaussian quadrature points for in-cell recovery DG are: 6 for cRDG$_1$, 8 for cRDG$_2$ and 10 for cRDG$_3$. This set ensures that the errors of the *Gauss-Chebyshev* integration [PTVF99] is smaller than the errors from in-cell recovery, in the case of combined in-cell and inter-cell recoveries (inter-cell recovery is more demanding). It is also instructive to note that, if only the in-cell recovery were applied, the minimum set of quadrature points would be: 4, 6 and 8 for cRDG$_1$, cRDG$_2$ and cRDG$_3$, respectively.

Limiting

Limiting is necessary in the presence of strong discontinuities (e.g., shocks, contacts). There is a number of limiting schemes available in R$_7$. We summarized them below.

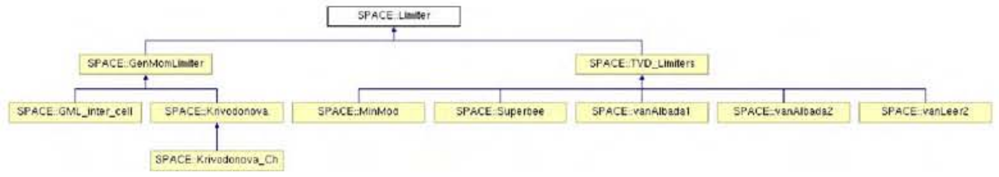

Fig. 3.11 : Hierarchical tree of the Class `Limiter`.

Finite Volume, cRDG$_0$. In the case of finite-volume discretization (Figure 3.7), we implemented TVD limiters [Tor99], Figure 3.11. In terms of cRDG's DoF, these limiters can be written as:

$$\begin{bmatrix} \vec{\mathcal{U}}_j^{(1),R} \\ \\ \vec{\mathcal{U}}_j^{(2),R} \end{bmatrix} = \begin{bmatrix} \frac{\vec{\mathcal{U}}^L_{j+\frac{1}{2}} - \vec{\mathcal{U}}^R_{j-\frac{1}{2}}}{2} \\ \\ \frac{\vec{\mathcal{U}}^L_{j+\frac{1}{2}} - 2\vec{\mathcal{U}}_j^{(0)} + \vec{\mathcal{U}}^R_{j-\frac{1}{2}}}{2} \end{bmatrix} \tag{3.46}$$

where[7]

$$\begin{aligned} \vec{\mathcal{U}}^L_{j+\frac{1}{2}} &= \vec{\mathcal{U}}_j^{(0)} + \frac{\Delta^+(1+\kappa^+)\Gamma\left(\Delta^-,\Delta^+\right) + \Delta^-(1-\kappa^+)\Gamma\left(\Delta^+,\Delta^-\right)}{4} \\ \vec{\mathcal{U}}^R_{j-\frac{1}{2}} &= \vec{\mathcal{U}}_j^{(0)} - \frac{\Delta^-(1+\kappa^-)\Gamma\left(\Delta^+,\Delta^-\right) + \Delta^+(1-\kappa^-)\Gamma\left(\Delta^-,\Delta^+\right)}{4} \end{aligned} \tag{3.47}$$

$\kappa^+ = \frac{\Delta x_j + \Delta x_{j-1} - \Delta x_{j+1}}{\Delta x_j + \Delta x_{j-1} + \Delta x_{j+1}}$, $\kappa^- = \frac{\Delta x_j - \Delta x_{j-1} + \Delta x_{j+1}}{\Delta x_j + \Delta x_{j-1} + \Delta x_{j+1}}$, $\Delta^+ = 2\Delta x_j \frac{\vec{\mathcal{U}}_{j+1}^{(0)} - \vec{\mathcal{U}}_j^{(0)}}{\Delta x_j + \Delta x_{j+1}}$, $\Delta^- = 2\Delta x_j \frac{\vec{\mathcal{U}}_j^{(0)} - \vec{\mathcal{U}}_{j-1}^{(0)}}{\Delta x_j + \Delta x_{j-1}}$ and the limiter $\Gamma(a,b)$ is defined by one of the following five options:

Class `vanAlbada1`. First Van Albada limiter is

$$\Gamma(a,b) = \frac{a^2 + ab}{a^2 + b^2 + \epsilon} \tag{3.48}$$

[7]On uniform mesh $\Delta x_j = \Delta x_{j-1} = \Delta x_{j+1}$ and $\kappa^+ = \kappa^- = \frac{1}{3}$.

Class `vanAlbada2`. Second Van Albada limiter is

$$\Gamma\left(a,b\right) = \frac{2ab}{a^2 + b^2 + \epsilon} \tag{3.49}$$

Class `MinMod`.

$$\begin{aligned} \Gamma\left(a,b\right) &= \max\left(0, \min\left(1, \frac{a}{r}\right)\right), \\ \text{where } r &= \begin{cases} \text{if } (|b| < \epsilon) & \epsilon \cdot \mathrm{Sign}\left(b\right) \\ \text{else} & b \end{cases} \end{aligned} \tag{3.50}$$

Class `vanLeer2`. Van Leer limiter is

$$\Gamma\left(a,b\right) = \begin{cases} \text{if } (|b+r| < \epsilon) & \frac{a+r}{\epsilon} \\ \text{else} & \frac{a+r}{b+r} \end{cases} \quad \text{where } r = |a|\,\mathrm{Sign}\left(b\right) \tag{3.51}$$

Class `Superbee`.

$$\begin{aligned} \Gamma\left(a,b\right) &= \max\left(0, \max\left(\min\left(1, \frac{2a}{r}\right), \min\left(2, \frac{a}{r}\right)\right)\right), \\ \text{where } r &= \begin{cases} \text{if } (|b| < \epsilon) & \epsilon \cdot \mathrm{Sign}\left(b\right) \\ \text{else} & b \end{cases} \end{aligned} \tag{3.52}$$

In eqs.(3.48-3.52), $\epsilon = 10^{-10}$.

In addition to TVD limiters, we also incorporated WENO$_3$ limiting [JS96] into the cRDG$_0$, as described below.

Classes `uni_PPM_WENO` and `irr_PPM_WENO`. Limited DoFs are defined by eq.(3.46), where

$$\begin{aligned} \vec{\mathcal{U}}^{\mathrm{L}}_{\mathrm{j}+\frac{1}{2}} &= \frac{\alpha_{\mathrm{R}}^{(\mathrm{L})}\mathfrak{P}_{\mathrm{R}}^{(\mathrm{L})} + \alpha_{\mathrm{L}}^{(\mathrm{L})}\mathfrak{P}_{\mathrm{L}}^{(\mathrm{L})}}{\alpha_{\mathrm{R}}^{(\mathrm{L})} + \alpha_{\mathrm{L}}^{(\mathrm{L})}} \\ \vec{\mathcal{U}}^{\mathrm{R}}_{\mathrm{j}-\frac{1}{2}} &= \frac{\alpha_{\mathrm{R}}^{(\mathrm{R})}\mathfrak{P}_{\mathrm{R}}^{(\mathrm{R})} + \alpha_{\mathrm{L}}^{(\mathrm{R})}\mathfrak{P}_{\mathrm{L}}^{(\mathrm{R})}}{\alpha_{\mathrm{R}}^{(\mathrm{R})} + \alpha_{\mathrm{L}}^{(\mathrm{R})}} \end{aligned} \tag{3.53}$$

$$\begin{aligned}\alpha^{(R)}_{L} &= \frac{\left(\Delta x_j+\Delta x_{j+1}\right)/\mathsf{IS}^{(L)}}{\Delta x_j+\Delta x_{j-1}+\Delta x_{j+1}}\\ \alpha^{(L)}_{R} &= \frac{\left(\Delta x_j+\Delta x_{j-1}\right)/\mathsf{IS}^{(R)}}{\Delta x_j+\Delta x_{j-1}+\Delta x_{j+1}}\end{aligned} \tag{3.54}$$

$$\begin{aligned}\alpha^{(R)}_{R} &= \frac{\Delta x_{j-1}/\mathsf{IS}^{(R)}}{\Delta x_j+\Delta x_{j-1}+\Delta x_{j+1}}\\ \alpha^{(L)}_{L} &= \frac{\Delta x_{j+1}/\mathsf{IS}^{(L)}}{\Delta x_j+\Delta x_{j-1}+\Delta x_{j+1}}\end{aligned} \tag{3.55}$$

$$\begin{aligned}\mathsf{IS}^{(L)} &= \left(\frac{4\Delta x_j^2}{\left(\Delta x_j+\Delta x_{j-1}\right)^2}\left(\vec{\mathcal{U}}^{(0)}_j-\vec{\mathcal{U}}^{(0)}_{j-1}\right)^2+\varepsilon_{\text{WENO}}\right)^2\\ \mathsf{IS}^{(R)} &= \left(\frac{4\Delta x_j^2}{\left(\Delta x_j+\Delta x_{j+1}\right)^2}\left(\vec{\mathcal{U}}^{(0)}_j-\vec{\mathcal{U}}^{(0)}_{j+1}\right)^2+\varepsilon_{\text{WENO}}\right)^2\end{aligned} \tag{3.56}$$

$$\begin{aligned}\mathfrak{P}^{(L)}_{R} &= \frac{\Delta x_{j+1}\vec{\mathcal{U}}^{(0)}_j+\Delta x_j\vec{\mathcal{U}}^{(0)}_{j+1}}{\Delta x_j+\Delta x_{j+1}}\\ \mathfrak{P}^{(R)}_{L} &= \frac{\Delta x_{j-1}\vec{\mathcal{U}}^{(0)}_j+\Delta x_j\vec{\mathcal{U}}^{(0)}_{j-1}}{\Delta x_j+\Delta x_{j-1}}\end{aligned} \tag{3.57}$$

$$\begin{aligned}\mathfrak{P}^{(L)}_{L} &= \vec{\mathcal{U}}^{(0)}_j+\frac{\Delta x_j}{\Delta x_j+\Delta x_{j-1}}\left(\vec{\mathcal{U}}^{(0)}_j-\vec{\mathcal{U}}^{(0)}_{j-1}\right)\\ \mathfrak{P}^{(R)}_{R} &= \vec{\mathcal{U}}^{(0)}_j+\frac{\Delta x_j}{\Delta x_j+\Delta x_{j+1}}\left(\vec{\mathcal{U}}^{(0)}_j-\vec{\mathcal{U}}^{(0)}_{j+1}\right)\end{aligned} \tag{3.58}$$

and $\varepsilon_{\text{WENO}} = 10^{-8}$.

<u>Generalized moment limiter, cRDG$_n$</u> (Class `Krivodonova`). In the case of the general $(R+1)^{\text{th}}$-order cRDG, the limiting can be applied using the generalization of the *moment limiter*, originally introduced by Krivodonova in [Kri07]. The procedure is applied at the end of each Runge-Kutta stage, and can be described as follows.

1. First, all DoFs at the cell $_{(\mathrm{j})}$ are transformed into the characteristic space, $\vec{\Psi}_{\mathrm{j}}^{(k)} = \mathbb{L}\ \vec{\mathcal{U}}_{\mathrm{j}}^{(k)}$, $k = 0, ..., R$, where $\mathbb{L}$ is the left eigenvector matrix of the Jacobian, evaluated at the solution state at the cell center, $\vec{\mathcal{U}}_{\mathrm{j}}^{\mathcal{R}}\left(x_{\mathrm{j}}\right)$.

2. Next, the limiting is started from $k = R, ..., 1$, as

$$\vec{\Psi}_{\mathrm{j}}^{L,(k)} = \mathsf{MinMod}\left(\begin{array}{c} \vec{\Psi}_{\mathrm{j}}^{(k)},\quad \underbrace{\left(\vec{\Psi}_{\mathrm{j+1}}^{(k-1)} - \vec{\Psi}_{\mathrm{j}}^{(k-1)}\right)\left(\dfrac{2\Delta x_{\mathrm{j}}}{\Delta x_{\mathrm{j}} + \Delta x_{\mathrm{j+1}}}\right)^{k}}_{\Delta_{\mathrm{j}}^{+,(k-1)}}, \\ \underbrace{\left(\vec{\Psi}_{\mathrm{j}}^{(k-1)} - \vec{\Psi}_{\mathrm{j-1}}^{(k-1)}\right)\left(\dfrac{2\Delta x_{\mathrm{j}}}{\Delta x_{\mathrm{j}} + \Delta x_{\mathrm{j-1}}}\right)^{k}}_{\Delta_{\mathrm{j}}^{-,(k-1)}} \end{array}\right) \tag{3.59}$$

   - If $\left|\vec{\Psi}_{\mathrm{j}}^{L,(k)} - \vec{\Psi}_{\mathrm{j}}^{(k)}\right| < 10^{-14}$, the limiting of the particular characteristic field is stopped.
   - Otherwise, set $\vec{\Psi}_{\mathrm{j}}^{(k)} = \vec{\Psi}_{\mathrm{j}}^{L,(k)}$, and continue limiting, until $k = 1$ is reached.

3. Finally, compute limited DoFs in physical space as $\vec{\mathcal{U}}_{\mathrm{j}}^{(k)} = \mathbb{R}\ \vec{\Psi}_{\mathrm{j}}^{(k)}$, $k = 0, ..., R$, where $\mathbb{R}$ is the right eigenvector matrix of the Jacobian.

MinMod function in eq.(3.59) is defined as

$$\mathsf{MinMod}\left(a, b, c\right) = \begin{cases} \mathrm{Sgn}\left(a\right)\min\left(\left|a\right|, \left|b\right|, \left|c\right|\right) & \text{if } \mathrm{Sgn}\left(a\right) = \mathrm{Sgn}\left(b\right) = \mathrm{Sgn}\left(c\right) \\ 0 & \text{otherwise} \end{cases} \tag{3.60}$$

In the limiting case of $n = 0$ ($R = 3$), eq.(3.59) reduces to the MinMod limiter [Tor99, Kri07].

Hyperbolic flux treatment algorithms

First, we compute the “Left” and the “Right” states at each edge $_{(\mathrm{j}+\frac{1}{2})}$,

$$\vec{\mathcal{U}}^{\mathrm{L}}_{\mathrm{j}+\frac{1}{2}} = \sum_{n=0}^{R} \vec{\mathcal{U}}^{(n),\mathcal{R}}_{\mathrm{j}} \, \mathfrak{L}_{(n)}(1) \quad \text{and} \quad \vec{\mathcal{U}}^{\mathrm{R}}_{\mathrm{j}+\frac{1}{2}} = \sum_{n=0}^{R} \vec{\mathcal{U}}^{(n),\mathcal{R}}_{\mathrm{j}+1} \, \mathfrak{L}_{(n)}(-1) \tag{3.61}$$

Then, numerical fluxes $\mathbf{H}_{\mathrm{j}\pm\frac{1}{2}}$ of the hyperbolic operator $\vec{\mathcal{H}}$ are computed using one of the six available flux treatment schemes, Fig.3.12.

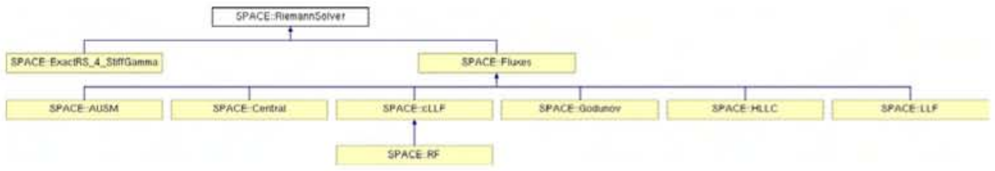

Fig. 3.12 : Hierarchical tree of the Class `RiemannSolver`.

I. Local Lax Friedrichs (Class `LLF`). This is the simplest flux treatment scheme, which requires only the knowledge of system's eigenvalues. LLF fluxes are simply computed as:

$$\mathbf{H}_{\mathrm{j}\pm\frac{1}{2}} = \frac{1}{2}\left[\vec{\mathcal{H}}\left(\vec{\mathcal{U}}^{\mathrm{L}}_{\mathrm{j}+\frac{1}{2}}\right) + \vec{\mathcal{H}}\left(\vec{\mathcal{U}}^{\mathrm{R}}_{\mathrm{j}+\frac{1}{2}}\right) + \alpha\lambda_{\max}\left(\vec{\mathcal{U}}^{\mathrm{L}}_{\mathrm{j}+\frac{1}{2}} - \vec{\mathcal{U}}^{\mathrm{R}}_{\mathrm{j}+\frac{1}{2}}\right)\right] \tag{3.62}$$

Where $\lambda_{\max}$ is the maximum absolute value of eigenvalues, computed at $\vec{\mathcal{U}}^{\mathrm{L}}_{\mathrm{j}+\frac{1}{2}}$ and $\vec{\mathcal{U}}^{\mathrm{R}}_{\mathrm{j}+\frac{1}{2}}$ states. Parameter α is usually set to 1 in shock-dynamics problems, and $\alpha \sim 0.1$ in low-Mach number applications.

II. Characteristic Local Lax Friedrichs (Class `cLLF`). This is an extension of the above simple form of LLF to the characteristic space. First, left and right states and corresponding fluxes are transformed into characteristic space,

$$\begin{aligned} \vec{\phi}^{\mathrm{L}} &= \mathbb{L}\,\vec{\mathcal{U}}^{\mathrm{L}}_{\mathrm{j}+\frac{1}{2}}; \qquad & \vec{\Phi}^{\mathrm{L}} &= \mathbb{L}\,\vec{\mathcal{H}}\left(\vec{\mathcal{U}}^{\mathrm{L}}_{\mathrm{j}+\frac{1}{2}}\right) \\ \vec{\phi}^{\mathrm{R}} &= \mathbb{L}\,\vec{\mathcal{U}}^{\mathrm{R}}_{\mathrm{j}+\frac{1}{2}}; \qquad & \vec{\Phi}^{\mathrm{R}} &= \mathbb{L}\,\vec{\mathcal{H}}\left(\vec{\mathcal{U}}^{\mathrm{R}}_{\mathrm{j}+\frac{1}{2}}\right) \end{aligned} \tag{3.63}$$

Then, LLF is applied in each $_{\mathrm{f}}$ of $_{\mathrm{NE}}$ fields, $_{\mathrm{f}=1,\dots,\mathrm{NE}}$:

$$\Phi_{\mathrm{f},\mathrm{j}\pm\frac{1}{2}} = \frac{1}{2}\left(\Phi_{\mathrm{f}}^{\mathrm{L}} + \Phi_{\mathrm{f}}^{\mathrm{R}} + \alpha\lambda_{\mathrm{f,max}}\left(\phi_{\mathrm{f}}^{\mathrm{L}} - \phi_{\mathrm{f}}^{\mathrm{R}}\right)\right) \tag{3.64}$$

In this case, $\lambda_{\mathrm{f,max}}$ is the maximum eigenvalue of the f^{th} field, computed at the "Left" and "Right" states. Once characteristic fluxes for all fields are computed, the physical fluxes are recovered as $\vec{\mathbb{H}}_{\mathrm{j}\pm\frac{1}{2}} = \mathbb{R}\,\vec{\Phi}_{\mathrm{j}\pm\frac{1}{2}}$. Eigensystems $[\mathbb{L},\mathbb{R},\Lambda]$ are computed at the edge "average" state, $\vec{\mathcal{U}}_{\mathrm{e}} = \sum_{n=0}^{p} \frac{\vec{\mathcal{U}}_{\mathrm{j}}^{(n)} + \vec{\mathcal{U}}_{\mathrm{j}+1}^{(n)}}{2}\,\mathfrak{L}_{(n)}(0)$, where p is the number of the evolved DoFs in DG.

III. Central differencing (Class `Central`). This option is not really recommended in general computations, as produces unstable results in many configurations, since it does not have numerical diffusion:

$$\mathbf{H}_{\mathrm{j}\pm\frac{1}{2}} = \frac{1}{2}\left[\vec{\mathcal{H}}\left(\vec{\mathcal{U}}^{\mathrm{L}}_{\mathrm{j}+\frac{1}{2}}\right) + \vec{\mathcal{H}}\left(\vec{\mathcal{U}}^{\mathrm{R}}_{\mathrm{j}+\frac{1}{2}}\right)\right] \tag{3.65}$$

IV. Roe-Fix (Class `RF`). This is a variation [SO89] of the above-described cLLF scheme, in which eq.(3.64) is replaced by

$$\Phi_{\mathrm{f},\mathrm{j}\pm\frac{1}{2}} = \begin{cases} \Phi_{\mathrm{f}}^{\mathrm{L}} & \text{if } \left(\lambda_{\mathrm{f}}^{\mathrm{L}} > 0 \ \&\ \lambda_{\mathrm{f}}^{\mathrm{R}} > 0\right) \\ \Phi_{\mathrm{f}}^{\mathrm{R}} & \text{if } \left(\lambda_{\mathrm{f}}^{\mathrm{L}} < 0 \ \&\ \lambda_{\mathrm{f}}^{\mathrm{R}} < 0\right) \\ \text{Apply } \mathsf{cLLF} & \text{otherwise} \end{cases} \tag{3.66}$$

V. Godunov (Class `Godunov`). First, exact Riemann solver is applied to the jump $\left[\vec{\mathcal{U}}^{\mathrm{L}}_{\mathrm{j}+\frac{1}{2}}, \vec{\mathcal{U}}^{\mathrm{R}}_{\mathrm{j}+\frac{1}{2}}\right]$ [GZI+76, Tor99]. This gives a state predicted at the new time, $\vec{\mathcal{U}}^{\mathrm{new}}_{\mathrm{j}+\frac{1}{2}}\left(t^{(n+1)}\right)$. Numerical flux is computed from this state, $\vec{\mathbb{H}}_{\mathrm{j}\pm\frac{1}{2}} = \vec{\mathcal{H}}\left(\vec{\mathcal{U}}^{\mathrm{new}}_{\mathrm{j}+\frac{1}{2}}\right)$.

VI. AUSM (Class `AUSM`). AUSM stands for Advection Upstream Splitting Method. It is developed as a numerical inviscid flux function for solving a general system of conservation equations. The AUSM first recognizes that the inviscid flux consist of two physically distinct parts, i.e., convective and pressure fluxes. The former is associated with the flow (advection) speed, while the latter with the acoustic speed; or respectively classified as the linear and nonlinear fields. Currently, the convective and pressure fluxes are formulated using the eigenvalues of the flux

Jacobian matrices. We implemented the latest version AUSM$^{+\mathrm{up}}$ as described in Dr. Liou's recent paper [Lio06].

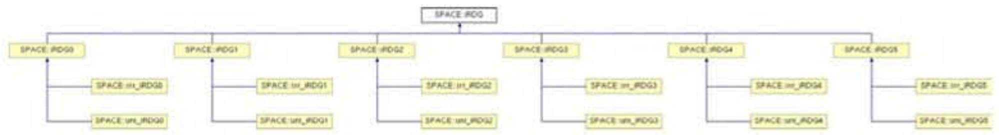

Fig. 3.13 : Hierarchical tree of the Class `iRDG`.

Diffusion fluxes: Inter-cell recovery (iRDG)

Numerical diffusion fluxes $\mathbf{D}_{\mathrm{j}\pm\frac{1}{2}}$ at cell edges are computed from the "recovered" inter-cell continuous solution profiles [vLN05, vLLR07, vRvL08, vLL09, LvL09]. An $(N+1)^{\mathrm{th}}$-order continuous "recovery" profile $\vec{\mathcal{V}}^{\mathcal{R}}_{\mathrm{j}-\frac{1}{2}}(x)$ between cells $_{[\mathrm{j}-1]}$ and $_{[\mathrm{j}]}$ is "recovered" from the in-cell (cRDG-recovered) discontinuous solutions $\vec{\mathcal{U}}^{\mathcal{R}}_{h_{\mathrm{j}}}(x,t)$ and $\vec{\mathcal{U}}^{\mathcal{R}}_{h_{\mathrm{j}-1}}(x,t)$ eq.(3.30) as

$$\vec{\mathcal{V}}^{\mathcal{R}}_{\mathrm{j}-\frac{1}{2}}(x) = \sum_{k=0}^{N} \vec{\mathcal{V}}^{(k),\mathcal{R}}_{\mathrm{j}-\frac{1}{2}} \mathfrak{H}_{(k)}\left(\zeta_{\mathrm{j}-\frac{1}{2}}\right) \tag{3.67}$$

where $\zeta_{\mathrm{j}-\frac{1}{2}} \equiv x - x_{\mathrm{j}-\frac{1}{2}}$. $\vec{\mathcal{V}}^{(k),\mathcal{R}}_{\mathrm{j}-\frac{1}{2}}$ are recovered in a weak sense, using the following van Leer et al. "recovery" constraints [vLN05]:

$$\int_{\mathcal{I}_{\mathrm{m}}} \mathfrak{L}_{(n)}(\xi_{\mathrm{m}}) \vec{\mathcal{U}}^{\mathcal{R}}_{\mathcal{I}_{\mathrm{m}}}(x)\, dx = \int_{\mathcal{I}_{\mathrm{m}}} \mathfrak{L}_{(n)}(\xi_{\mathrm{m}}) \vec{\mathcal{V}}^{\mathcal{R}}_{\mathrm{j}-\frac{1}{2}}(x)\, dx \qquad \text{where } \mathrm{m}{=}\mathrm{j},\mathrm{j}{-}1 \text{ and } n{=}0,\ldots,R \tag{3.68}$$

which corresponds to $(N+1)$ equations solved for unknowns $\vec{\mathcal{V}}^{(k=0,\ldots,N),\mathcal{R}}_{\mathrm{j}-\frac{1}{2}}$.

Then, numerical diffusion fluxes are defined as

$$\mathbf{D}_{\mathrm{j}\pm\frac{1}{2}} \equiv \underbrace{\mathbb{D}\left(\vec{\mathcal{V}}_{\mathrm{j}-\frac{1}{2}}(0)\right)}_{\text{Diffusion coeff. matrix}} \underbrace{\frac{d}{dx}\vec{\mathcal{V}}_{\mathrm{j}-\frac{1}{2}}(0)}_{\vec{\mathcal{V}}'_{\mathrm{j}-\frac{1}{2}}(0)} \tag{3.69}$$

requiring $\vec{\mathcal{V}}_{j-\frac{1}{2}}(0)$ and $\vec{\mathcal{V}}'_{j-\frac{1}{2}}(0)$, which are computed from the recovered profiles eq.(3.67) and given below[8] (see Figure 3.13).

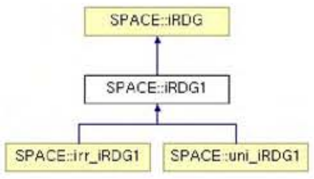

Fig. 3.14 : Hierarchical tree of the Class `iRDG1`.

PIECEWISE-CONSTANT[9], $\mathsf{iRDG}_0^{(4)}$ (Figure 3.14):

Regular grid, Class `uni_iRDG1`:

$$\begin{aligned}\vec{\mathcal{V}}_{j-\frac{1}{2}} &= \frac{3\left(\vec{\mathcal{U}}_{j}^{(0)}+\vec{\mathcal{U}}_{j-1}^{(0)}\right)+2\left(\vec{\mathcal{U}}_{j-1}^{(1)}-\vec{\mathcal{U}}_{j}^{(1)}\right)}{6}\\ \vec{\mathcal{V}}'_{j-\frac{1}{2}} &= -\frac{9\left(\vec{\mathcal{U}}_{j-1}^{(0)}-\vec{\mathcal{U}}_{j}^{(0)}\right)+5\left(\vec{\mathcal{U}}_{j}^{(1)}+\vec{\mathcal{U}}_{j-1}^{(1)}\right)}{4\Delta x}\end{aligned}$$

Irregular grid, Class `irr_iRDG1`: (3.70)

$$\begin{aligned}\vec{\mathcal{V}}_{j-\frac{1}{2}} &= \frac{\left(\begin{array}{l}3\left(\left(\vec{\mathcal{U}}_{j-1}^{(0)}+\vec{\mathcal{U}}_{j-1}^{(1)}\right)\Delta x_j^3+\Delta x_{j-1}^3\left(\vec{\mathcal{U}}_{j}^{(0)}-\vec{\mathcal{U}}_{j}^{(1)}\right)\right)+\\ \quad+\Delta x_j\Delta x_{j-1}\left(\Delta x_{j-1}\left(9\vec{\mathcal{U}}_{j}^{(0)}-5\vec{\mathcal{U}}_{j}^{(1)}\right)+\Delta x_j\left(9\vec{\mathcal{U}}_{j-1}^{(0)}+5\vec{\mathcal{U}}_{j-1}^{(1)}\right)\right)\end{array}\right)}{3\left(\Delta x_j+\Delta x_{j-1}\right)^3}\\ \vec{\mathcal{V}}'_{j-\frac{1}{2}} &= \frac{2\left(\begin{array}{l}\Delta x_{j-1}^2\left(\left(9\left(\vec{\mathcal{U}}_{j}^{(0)}-\vec{\mathcal{U}}_{j-1}^{(0)}\right)-5\left(\vec{\mathcal{U}}_{j}^{(1)}+\vec{\mathcal{U}}_{j-1}^{(1)}\right)\right)\Delta x_j^2+\right.\\ \quad\left.+\Delta x_{j-1}\left(\Delta x_{j-1}-\Delta x_j\right)\vec{\mathcal{U}}_{j}^{(1)}\right)-\Delta x_j^3\left(\Delta x_{j-1}-\Delta x_j\right)\vec{\mathcal{U}}_{j-1}^{(1)}\end{array}\right)}{\Delta x_j\Delta x_{j-1}\left(\Delta x_j+\Delta x_{j-1}\right)^3}\end{aligned}$$

PIECEWISE-LINEAR[10], $\mathsf{iRDG}_1^{(6)}$ (Figure 3.15):

Regular grid, Class `uni_iRDG2`:

$$\begin{aligned}\vec{\mathcal{V}}_{j-\frac{1}{2}} &= \frac{16\left(\vec{\mathcal{U}}_{j}^{(0)}+\vec{\mathcal{U}}_{j-1}^{(0)}\right)+13\left(\vec{\mathcal{U}}_{j-1}^{(1)}-\vec{\mathcal{U}}_{j}^{(1)}\right)+7\left(\vec{\mathcal{U}}_{j}^{(2)}+\vec{\mathcal{U}}_{j-1}^{(2)}\right)}{32}\\ \vec{\mathcal{V}}'_{j-\frac{1}{2}} &= -\frac{75\left(\vec{\mathcal{U}}_{j-1}^{(0)}-\vec{\mathcal{U}}_{j}^{(0)}\right)+55\left(\vec{\mathcal{U}}_{j}^{(1)}+\vec{\mathcal{U}}_{j-1}^{(1)}\right)+24\left(\vec{\mathcal{U}}_{j-1}^{(2)}-\vec{\mathcal{U}}_{j}^{(2)}\right)}{20\Delta x}\end{aligned} \qquad (3.71)$$

[8]Here, we show only the *piecewise-constant* and *piecewise-linear* discretizations on irregular grids.

[9]Note that $\vec{\mathcal{U}}_{j,j-1}^{(n=0,1)}$ used here are those from the *in-cell recovery*, Section 3.3.1.

[10]Note that $\vec{\mathcal{U}}_{j,j-1}^{(n=0,1,2)}$ used here are those from the *in-cell recovery*, Section 3.3.1.

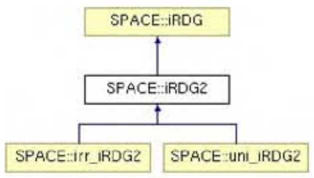

Fig. 3.15 : Hierarchical tree of the Class `iRDG2`.

Irregular grid, Class `irr_iRDG2`:

$$
\begin{aligned}
\vec{\mathcal{V}}_{j-\frac{1}{2}} &= \frac{\left(\begin{array}{l} 5(\vec{\mathcal{U}}^{(1)}_{\mathrm{j-1}} + \vec{\mathcal{U}}^{(2)}_{\mathrm{j-1}} + \vec{\mathcal{U}}^{(0)}_{\mathrm{j-1}}(5\xi_{\mathrm{j-1}}(2\xi_{\mathrm{j-1}}+1)+1)) + \xi_{\mathrm{j-1}}(16\vec{\mathcal{U}}^{(2)}_{\mathrm{j-1}} \\ +5\vec{\mathcal{U}}^{(1)}_{\mathrm{j-1}}(7\xi_{\mathrm{j-1}}+5) + \xi_{\mathrm{j-1}}(14\vec{\mathcal{U}}^{(2)}_{\mathrm{j-1}} + \xi_{\mathrm{j-1}}(-5\vec{\mathcal{U}}^{(1)}_{\mathrm{j}}(\xi_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}+5) \\ +7) + 5\vec{\mathcal{U}}^{(0)}_{\mathrm{j}}(\xi_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}+5)+10) + \vec{\mathcal{U}}^{(2)}_{\mathrm{j}}(\xi_{\mathrm{j-1}}(5\xi_{\mathrm{j-1}}+16)+14)))) \end{array}\right)}{5(\xi_{\mathrm{j-1}}+1)^5} \\
\vec{\mathcal{V}}'_{j-\frac{1}{2}} &= \frac{\left(\begin{array}{l} 2\left((-3\xi_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}(5\xi_{\mathrm{j-1}}+7)-16)\vec{\mathcal{U}}^{(2)}_{\mathrm{j}} + 84\vec{\mathcal{U}}^{(2)}_{\mathrm{j}} + 300(\vec{\mathcal{U}}^{(0)}_{\mathrm{j}} - \vec{\mathcal{U}}^{(0)}_{\mathrm{j-1}}) \right. \\ +5\vec{\mathcal{U}}^{(1)}_{\mathrm{j}}(\xi_{\mathrm{j-1}}+3)(\xi_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}+2)-14))\xi^3_{\mathrm{j-1}} - 5\vec{\mathcal{U}}^{(1)}_{\mathrm{j-1}}\left(\xi_{\mathrm{j-1}}\left(42\xi^2_{\mathrm{j-1}}\right.\right. \\ \left.\left.\left. +8\xi_{\mathrm{j-1}}-5\right)-1\right) - 3\vec{\mathcal{U}}^{(2)}_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}(4\xi_{\mathrm{j-1}}(7\xi_{\mathrm{j-1}}+4)-7)-5)\right) \end{array}\right)}{5\Delta x_{\mathrm{j}}\xi_{\mathrm{j-1}}(\xi_{\mathrm{j-1}}+1)^5}
\end{aligned}
\tag{3.72}
$$

where $\xi_{\mathrm{j-1}} = \frac{\Delta x_{\mathrm{j-1}}}{\Delta x_{\mathrm{j}}}$.

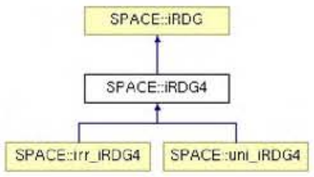

Fig. 3.16 : Hierarchical tree of the Class `iRDG4`.

PIECEWISE-QUADRATIC[11], $\mathsf{iRDG}_2^{(10)}$ (Figure 3.16).

(Regular grid), Class `uni_iRDG4`:

$$
\begin{aligned}
\vec{\mathcal{V}}_{j-\frac{1}{2}} &= \frac{\left(\begin{array}{c} 5376\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(0)}+\vec{\mathcal{U}}_{\mathrm{j-1}}^{(0)}\right)+4886\left(\vec{\mathcal{U}}_{\mathrm{j-1}}^{(1)}-\vec{\mathcal{U}}_{\mathrm{j}}^{(1)}\right)+3906\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(2)}+\vec{\mathcal{U}}_{\mathrm{j-1}}^{(2)}\right)+ \\ +2511\left(\vec{\mathcal{U}}_{\mathrm{j-1}}^{(3)}-\vec{\mathcal{U}}_{\mathrm{j}}^{(3)}\right)+1001\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(4)}+\vec{\mathcal{U}}_{\mathrm{j-1}}^{(4)}\right) \end{array}\right)}{10752} \\
\vec{\mathcal{V}}'_{j-\frac{1}{2}} &= \frac{\left(\begin{array}{c} 2835\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(0)}-\vec{\mathcal{U}}_{\mathrm{j-1}}^{(0)}\right)-2451\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(1)}+\vec{\mathcal{U}}_{\mathrm{j-1}}^{(1)}\right)+1773\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(2)}-\vec{\mathcal{U}}_{\mathrm{j-1}}^{(2)}\right)- \\ -981\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(3)}+\vec{\mathcal{U}}_{\mathrm{j-1}}^{(3)}\right)+320\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(4)}-\vec{\mathcal{U}}_{\mathrm{j-1}}^{(4)}\right) \end{array}\right)}{384\Delta x}
\end{aligned}
\tag{3.73}
$$

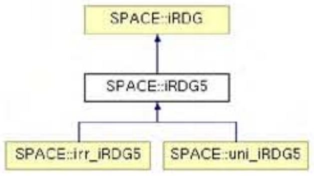

Fig. 3.17 : Hierarchical tree of the Class `iRDG5`.

[11]Note that $\vec{\mathcal{U}}_{\mathrm{j,j-1}}^{(n=0,..,4)}$ used here are those from the *in-cell recovery*, Section 3.3.1.

PIECEWISE-CUBIC[12], $\mathsf{iRDG}_3^{(12)}$ (Figure 3.17).

(Regular grid), Class `uni_iRDG5`:

$$
\begin{aligned}
\vec{\mathcal{V}}_{j-\frac{1}{2}} &= \frac{\left(\begin{array}{l} 8448\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(0)}+\vec{\mathcal{U}}_{\mathrm{j-1}}^{(0)}\right)+7854\left(-\vec{\mathcal{U}}_{\mathrm{j}}^{(1)}+\vec{\mathcal{U}}_{\mathrm{j-1}}^{(1)}\right)+6666\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(2)}+\vec{\mathcal{U}}_{\mathrm{j-1}}^{(2)}\right)+ \\ +4939\left(-\vec{\mathcal{U}}_{\mathrm{j}}^{(3)}+\vec{\mathcal{U}}_{\mathrm{j-1}}^{(3)}\right)+2893\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(4)}+\vec{\mathcal{U}}_{\mathrm{j-1}}^{(4)}\right)+1024\left(-\vec{\mathcal{U}}_{\mathrm{j}}^{(5)}+\vec{\mathcal{U}}_{\mathrm{j-1}}^{(5)}\right) \end{array}\right)}{16896} \\
\vec{\mathcal{V}}'_{j-\frac{1}{2}} &= \frac{\left(\begin{array}{l} 29106\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(0)}-\vec{\mathcal{U}}_{\mathrm{j-1}}^{(0)}\right)-26034\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(1)}+\vec{\mathcal{U}}_{\mathrm{j-1}}^{(1)}\right)+20457\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(2)}-\vec{\mathcal{U}}_{\mathrm{j-1}}^{(2)}\right)- \\ -13509\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(3)}+\vec{\mathcal{U}}_{\mathrm{j-1}}^{(3)}\right)+6793\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(4)}-\vec{\mathcal{U}}_{\mathrm{j-1}}^{(4)}\right)-1989\left(\vec{\mathcal{U}}_{\mathrm{j}}^{(5)}+\vec{\mathcal{U}}_{\mathrm{j-1}}^{(5)}\right) \end{array}\right)}{3072\Delta x}
\end{aligned}
\tag{3.74}
$$

Dirichlet Boundary Conditions

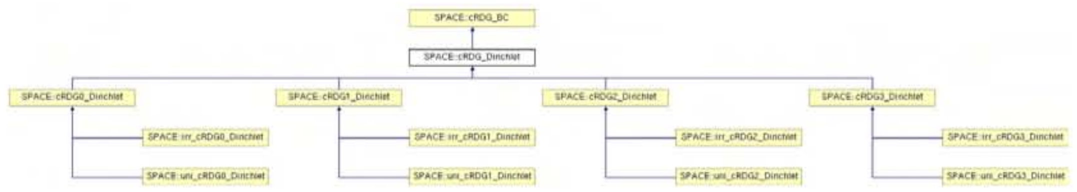
SPACE::cRDG_BC
SPACE::cRDG_Dirichlet
SPACE::cRDG0_Dirichlet
SPACE::cRDG1_Dirichlet
SPACE::cRDG2_Dirichlet
SPACE::cRDG3_Dirichlet
SPACE::irr_cRDG0_Dirichlet
SPACE::uni_cRDG0_Dirichlet
SPACE::irr_cRDG1_Dirichlet
SPACE::uni_cRDG1_Dirichlet
SPACE::irr_cRDG2_Dirichlet
SPACE::uni_cRDG2_Dirichlet
SPACE::irr_cRDG3_Dirichlet
SPACE::uni_cRDG3_Dirichlet

Fig. 3.18 : Hierarchical tree of the Class `cRDG_Dirichlet`.

Specification of boundary conditions for cRDG (and DG) requires special attention. This is because typically we have BCs for primitive variables, while solving DG evolution eq.(3.24) also requires boundary conditions for high-order DoFs – more than available in practical situations. We deal with this "under-specification" problem by specifying Neumann BCs for high-order DoFs, as follows[13]. Suppose, we are given a boundary value U_{BC} for a scalar $\mathcal{U}$.

cRDG$_0$. Forcing the in-cell recovered solution for the first real cell $_{(\mathrm{j})}$ to satisfy

$$
\begin{aligned}
\mathcal{U}_{\mathrm{j}}^{\mathcal{R}}\left(x_{\mathrm{j}-\frac{1}{2}}\right) &= \mathsf{U}_{\mathsf{BC}} \\
\left(\mathcal{U}_{\mathrm{j}}^{\mathcal{R}}\left(x_{\mathrm{j}-\frac{1}{2}}\right)\right)'' &= 0
\end{aligned}
\tag{3.75}
$$

[12]Note that $\vec{\mathcal{U}}_{\mathrm{j,j-1}}^{(n=0,..,5)}$ used here are those from the *in-cell recovery*, Section 3.3.1.

[13]For bravity, only left boundary and uniform mesh will be described. Extensions to the right boundary and irregular mesh are straightforward.

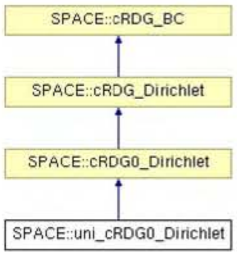

Fig. 3.19 : Hierarchical tree of the Class `uni_cRDG0_Dirichlet`.

results in:

$$\begin{aligned}\mathcal{U}^{(0)}_{j-1} &= \frac{6\mathsf{U}_{BC} - 5\mathcal{U}^{(0)}_{j} + \mathcal{U}^{(0)}_{j+1}}{2} \\ \mathcal{U}^{(0)}_{j-2} &= 6\left(\mathsf{U}_{BC} - \mathcal{U}^{(0)}_{j}\right) + \mathcal{U}^{(0)}_{j+1}\end{aligned} \tag{3.76}$$

The order of spatial discretization with this BC is $O\left(\Delta x^2\right)$.

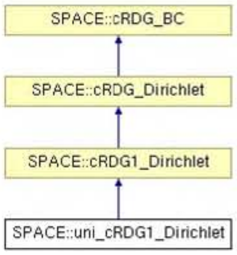

Fig. 3.20 : Hierarchical tree of the Class `uni_cRDG1_Dirichlet`.

cRDG$_1$. Forcing the in-cell recovered solution for the first ghost cell $_{(j-1)}$ and the first real cell $_{(j)}$ to satisfy

$$\begin{gathered}\mathcal{U}^{\mathcal{R}}_{j}\left(x_{j-\frac{1}{2}}\right) = \mathsf{U}_{BC} \\ \left(\mathcal{U}^{\mathcal{R}}_{j}\left(x_{j-\frac{1}{2}}\right)\right)^{(v)} = \left(\mathcal{U}^{\mathcal{R}}_{j-1}\left(x_{j-\frac{1}{2}}\right)\right)^{(v)} = \left(\mathcal{U}^{\mathcal{R}}_{j-1}\left(x_{j-\frac{1}{2}}\right)\right)^{(iv)} = 0\end{gathered} \tag{3.77}$$

results in

$$\begin{aligned}
\mathcal{U}^{(0)}_{\mathrm{j-1}} &= \frac{132\mathsf{U}_{\mathsf{BC}} - 77\mathcal{U}^{(0)}_{\mathrm{j}} + 177\mathcal{U}^{(1)}_{\mathrm{j}} - 47\mathcal{U}^{(0)}_{\mathrm{j+1}} + 33\mathcal{U}^{(1)}_{\mathrm{j+1}}}{8} \\
\mathcal{U}^{(1)}_{\mathrm{j-1}} &= \frac{-180\mathsf{U}_{\mathsf{BC}} + 105\mathcal{U}^{(0)}_{\mathrm{j}} - 269\mathcal{U}^{(1)}_{\mathrm{j}} + 75\mathcal{U}^{(0)}_{\mathrm{j+1}} - 53\mathcal{U}^{(1)}_{\mathrm{j+1}}}{8} \\
\mathcal{U}^{(0)}_{\mathrm{j-2}} &= \frac{492\mathsf{U}_{\mathsf{BC}} - 261\mathcal{U}^{(0)}_{\mathrm{j}} + 769\mathcal{U}^{(1)}_{\mathrm{j}} - 227\mathcal{U}^{(0)}_{\mathrm{j+1}} + 161\mathcal{U}^{(1)}_{\mathrm{j+1}}}{4} \\
\mathcal{U}^{(1)}_{\mathrm{j-2}} &= -90\mathsf{U}_{\mathsf{BC}} + 45\mathcal{U}^{(0)}_{\mathrm{j}} - 147\mathcal{U}^{(1)}_{\mathrm{j}} + 45\mathcal{U}^{(0)}_{\mathrm{j+1}} - 32\mathcal{U}^{(1)}_{\mathrm{j+1}}
\end{aligned} \tag{3.78}$$

The order of spatial discretization with this BC is $O\left(\Delta x^4\right)$.

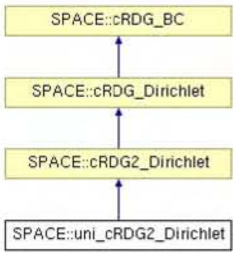

Fig. 3.21 : Hierarchical tree of the Class `uni_cRDG2_Dirichlet`.

cRDG$_2$. Forcing the in-cell recovered solution for the first ghost cell $_{(\mathrm{j-1})}$ and the first real cell $_{(\mathrm{j})}$ to satisfy

$$\begin{gathered}
\mathcal{U}^{\mathcal{R}}_{\mathrm{j}}\left(x_{\mathrm{j-\frac{1}{2}}}\right) = \mathsf{U}_{\mathsf{BC}} \\
\left(\mathcal{U}^{\mathcal{R}}_{\mathrm{j}}\left(x_{\mathrm{j-\frac{1}{2}}}\right)\right)^{(\mathrm{vii})} = \left(\mathcal{U}^{\mathcal{R}}_{\mathrm{j}}\left(x_{\mathrm{j-\frac{1}{2}}}\right)\right)^{(\mathrm{viii})} = \left(\mathcal{U}^{\mathcal{R}}_{\mathrm{j-1}}\left(x_{\mathrm{j-\frac{1}{2}}}\right)\right)^{(\mathrm{vi})} = \\
= \left(\mathcal{U}^{\mathcal{R}}_{\mathrm{j-1}}\left(x_{\mathrm{j-\frac{1}{2}}}\right)\right)^{(\mathrm{vii})} = \left(\mathcal{U}^{\mathcal{R}}_{\mathrm{j-1}}\left(x_{\mathrm{j-\frac{1}{2}}}\right)\right)^{(\mathrm{viii})} = 0
\end{gathered} \tag{3.79}$$

results in:

$$\begin{aligned}
\mathcal{U}^{(0)}_{\mathrm{j-1}} &= \frac{\left(\begin{gathered} 18840\mathsf{U}_{\mathsf{BC}} - 31270\mathcal{U}^{(0)}_{\mathrm{j}} + 4185\mathcal{U}^{(1)}_{\mathrm{j}} - 34719\mathcal{U}^{(2)}_{\mathrm{j}} + 12590\mathcal{U}^{(0)}_{\mathrm{j+1}} - \\ - 10845\mathcal{U}^{(1)}_{\mathrm{j+1}} + 6369\mathcal{U}^{(2)}_{\mathrm{j+1}} \end{gathered}\right)}{160} \\
\mathcal{U}^{(1)}_{\mathrm{j-1}} &= \frac{\left(\begin{gathered} -6552\mathsf{U}_{\mathsf{BC}} + 11238\mathcal{U}^{(0)}_{\mathrm{j}} - 1193\mathcal{U}^{(1)}_{\mathrm{j}} + 12591\mathcal{U}^{(2)}_{\mathrm{j}} - 4686\mathcal{U}^{(0)}_{\mathrm{j+1}} + \\ + 4045\mathcal{U}^{(1)}_{\mathrm{j+1}} - 2385\mathcal{U}^{(2)}_{\mathrm{j+1}} \end{gathered}\right)}{32} \\
\mathcal{U}^{(2)}_{\mathrm{j-1}} &= \frac{\left(\begin{gathered} 3720\mathsf{U}_{\mathsf{BC}} - 6690\mathcal{U}^{(0)}_{\mathrm{j}} + 355\mathcal{U}^{(1)}_{\mathrm{j}} - 7589\mathcal{U}^{(2)}_{\mathrm{j}} + 2970\mathcal{U}^{(0)}_{\mathrm{j+1}} - \\ - 2575\mathcal{U}^{(1)}_{\mathrm{j+1}} + 1531\mathcal{U}^{(2)}_{\mathrm{j+1}} \end{gathered}\right)}{32}
\end{aligned} \tag{3.80}$$

$$
\begin{aligned}
\mathcal{U}_{j-2}^{(0)} &= \frac{\left(\begin{array}{c} 97620\mathsf{U}_{BC} - 180825\mathcal{U}_{j}^{(0)} + 3480\mathcal{U}_{j}^{(1)} - 204567\mathcal{U}_{j}^{(2)} + 83245\mathcal{U}_{j+1}^{(0)} - \\ - 72510\mathcal{U}_{j+1}^{(1)} + 43497\mathcal{U}_{j+1}^{(2)} \end{array}\right)}{40} \\
\mathcal{U}_{j-2}^{(1)} &= \frac{\left(\begin{array}{c} -216720\mathsf{U}_{BC} + 410940\mathcal{U}_{j}^{(0)} + 2205\mathcal{U}_{j}^{(1)} + 465723\mathcal{U}_{j}^{(2)} - 194220\mathcal{U}_{j+1}^{(0)} + \\ + 169595\mathcal{U}_{j+1}^{(1)} + 43497 - 102213\mathcal{U}_{j+1}^{(2)} \end{array}\right)}{80} \\
\mathcal{U}_{j-2}^{(2)} &= \frac{\left(\begin{array}{c} 6900\mathsf{U}_{BC} - 13425\mathcal{U}_{j}^{(0)} - 435\mathcal{U}_{j}^{(1)} - 15216\mathcal{U}_{j}^{(2)} + 6525\mathcal{U}_{j+1}^{(0)} - \\ - 5715\mathcal{U}_{j+1}^{(1)} + 3464\mathcal{U}_{j+1}^{(2)} \end{array}\right)}{8}
\end{aligned}
$$

The order of spatial discretization with this BC is $O\left(\Delta x^6\right)$.

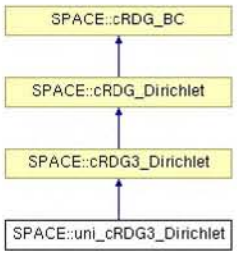

Fig. 3.22 : Hierarchical tree of the Class `uni_cRDG3_Dirichlet`.

cRDG$_3$. DoFs for ghost cells are computed forcing the in-cell recovered solution for the first ghost cell $_{(j-1)}$ and the first real cell $_{(j)}$ to satisfy

$$
\begin{aligned}
\mathcal{U}_{j}^{\mathcal{R}}\left(x_{j-\frac{1}{2}}\right) &= \mathsf{U}_{BC} \\
\left(\mathcal{U}_{j}^{\mathcal{R}}\left(x_{j-\frac{1}{2}}\right)\right)^{(xi)} = \left(\mathcal{U}_{j}^{\mathcal{R}}\left(x_{j-\frac{1}{2}}\right)\right)^{(x)} &= \left(\mathcal{U}_{j}^{\mathcal{R}}\left(x_{j-\frac{1}{2}}\right)\right)^{(ix)} = \left(\mathcal{U}_{j-1}^{\mathcal{R}}\left(x_{j-\frac{1}{2}}\right)\right)^{(xi)} = \\
= \left(\mathcal{U}_{j-1}^{\mathcal{R}}\left(x_{j-\frac{1}{2}}\right)\right)^{(x)} &= \left(\mathcal{U}_{j-1}^{\mathcal{R}}\left(x_{j-\frac{1}{2}}\right)\right)^{(ix)} = \left(\mathcal{U}_{j-1}^{\mathcal{R}}\left(x_{j-\frac{1}{2}}\right)\right)^{(viii)} = 0
\end{aligned} \tag{3.81}
$$

The order of spatial discretization with this BC is $O\left(\Delta x^8\right)$.

Neumann Boundary Conditions

Suppose we are given a gradient of a scalar $\mathcal{U}$ at the boundary, U'_{BC}. The ghost cells are populated a follows[14].

[14]For bravity, only left boundary and uniform mesh will be described. Extensions to the right boundary and irregular mesh are straightforward.

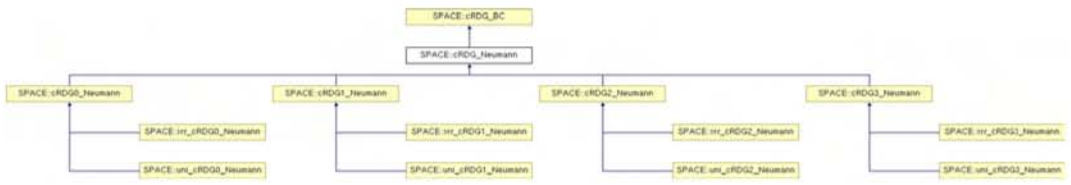

Fig. 3.23 : Hierarchical tree of the Class `cRDG_Neumann`.

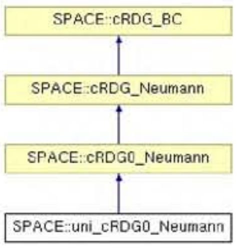

Fig. 3.24 : Hierarchical tree of the Class `uni_cRDG0_Neumann`.

cRDG$_0$. Forcing the in-cell recovered solution for the first real cell $_{(\mathrm{j})}$ to satisfy

$$\begin{aligned} \left(\mathcal{U}^{\mathcal{R}}_{\mathrm{j}}\left(x_{\mathrm{j}-\frac{1}{2}}\right)\right)' &= \mathsf{U}'_{\mathsf{BC}} \\ \left(\mathcal{U}^{\mathcal{R}}_{\mathrm{j}}\left(x_{\mathrm{j}-\frac{1}{2}}\right)\right)'' &= 0 \end{aligned} \tag{3.82}$$

results in:

$$\begin{aligned} \mathcal{U}^{(0)}_{\mathrm{j}-1} &= \mathcal{U}^{(0)}_{\mathrm{j}} - \Delta x_{\mathrm{j}} \mathsf{U}'_{\mathsf{BC}} \\ \mathcal{U}^{(0)}_{\mathrm{j}-2} &= \mathcal{U}^{(0)}_{\mathrm{j}} - 2\Delta x_{\mathrm{j}} \mathsf{U}'_{\mathsf{BC}} \end{aligned} \tag{3.83}$$

The order of spatial discretization with this BC is $O\left(\Delta x\right)$.

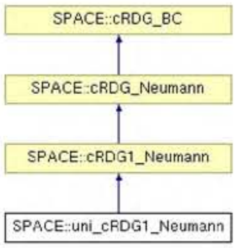

Fig. 3.25 : Hierarchical tree of the Class `uni_cRDG1_Neumann`.

cRDG$_1$. Forcing the in-cell recovered solution for the first ghost cell $_{(\mathrm{j}-1)}$ and the first real cell $_{(\mathrm{j})}$ to satisfy

$$\begin{array}{l} \left(\mathcal{U}^{\mathcal{R}}_{\mathrm{j}}\left(x_{\mathrm{j}-\frac{1}{2}}\right)\right)' = \left(\mathcal{U}^{\mathcal{R}}_{\mathrm{j}-1}\left(x_{\mathrm{j}-\frac{1}{2}}\right)\right)' = \mathsf{U}'_{\mathsf{BC}} \\ \left(\mathcal{U}^{\mathcal{R}}_{\mathrm{j}}\left(x_{\mathrm{j}-\frac{1}{2}}\right)\right)^{(\mathrm{v})} = \left(\mathcal{U}^{\mathcal{R}}_{\mathrm{j}-1}\left(x_{\mathrm{j}-\frac{1}{2}}\right)\right)^{(\mathrm{v})} = 0 \end{array} \tag{3.84}$$

results in

$$\begin{array}{rcl} \mathcal{U}^{(0)}_{\mathrm{j}-1} & = & \frac{-44\mathsf{U}'_{\mathsf{BC}}\Delta x + 99\mathcal{U}^{(0)}_{\mathrm{j}} + 135\mathcal{U}^{(1)}_{\mathrm{j}} - 75\mathcal{U}^{(0)}_{\mathrm{j}+1} + 55\mathcal{U}^{(1)}_{\mathrm{j}+1}}{24} \\ \mathcal{U}^{(1)}_{\mathrm{j}-1} & = & \frac{20\mathsf{U}'_{\mathsf{BC}}\Delta x - 45\mathcal{U}^{(0)}_{\mathrm{j}} - 89\mathcal{U}^{(1)}_{\mathrm{j}} + 45\mathcal{U}^{(0)}_{\mathrm{j}+1} - 33\mathcal{U}^{(1)}_{\mathrm{j}+1}}{8} \\ \mathcal{U}^{(0)}_{\mathrm{j}-2} & = & -50\mathsf{U}'_{\mathsf{BC}}\Delta x - 118\mathcal{U}^{(0)}_{\mathrm{j}} - 72\mathcal{U}^{(1)}_{\mathrm{j}} + 119\mathcal{U}^{(0)}_{\mathrm{j}+1} - 68\mathcal{U}^{(1)}_{\mathrm{j}+1} \\ \mathcal{U}^{(1)}_{\mathrm{j}-2} & = & 78\mathsf{U}'_{\mathsf{BC}}\Delta x + 183\mathcal{U}^{(0)}_{\mathrm{j}} + 107\mathcal{U}^{(1)}_{\mathrm{j}} - 183\mathcal{U}^{(0)}_{\mathrm{j}+1} + 104\mathcal{U}^{(1)}_{\mathrm{j}+1} \end{array} \tag{3.85}$$

The order of spatial discretization with this BC is $O\left(\Delta x^5\right)$.

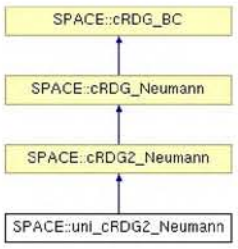

Fig. 3.26 : Hierarchical tree of the Class `uni_cRDG2_Neumann`.

cRDG$_2$. DoFs for ghost cells are computed forcing the in-cell recovered solution for the first ghost cell $_{(\mathrm{j}-1)}$ and the first real cell $_{(\mathrm{j})}$ to satisfy

$$\begin{array}{l} \left(\mathcal{U}^{\mathcal{R}}_{\mathrm{j}}\left(x_{\mathrm{j}-\frac{1}{2}}\right)\right)' = \left(\mathcal{U}^{\mathcal{R}}_{\mathrm{j}-1}\left(x_{\mathrm{j}-\frac{1}{2}}\right)\right)' = \mathsf{U}'_{\mathsf{BC}} \\ \\ \left(\mathcal{U}^{\mathcal{R}}_{\mathrm{j}}\left(x_{\mathrm{j}-\frac{1}{2}}\right)\right)^{(\mathrm{viii})} = \left(\mathcal{U}^{\mathcal{R}}_{\mathrm{j}}\left(x_{\mathrm{j}-\frac{1}{2}}\right)\right)^{(\mathrm{vii})} = \\ = \left(\mathcal{U}^{\mathcal{R}}_{\mathrm{j}-1}\left(x_{\mathrm{j}-\frac{1}{2}}\right)\right)^{(\mathrm{viii})} = \left(\mathcal{U}^{\mathcal{R}}_{\mathrm{j}-1}\left(x_{\mathrm{j}-\frac{1}{2}}\right)\right)^{(\mathrm{vi})} = 0 \end{array} \tag{3.86}$$

The order of spatial discretization with this BC is $O\left(\Delta x^6\right)$.

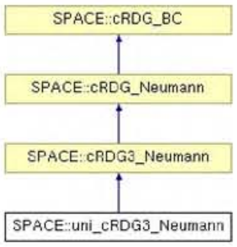

Fig. 3.27 : Hierarchical tree of the Class `uni_cRDG3_Neumann`.

cRDG$_3$. DoFs for ghost cells are computed forcing the in-cell recovered solution for the first ghost cell $_{(j-1)}$ and the first real cell $_{(j)}$ to satisfy

$$\left(\mathcal{U}^{\mathcal{R}}_{j}\left(x_{j-\frac{1}{2}}\right)\right)' = \left(\mathcal{U}^{\mathcal{R}}_{j-1}\left(x_{j-\frac{1}{2}}\right)\right)' = \mathsf{U}'_{\mathsf{BC}}$$

$$\begin{aligned}\left(\mathcal{U}^{\mathcal{R}}_{j}\left(x_{j-\frac{1}{2}}\right)\right)^{(\mathrm{xi})} = \left(\mathcal{U}^{\mathcal{R}}_{j}\left(x_{j-\frac{1}{2}}\right)\right)^{(\mathrm{x})} = \left(\mathcal{U}^{\mathcal{R}}_{j}\left(x_{j-\frac{1}{2}}\right)\right)^{(\mathrm{ix})} = \\ = \left(\mathcal{U}^{\mathcal{R}}_{j-1}\left(x_{j-\frac{1}{2}}\right)\right)^{(\mathrm{xi})} = \left(\mathcal{U}^{\mathcal{R}}_{j-1}\left(x_{j-\frac{1}{2}}\right)\right)^{(\mathrm{x})} = \left(\mathcal{U}^{\mathcal{R}}_{j-1}\left(x_{j-\frac{1}{2}}\right)\right)^{(\mathrm{viii})} = 0\end{aligned} \tag{3.87}$$

The order of spatial discretization with this BC is $O\left(\Delta x^{8}\right)$.

Characteristic Boundary Conditions

Characteristic boundary conditions are specified analyzing eigensystem of the hyperbolic part of eq.(3.1). The number of BCs is defined by the number of incoming waves. Special care must be exercised to not specify conflicting BCs.

Input options

The name of the input file for space discretization of a component derived from the class `DG_1D_CV` (Figure 3.5) is specified in the component's input file, under

```
[GeneralOptions]
  space_discr_file = discretization
[]
```

. File "`discretization.inp`" should be located in the "`INPUT/`" folder. The following input options are currently available:

-
```
[General]
  space_discr = ...
[]
```
, space discretization scheme:

(0) Finite Volume, class `DG0_1D`.

(1) Linear DG, class `DG1_1D`.

(2) Quadratic DG, class `DG2_1D`.

(3) Cubic DG, class `DG3_1D`.

-
```
[General]
  recovery_InCell = true/false
[]
```
: apply/don't in-cell recovery (cRDG).

-
```
[General]
  Limiting = ...
[]
```
, options for limiters.

  In the case of $DG_{n>0}$:

  (0) Unlimited.

  ($\neq 0$) Krivodonova' limiting (class `Krivodonova`).

  In the case of finite-volume (DG_0):

  (0) Unlimited.

  (1) $WENO_3$ (Classes `uni_PPM_WENO` or `irr_PPM_WENO`).

  (2) Van Albada 1 (Classes `uni_PPM` or `irr_PPM` plus `vanAlbada1`).

  (3) Van Albada 2 (Classes `uni_PPM` or `irr_PPM` plus `vanAlbada2`).

  (4) MinMod (Classes `uni_PPM` or `irr_PPM` plus `MinMod`).

  (5) Van Leer 2 (Classes `uni_PPM` or `irr_PPM` plus `vanLeer2`).

  (6) Superbee (Classes `uni_PPM` or `irr_PPM` plus `Superbee`).

-
```
[General]
  num_of_QP = ...
[]
```
: this is the number of quadrature points that code will use, if it is greater than the minimum (default) number of Gaussian quadrature points for in-cell recovery (see the Note at the end of Section 3.3.1).

-
```
[General]
  flux_scheme = ...
[]
```
, flux scheme used for hyperbolic operator. The following options are available:

  (0) Central (class `Central`).

(1) Local-Lax-Friedrichs (class `LLF`).

(2) AUSM$^+$-up (class `AUSM`).

(3) Godunov (class `Godunov`).

(4) LLF in characteristic space (class `cLLF`).

(5) Roe-Fix (class `RF`).

- Parameter α in LLF and cLLF schemes is defined as

```
[LLF]
  alpha = ...
[]
```
.

- Parameters of the AUSM$^+$-up scheme [Lio06] are defined as

```
[AUSM]
  Split_M_number = ...
  Split_P_funct = ...
  Interf_C_sound = ...
  low_MACH_AUSM = ...
  Mach2_inf_AUSM = ...
[]
```

DG for Primitive (Non-Conservative) Variables

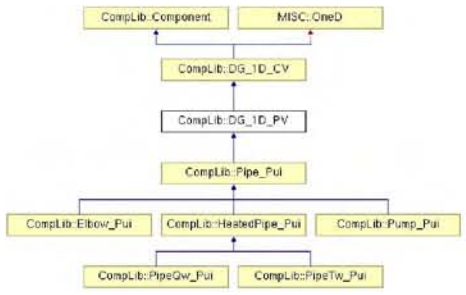

Fig. 3.28 : Hierarchical tree of the Class `DG_1D_PV`.

In many practical applications, the use of conservative variables is not the best choice for linear algebra. For example, in stiff fluids low-Mach-number flows, partial derivatives of pressure relative to density and temperature are very large, $\left.\frac{\partial P}{\partial \rho}\right|_{T_0}$ and $\left.\frac{\partial P}{\partial T}\right|_{\rho_0} \gg 1$. In these flows, density is nearly constant (incompressible limit), while pressure and internal energy/temperature may vary significantly. Thus, from the linearization around a state (P_0, ρ_0, T_0)

$$P \approx P_0 + (\rho - \rho_0) \left.\frac{\partial P}{\partial \rho}\right|_{T_0} + (T - T_0) \left.\frac{\partial P}{\partial T}\right|_{\rho_0} \tag{3.88}$$

one can see that small error in density may cause significant errors in pressure. Much better choice would be to use the set of non-conservative variables, for example, $\begin{bmatrix} P \\ u \\ i \end{bmatrix}$.

Modification of the described in Sections 3.3.1-3.3.1 DG schemes is rather straightforward. We solve for $(p+1)$ degrees of freedom (DoFs) of primitive variables in each cell, $\vec{\mathcal{V}}_{\text{cell}}^{(n=0,\dots,p)}$, representing a solution inside cell $\mathcal{I}_{\text{cell}}$ as:

$$\vec{\mathcal{V}}_{\mathcal{I}_{\text{cell}}}(x) = \sum_{n=0}^{p} \vec{\mathcal{V}}_{\text{cell}}^{(n)} \, \mathfrak{L}_{(n)}\left(\xi_{\text{cell}}\right) \tag{3.89}$$

In implicit schemes, we drive residuals for mass, momentum and energy conservation of type eq.(3.25) to zero. In this formulation, source terms defined by eq.(3.27) are computed using primitive-variable- in-cell and inter-cell recovered solutions. Time derivatives require $\vec{\mathcal{U}}_{\mathrm{j}}^{[n+1]} - \vec{\mathcal{U}}_{\mathrm{j}}^{[n]}$. The DoFs for conservative vectors are computed using high-order mapping $\vec{\mathcal{V}}_{\mathrm{j}}^{[\times]} \to \vec{\mathcal{U}}_{\mathrm{j}}^{[\times]}$:

$$\vec{\mathcal{U}}_{\mathrm{j}}^{(n),R,[\times]} = \frac{1+2n}{\Delta x_{\mathrm{j}}} \int\limits_{\mathcal{I}_{\mathrm{j}}} \mathfrak{L}_{(n)}\left(\xi_{\mathrm{j}}\right) \vec{\mathcal{U}}\left(\vec{\mathcal{V}}_{\mathcal{I}_{\mathrm{cell}}}^{[\times]}(x)\right) dx \tag{3.90}$$

where we used *Gauss-Chebyshev* integration [PTVF99].

In terms of JFNK, the conservative residuals are used to advance solution for primitive variables as

$$\begin{aligned} \vec{\mathcal{X}}^{(a)} &= \vec{\mathcal{X}}^{(a-1)} - (\mathbb{J}\mathbb{A})^{-1}\, r\vec{e}s^{(a-1)} \\ a &= 0, 1, \ldots \end{aligned} \tag{3.91}$$

where $\mathbb{A}$ is defined are the Jacobian of transformation $\frac{\partial \vec{\mathcal{U}}}{\partial \vec{\mathcal{V}}}$, which with a proper choice of primitive variables ought to generate a better-conditioned JFNK's Jacobian matrix $\mathbb{J}\mathbb{A}$.

Importantly, since we compute residuals using conservative formulation, the only possible conservation errors might sneak in only due to discretization/integration errors in eq.(3.90).

3.3.2 Two-Dimensional Structured Mesh

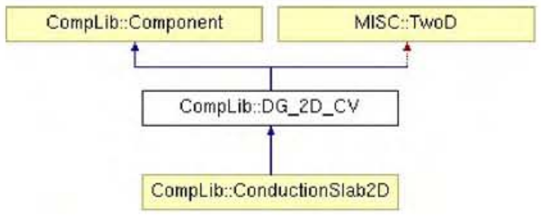

Fig. 3.29 : Hierarchical tree of the Class DG_2D_CV.

In this section we describe 2D structured-mesh discontinuous Galerkin Legendre-polynomial based spatial discretization, as implemented in components derived from DG_2D_CV, Figure 3.29.

Extension of the cRDG to 2D structured grid is straightforward. This was demonstrated for linear DG in [NPM09], where we developed the *in-cell* and *inter-cell* recovery for equations of type eq.(3.1), demonstrating their applications for solving Navier-Stokes equations and coupled non-linear neutron diffusion and heat conduction problem on 2D regular structured grids. The key aspect in the developing in-cell recovery was to choose a proper set of weak statements of type eq.(3.31), in order to be able to construct additional degrees of freedom. Specifically, we dropped some 5^{th}- and 6^{th}-order cross-derivatives, and chose only two weak statements for von Neumann (face) neighbors, and one weak statement ("cell-average") for vertex neighbors. The resulting cRDG_1 was formally 4^{th}-order accurate (6^{th}-order in normal derivatives). More details can be found in [NPM09].

3.3.3 Two-Dimensional Unstructured Mesh

Extension of the cRDG to unstructured grid is more demanding. It is much more practical to apply ***reconstruction***, rather than *recovery*. In [LLNM09], Luo et al. developed the *"In-Cell Reconstructed DG"* for compressible Euler equations. A quadratic polynomial solution is reconstructed based on underlying linear DG solution using only von Neumann (face) neighbors and a least-squares method. The method was shown to be as accurate as quadratic DG, but computationally more efficient. This is because the vector of unknowns for the $\text{cRDG}_1^{(3)}$ is two times smaller, and the numbers of quadrature points for domain and face integrations were also less than those required for the $\text{DG}_2^{(3)}$. The $\text{cRDG}_1^{(3)}$ was demonstrated to perform excellently on arbitrary grids (including hybrid triangle/quads, and highly-stretched Navier-Stokes meshes), using examples of low and intermediate Mach number flows around cylinder, NACA0012 airfoil, and three-element airfoil (for details, see [LLNM09]).

3.4 Jacobian-Free Newton Krylov (JFNK) Method

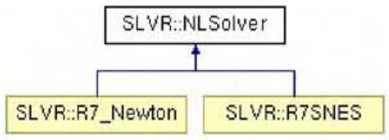

Fig. 3.30 : Hierarchical tree of the Class `NLSolver`.

EACH stage of IRK (Section 3.2.2) requires solution of the non-linear system in the form

$$\vec{res}\left(\vec{\mathcal{X}}\right) = 0 \tag{3.92}$$

where

$$\vec{\mathcal{X}} = \left(\vec{\mathcal{U}}_{1}^{(k=0,\dots,p)^{\mathsf{T}}}, \vec{\mathcal{U}}_{2}^{(k=0,\dots,p)^{\mathsf{T}}}, \dots, \vec{\mathcal{U}}_{\mathrm{N_{cells}}}^{(k=0,\dots,p)^{\mathsf{T}}}\right)^{\mathsf{T}} \tag{3.93}$$

is a solution vector which includes all $(p+1)$ degrees of freedom for all variables in all N_{cells} computational cells. The residual vector $\vec{res}$ for each DoF at the cell $_{(\mathrm{c})}$ takes the form:

$$res_{\mathcal{U}_{\mathrm{c}}^{(k)}} = \mathcal{U}_{\mathrm{c}}^{(k)[rk]} - \mathcal{U}_{\mathrm{c}}^{(k)[n]} - \Delta t \sum_{r=1}^{rk} \mathsf{a}_{rk,r} \mathcal{S}_{\mathrm{c}}^{(k)}\left(\vec{\mathcal{X}}^{[r]}\right) \tag{3.94}$$

In the following sections, we will describe nonlinear algorithm used to solve the system eq.(3.92). Input parameters controlling this nonlinear solve are specified in file "`INPUT/JFNK.inp`".

3.4.1 Newton's method

We solve Eq.(3.94) with Newton's method, iteratively, as a sequence of linear problems defined by

$$\mathbb{J}^{a}\delta\vec{\mathcal{X}}^{a} = -\vec{res}\left(\vec{\mathcal{X}}^{a}\right) \tag{3.95}$$

The matrix $\mathbb{J}^{a}$ is the Jacobian of the a^{th} Newton's iteration and $\delta\vec{\mathcal{X}}^{a}$ is the update vector. Each $_{(c)^{\text{th}}}$ element of the Jacobian matrix is a partial derivative of the i^{th}

equation with respect to the j^{th} variable:

$$\mathbb{J}_{i,j} \equiv \frac{\partial res_i}{\partial \mathcal{X}_j} \tag{3.96}$$

The linear system Eq.(3.95) is solved for $\delta\vec{\mathcal{X}}^a$, and the new Newton's iteration value for $\vec{\mathcal{X}}$ is then computed as

$$\vec{\mathcal{X}}^{a+1} = \vec{\mathcal{X}}^a + \delta\vec{\mathcal{X}}^a \tag{3.97}$$

Newton's iterations on $\vec{\mathcal{X}}$ are continued until the convergence criterion

$$\left|\left|r\vec{e}s\left(\vec{\mathcal{X}}^a\right)\right|\right|_2 < \text{tol}_{\text{N}} \left|\left|r\vec{e}s\left(\vec{\mathcal{X}}^0\right)\right|\right|_2 \tag{3.98}$$

is satisfied. The nonlinear tolerance is $\text{tol}_{\text{N}} = 10^{-8}$.

The algorithm described above is the simplest nonlinear solver (`NonLinear_Slvr`=2). There are several other Newton-like algorithms available [BGMS97, BBG+01, BBE+04], including:

- *Line Search with cubic backtracking* [DS83] (`NonLinear_Slvr`=0, default);
- *Line Search with quadratic backtracking* [DS83] (`NonLinear_Slvr`=1) and
- *Trust Region Method* [MSGH84] (`NonLinear_Slvr`=4).

3.4.2 Krylov subspace iterations (GMRES)

The linear solver used in our code is the Arnoldi-based *Generalized Minimal RESidual method (GMRES)* [SS86]. It belongs to the general class of Krylov subspace iteration methods. These projection (Galerkin) or generalized projection generalized projection (Petrov-Galerkin) methods [Saa03] are suitable for solving non-symmetric linear systems of the form Eq.(3.95), using the Krylov subspace, $\mathbb{K}_j$,

$$\mathbb{K}_j = \text{span}\left(\vec{r}_0, \mathbb{J}\vec{r}_0, \mathbb{J}^2\vec{r}_0, ..., \mathbb{J}^{j-1}\vec{r}_0\right) \tag{3.99}$$

where $\vec{r}_0 = \mathbb{J}^a \delta\vec{\mathcal{X}}_0^a + r\vec{e}s\left(\vec{\mathcal{X}}^a\right)$. In GMRES, the Arnoldi basis vectors form a trial subspace out of which the m^{th}-iteration solution is constructed:

$$\delta\vec{\mathcal{X}}_m^a = \delta\vec{\mathcal{X}}_0^a + \mathfrak{k}_0\vec{r}_0 + \mathfrak{k}_1\mathbb{J}\vec{r}_0 + \mathfrak{k}_2\mathbb{J}^2\vec{r}_0 + \ldots + \mathfrak{k}_m\mathbb{J}^m\vec{r}_0 \tag{3.100}$$

where $(\mathfrak{k}_0, \mathfrak{k}_1, \ldots, \mathfrak{k}_m)$ are "coordinates" of the m^{th} trial solution in the Krylov subspace. As one can see, only matrix-vector products are required to create new trial vectors. The iterations are terminated based on a by-product (free) estimate of the residual that does not require explicit construction of intermediate residual vectors. This is a major advantage of GMRES over other Krylov methods. GMRES has a residual minimization property in the Euclidean norm. The major drawback of GMRES is that it requires the storage of all previous Arnoldi/(Krylov) basis vectors. This problem can be alleviated with efficient preconditioning (see Chapter 3.5).

3.4.3 Jacobian-free implementation

Since GMRES does not require individual elements of the Jacobian matrix $\mathbb{J}$, it never needs to be constructed. Instead only matrix-vector multiplications $\mathbb{J}\vec{\kappa}$ are needed, where $\vec{\kappa} \in (\vec{r}_0, \mathbb{J}\vec{r}_0, \mathbb{J}^2\vec{r}_0, \ldots)$ are Krylov vectors. Thus, *Jacobian-free implementations* are possible. The action of the Jacobian matrix can be approximated by Fréchet derivatives

$$\mathbb{J}\vec{\kappa} \approx \frac{r\vec{e}s\left(\vec{\mathcal{X}} + \varepsilon\vec{\kappa}\right) - r\vec{e}s\left(\vec{\mathcal{X}}\right)}{\varepsilon} \tag{3.101}$$

There are two approaches for choosing ε, controlled by `NLSvr_mf_type` parameter.

Brown & Saad. The first (default, `NLSvr_mf_type`=0) approach is taken from [BS90]:

$$\varepsilon = \begin{cases} \epsilon_{\text{rel}} \frac{\vec{\mathcal{X}}^{\mathsf{T}}\vec{\kappa}}{\|\vec{\kappa}\|_2^2} & \text{if } \left|\vec{\mathcal{X}}'\vec{\kappa}\right| > \mathcal{X}_{\min}\|\vec{\kappa}\|_1 \\ \epsilon_{\text{rel}}\mathcal{X}_{\min}\,\text{Sign}\left(\vec{\mathcal{X}}^{\mathsf{T}}\vec{\kappa}\right)\frac{\|\vec{\kappa}\|_1}{\|\vec{\kappa}\|_2^2} & \text{otherwise} \end{cases} \tag{3.102}$$

There are two control parameters: ϵ_{rel} (`NLSvr_Erel`=10^{-8} on default) and $\mathcal{X}_{\min}$ (`NLSvr_Umin`=10^{-6} on default).

Pernice & Walker. The second approach is taken from [PW98]:

$$\varepsilon = \frac{\epsilon_{\text{rel}}\sqrt{1+\left|\left|\vec{\mathcal{X}}\right|\right|}}{||\vec{\kappa}||} \tag{3.103}$$

The only control parameter is ϵ_{rel} (`NLSvr_Erel`=10^{-8} on default). Note that for the entire linear iterative process $\vec{\mathcal{X}}$ does not change. Therefore, $\sqrt{1+\left|\left|\vec{\mathcal{X}}\right|\right|}$ need be computed only once.

With the Jacobian-free formulation, the work associated with forming the Jacobian matrix and its storage can be eliminated, which is a significant saving of both CPU time and storage, provided that the number of Krylov vectors is kept small (see Chapter 3.5). Moreover, in many non-linear applications, the Jacobian matrix is not available due to size and complexity.

3.4.4 Inexact Newton

One important modification to Newton's method employed here is called an *inexact Newton's method* [KK04]. The term "inexact" refers to the accuracy of the iterative linear solver. The basic idea is that the linear system must be solved to a tight tolerance only when the added accuracy matters – i.e., when it affects the convergence of the Newton's iterations. This is accomplished by making the convergence of the linear residual proportional to the non-linear residual:

$$\left|\left|\mathbb{J}^{a}\delta\vec{\mathcal{X}}^{a}_{m} + r\vec{e}s\left(\vec{\mathcal{X}}^{a}\right)\right|\right| \leq \nu_{a}\left|\left|r\vec{e}s\left(\vec{\mathcal{X}}^{a}\right)\right|\right| \tag{3.104}$$

By default, ν_{a} is a constant. Alternatively, one can invoke the algorithm by Eisenstat and Walker [EW96] (by setting parameter `NLSvr_EW`=true), which computes ν_{a} at each step of the nonlinear solver.

3.5 Preconditioning

BECAUSE GMRES (Section 3.4.2) stores all of the previous Krylov vectors, it is necessary to keep the number of iterations relatively small, to prevent the storage and CPU time from becoming prohibitive. This is accomplished by preconditioning the linear system. *Preconditioning is a transformation of the original linear system into one with the same solution, but is easier to solve with an iterative solver* [Saa03]. We are using the right-preconditioned form of the linear system,

$$\underbrace{\mathbb{J}^{a}\mathbb{P}^{-1}}_{\hat{\mathbb{J}}}\underbrace{\mathbb{P}\delta\vec{\mathcal{X}}^{a}}_{\delta\vec{\mathcal{Y}}^{a}} = -r\vec{e}s\left(\vec{\mathcal{X}}^{a}\right) \tag{3.105}$$

where $\mathbb{P}^{-1}$ approximates $\mathbb{J}^{-1}$. The right-preconditioned version of Eq.(3.101) is

$$\mathbb{J}\mathbb{P}^{-1}\vec{\kappa} \approx \frac{r\vec{e}s\left(\vec{\mathcal{X}}+\varepsilon\mathbb{P}^{-1}\vec{\kappa}\right)-r\vec{e}s\left(\vec{\mathcal{X}}\right)}{\varepsilon} \tag{3.106}$$

This operation is applied once per GMRES iteration, in two steps:

```
 I. Preconditioning:  approximately solve $\delta\vec{\mathcal{Y}}^{a} = \mathbb{P}^{-1}\vec{\kappa}$
II. Compute matrix-free product:  $\mathbb{J}\delta\mathcal{Y}^{a} \approx \frac{r\vec{e}s\left(\vec{\mathcal{X}}^{a}+\varepsilon\delta\vec{\mathcal{Y}}^{a}\right)-r\vec{e}s\left(\vec{\mathcal{X}}^{a}\right)}{\varepsilon}$
```

Finding a good preconditioner is often a combination of art, science, and intuition. A mathematically good preconditioner should efficiently cluster the eigenvalues of the iteration matrix [Saa03, KK04]. A preconditioner can also be defined as any subsidiary *approximate solver* that is combined with an outer iteration technique (e.g., multigrid, or one of the Krylov iteration solvers). One of the simplest and most popular ways of defining a preconditioner is to perform an incomplete lower-upper (ILU) factorization of the original matrix $\mathbb{J}$. A number of variations – ILU(k), ILUT, ILUS, ILUC, etc. – are discussed in [Saa03].

An important class of preconditioners for the JFNK method is referred to as *Physics-Based-Preconditioning (PBP)* or PDE-based [KK04]. The motivation behind this approach is that there exist numerous legacy operator-split algorithms to solve nonlinear systems. These algorithms were developed with insight into physical time scales of the problem. A direct benefit of this insight – a reduced implicit system, or a sequence of segregated semi-implicit solvers can be applied, instead of attempting to solve the fully-coupled system. Relevant fluid dynamics examples include the semi-implicit all-speed-flow *Implicit Continuous-fluid Eulerian (ICE)* algorithm [HA71], the semi-implicit incompressible-flow *SIMPLE* [Pat80] and the *Projection* algorithms [Cho67].

3.6 Equation of State and Thermophysical Properties

3.6.1 Simple Substances

THE simplest descriptions of matter are for substances idealized as having only one relevant reversible work mode. These substances are called *simple substances*. The state postulate defines that the number of independently variable intensive thermodynamic properties of a simple substance is *two*.

More specifically, we are concerned with substances for which the only important reversible work mode is volume change ($P\ dv$ work). The theory of simple compressible substances is quite well developed [RP77, Atk94], and considerable data have been accumulated relating the thermodynamic properties of many such substances. While realizing that no substance is truly simple, it is customary in engineering analyses to satisfactory treat materials involved as simple compressible substances.

The hierarchical tree of the Class `Simple_Substance` is shown in Figure 3.32.

From the state postulate we know that the temperature (T) and pressure (P) of a simple compressible substance can be expressed functionally as

$$\begin{aligned} T &= T\,(i, v) \\ P &= P\,(i, v) \end{aligned}$$

where i and $v = \frac{1}{\rho}$ are the specific internal energy and specific volume, respectively. These relations imply that we could completely fix the intensive thermodynamic state of a simple compressible substance by specifying any two independently variable intensive thermodynamic properties. It is well known [RP77] that *temperature* and *specific volume* are always independent properties for a simple compressible substance, and we can think of the pressure and specific internal energy as being functions of these properties,

$$\begin{aligned} P &= P\,(T, v) \\ i &= i\,(T, v) \end{aligned}$$

In certain special cases these equations can be expressed in explicit algebraic form, but in general it is easier to represent them in tables. The equations, in algebraic,

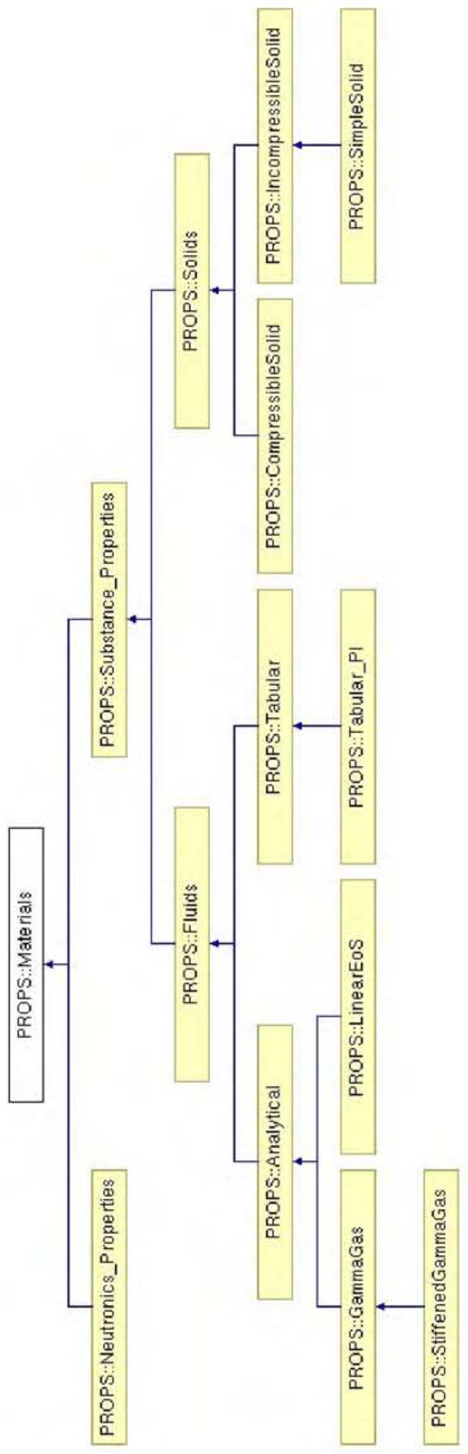

Fig. 3.31 : Inheritance tree for Class `Materials`.

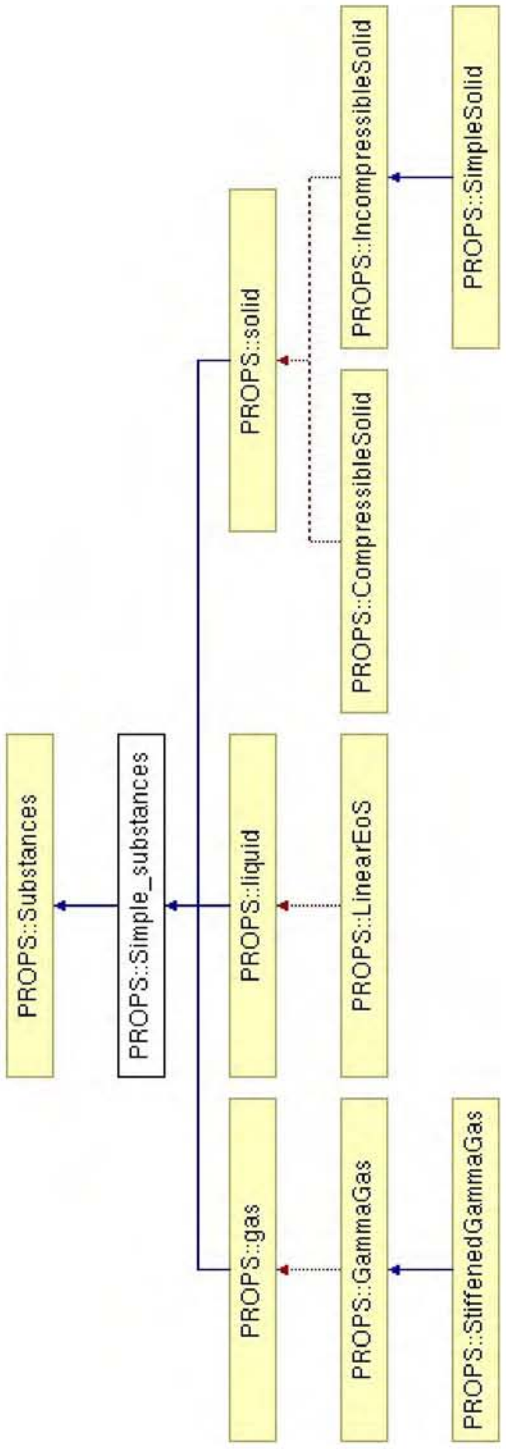

Fig. 3.32 : Inheritance tree for Class `Simple_Substance`.

graphical, or tabular form, which relate the intensive thermodynamic properties of any substance are termed the *equations of state* of the substance.

The class of `Simple_Substance` materials incorporates those which do not allow phase transition (melting, freezing, evaporation, condensation). Thus, pressure P and temperature T are always independent variables, which allows for some simplifications. In particular, using the calculus of functions of two variables, for any three functions x, y and z, any two of which may be selected as the independent pair,

$$\left(\frac{\partial x}{\partial y}\right)_z \left(\frac{\partial y}{\partial z}\right)_x \left(\frac{\partial z}{\partial x}\right)_y = -1 \tag{3.107}$$

This allows to compute important partials of independent variables. For example,

- $\left(\frac{\partial P}{\partial T}\right)_v$ from $v = v(P,T)$, as

$$\left(\frac{\partial P}{\partial T}\right)_v \left(\frac{\partial T}{\partial v}\right)_P \left(\frac{\partial v}{\partial P}\right)_T = -1 \implies \left(\frac{\partial P}{\partial T}\right)_v = -\frac{\left(\frac{\partial v}{\partial T}\right)_P}{\left(\frac{\partial v}{\partial P}\right)_T} \tag{3.108}$$

- $\left(\frac{\partial P}{\partial i}\right)_\rho$ from $\rho = \rho(P,i)$:

$$\left(\frac{\partial P}{\partial i}\right)_\rho = -\frac{\left(\frac{\partial \rho}{\partial i}\right)_P}{\left(\frac{\partial \rho}{\partial P}\right)_i} \tag{3.109}$$

To simplify farther, we separately treat *Fluids* (liquids and gases), 3.6.2, and the *Solids*.

Enthalpy. A thermodynamic property which is of particular importance is the *enthalpy* h, defined by

$$h \equiv i + Pv \tag{3.110}$$

Specific heats. Consider the specific internal energy as a function of T and v, $i = i(T,v)$. The difference in energy between any two infinitesimally close states is then

$$di = \left(\frac{\partial i}{\partial T}\right)_v dT + \left(\frac{\partial i}{\partial v}\right)_T dv \tag{3.111}$$

The slope of a line of constant v on a $(i - T)$ thermodynamic plane is a function of state, and called *specific heat at constant volume*[15]:

$$C_v \equiv \left(\frac{\partial i}{\partial T}\right)_v \tag{3.112}$$

Using the calculus of functions of two variables[16],

$$C_v = \frac{1}{\left(\frac{\partial T}{\partial i}\right)_P - \frac{\left(\frac{\partial T}{\partial P}\right)_i \left(\frac{\partial \rho}{\partial i}\right)_P}{\left(\frac{\partial \rho}{\partial P}\right)_i}} \tag{3.116}$$

which can be used for tabulated equations of state, Section 3.6.4.

Another specific heat is defined as

$$C_P \equiv \left(\frac{\partial h}{\partial T}\right)_P = \left(\frac{\partial i}{\partial T}\right)_P - \frac{P}{\rho^2}\left(\frac{\partial \rho}{\partial T}\right)_P \tag{3.117}$$

[15]Sometimes called *specific isochoric heat capacity*.

[16] In fact,

$$\begin{aligned} d\rho &= \left(\frac{\partial \rho}{\partial P}\right)_i dP + \left(\frac{\partial \rho}{\partial i}\right)_P di \\ dT &= \left(\frac{\partial T}{\partial P}\right)_i dP + \left(\frac{\partial T}{\partial i}\right)_P di \\ dT &= \left(\frac{\partial T}{\partial \rho}\right)_i d\rho + \left(\frac{\partial T}{\partial i}\right)_\rho di \end{aligned}$$

Eliminating dT in the last two equations, plugging $d\rho$ from the first one, and collecting the terms with dP on the left and with di on the right,

$$dP\left[\left(\frac{\partial T}{\partial P}\right)_i - \left(\frac{\partial T}{\partial \rho}\right)_i\left(\frac{\partial \rho}{\partial P}\right)_i\right] = di\left[\left(\frac{\partial T}{\partial i}\right)_\rho - \left(\frac{\partial T}{\partial i}\right)_P + \left(\frac{\partial T}{\partial \rho}\right)_i\left(\frac{\partial \rho}{\partial i}\right)_P\right] \tag{3.113}$$

Since the changes di and dP are independent, setting $di = 0$ leads to

$$\left(\frac{\partial T}{\partial \rho}\right)_i = \frac{\left(\frac{\partial T}{\partial P}\right)_i}{\left(\frac{\partial \rho}{\partial P}\right)_i} \tag{3.114}$$

Similarly, setting $dP = 0$ leads to

$$\left(\frac{\partial T}{\partial i}\right)_\rho = \left(\frac{\partial T}{\partial i}\right)_P - \left(\frac{\partial T}{\partial \rho}\right)_i\left(\frac{\partial \rho}{\partial i}\right)_P \tag{3.115}$$

which is inverse of equation (3.116).

and called the *specific heat at constant pressure*[17]. Using the calculus of functions of two variables[18],

$$C_{P} = \frac{1 - \frac{P}{\rho^2}\left(\frac{\partial\rho}{\partial i}\right)_P}{\left(\frac{\partial T}{\partial i}\right)_P} \tag{3.121}$$

Both C_P and C_v constitute two of the most important thermodynamic derivative functions, and their values have been experimentally determined as functions of the thermodynamic state for a tremendous number of simple compressible substances, [RP77, Atk94].

Compressibilities. Consider the specific volume as a function of T and P, $v = v(T, P)$. The difference in specific volume between any two infinitesimally close

[17] It is also sometimes called the *heat capacity at constant pressure*.

[18] In fact,

$$\begin{aligned} d\rho &= \left(\frac{\partial\rho}{\partial P}\right)_i dP &+& \left(\frac{\partial\rho}{\partial i}\right)_P di \\ dT &= \left(\frac{\partial T}{\partial P}\right)_i dP &+& \left(\frac{\partial T}{\partial i}\right)_P di \\ dT &= \left(\frac{\partial T}{\partial\rho}\right)_i d\rho &+& \left(\frac{\partial T}{\partial P}\right)_\rho dP \end{aligned}$$

Eliminating dT in the last two equations, plugging $d\rho$ from the first one, and collecting the terms with dP on the left and with di on the right,

$$dP\left[\left(\frac{\partial T}{\partial P}\right)_i - \left(\frac{\partial T}{\partial P}\right)_\rho - \left(\frac{\partial T}{\partial\rho}\right)_P\left(\frac{\partial\rho}{\partial P}\right)_i\right] = di\left[\left(\frac{\partial\rho}{\partial i}\right)_P\left(\frac{\partial T}{\partial\rho}\right)_P - \left(\frac{\partial T}{\partial i}\right)_P\right] \tag{3.118}$$

Since the changes di and dP are independent, setting $dP = 0$ leads to

$$\left(\frac{\partial T}{\partial\rho}\right)_P = \frac{\left(\frac{\partial T}{\partial i}\right)_P}{\left(\frac{\partial\rho}{\partial i}\right)_P} \tag{3.119}$$

Similarly, setting $di = 0$ leads to

$$\left(\frac{\partial T}{\partial P}\right)_\rho = \left(\frac{\partial T}{\partial P}\right)_i - \left(\frac{\partial T}{\partial i}\right)_P \frac{\left(\frac{\partial\rho}{\partial P}\right)_i}{\left(\frac{\partial\rho}{\partial i}\right)_P} \tag{3.120}$$

Combining eq.(3.119) and (3.117) leads to eq.(3.121).

states is then

$$dv = \left(\frac{\partial v}{\partial T}\right)_P dT + \left(\frac{\partial v}{\partial P}\right)_T dP$$

The slope $\left(\frac{\partial v}{\partial T}\right)_P$ represents the sensitivity of the specific volume to changes in temperature at constant pressure, leading to the definition of the *isobaric compressibility*:

$$\beta \equiv \frac{1}{v}\left(\frac{\partial v}{\partial T}\right)_P = -\frac{1}{\rho}\left(\frac{\partial \rho}{\partial T}\right)_P \tag{3.122}$$

Using eq.(3.127),

$$\beta = -\frac{1}{\rho}\frac{\left(\frac{\partial \rho}{\partial i}\right)_P}{\left(\frac{\partial T}{\partial i}\right)_P} \tag{3.123}$$

The *coefficient of linear expansion* used in elementary strength-of-materials textbooks is

$$\alpha = \frac{1}{3}\beta$$

The slope $\left(\frac{\partial v}{\partial P}\right)_T$ is a measure of the change in specific volume associated with a change in pressure at constant temperature, defining the *isothermal compressibility*:

$$\chi \equiv -\frac{1}{v}\left(\frac{\partial v}{\partial P}\right)_T = \frac{1}{\rho}\left(\frac{\partial \rho}{\partial P}\right)_T \tag{3.124}$$

Young's modulus of elasticity is proportional to χ. Using the calculus of functions of two variables[19],

$$\chi = \frac{1}{\rho}\left[\left(\frac{\partial \rho}{\partial P}\right)_i - \frac{\left(\frac{\partial \rho}{\partial i}\right)_P \left(\frac{\partial T}{\partial P}\right)_i}{\left(\frac{\partial T}{\partial i}\right)_P}\right] \tag{3.128}$$

[19] In fact,

$$\begin{aligned} d\rho &= \left(\frac{\partial \rho}{\partial P}\right)_T dP + \left(\frac{\partial \rho}{\partial T}\right)_P dT \\ d\rho &= \left(\frac{\partial \rho}{\partial P}\right)_i dP + \left(\frac{\partial \rho}{\partial i}\right)_P di \\ dT &= \left(\frac{\partial T}{\partial P}\right)_i dP + \left(\frac{\partial T}{\partial i}\right)_P di \end{aligned}$$

Eliminating $d\rho$ in the first two equations, plugging into that the last equation for dT, and collecting terms for dP on the left and for di on the right,

$$dP\left[\left(\frac{\partial \rho}{\partial P}\right)_T - \left(\frac{\partial \rho}{\partial P}\right)_i + \left(\frac{\partial \rho}{\partial T}\right)_P \left(\frac{\partial T}{\partial P}\right)_i\right] = di\left[\left(\frac{\partial \rho}{\partial i}\right)_P - \left(\frac{\partial \rho}{\partial T}\right)_P \left(\frac{\partial T}{\partial i}\right)_P\right] \tag{3.125}$$

It is also possible to demonstrate that

$$\chi = \frac{C_{P}}{\rho C_{v} c^2} \tag{3.129}$$

The sound speeds of materials are related to the partial derivatives of the pressure [Atk94], $\left(\frac{\partial P}{\partial \rho}\right)_i$ and $\left(\frac{\partial P}{\partial i}\right)_\rho$:

$$c = \sqrt{\left(\frac{\partial P}{\partial \rho}\right)_i + \frac{P\left(\frac{\partial P}{\partial i}\right)_\rho}{\rho^2}} \tag{3.130}$$

Using eq.(3.109),

$$c = \sqrt{\frac{1}{\left(\frac{\partial \rho}{\partial P}\right)_i}\left(1 - \frac{P\left(\frac{\partial \rho}{\partial i}\right)_P}{\rho^2}\right)} \tag{3.131}$$

3.6.2 Fluids

The hierarchical tree of the Class `Fluids` is shown in Figure 3.33. We split fluid materials into two groups: those which have *analytical* form of equations of state (Sections 3.6.3-3.6.4), and those which are specified in *tabular/bi-cubic interpolation* form.

3.6.3 Gamma Law Gas

Class `GammaGas` is a subset of general fluids; see Figure 3.34. Equation of state for an *ideal* or *perfect gas* is

$$P = \rho R T \tag{3.132}$$

Since the changes di and dP are independent, setting $di = 0$ leads to

$$\left(\frac{\partial \rho}{\partial P}\right)_T = \left(\frac{\partial \rho}{\partial P}\right)_i - \left(\frac{\partial \rho}{\partial i}\right)_P \frac{\left(\frac{\partial T}{\partial P}\right)_i}{\left(\frac{\partial T}{\partial i}\right)_P} \tag{3.126}$$

Similarly, setting $dP = 0$ generates

$$\left(\frac{\partial \rho}{\partial T}\right)_P = \frac{\left(\frac{\partial \rho}{\partial i}\right)_P}{\left(\frac{\partial T}{\partial i}\right)_P} \tag{3.127}$$

Combining eq.(3.126) with eq.(3.124) leads to eq.(3.128).

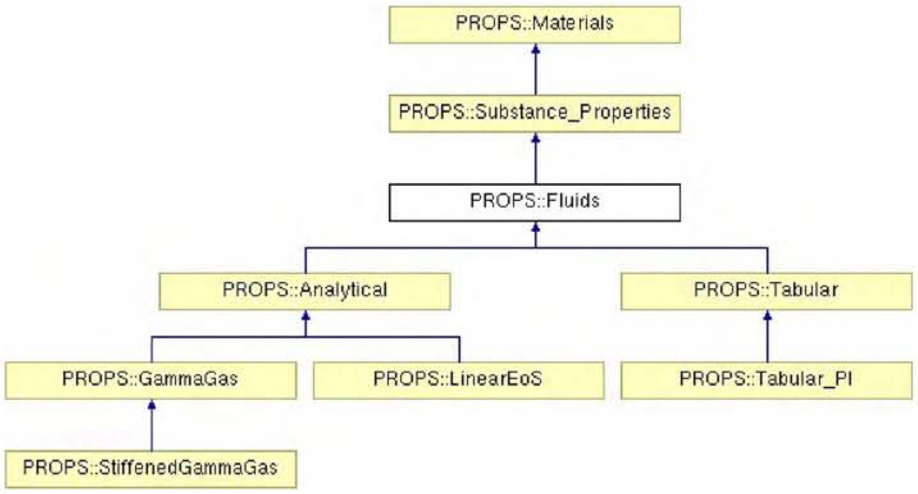

Fig. 3.33 : Inheritance tree for Class `Fluids`.

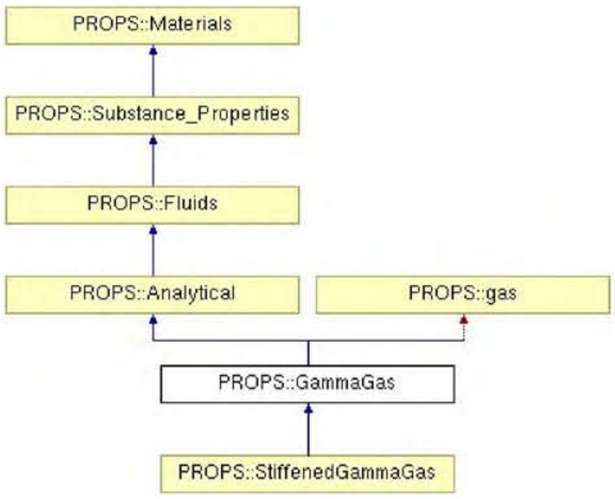

Fig. 3.34 : Inheritance tree for the Class `GammaGas`.

where $R = \frac{R_u}{M}$ is the specific gas constant, with the universal gas constant $R_u \approx 8.31451 \frac{\mathrm{J}}{\mathrm{mol \cdot K}}$ and M is the molecular weight of the gas.

One of the important features of a perfect gas is that its internal energy depends only upon its temperature. Thus, eq.(3.111) reduces to:

$$di = C_v(T)dT \tag{3.133}$$

and C_v depends only upon T. From eq.(3.110), the enthalpy h also depends only on temperature, leading to

$$dh = C_P(T)dT \tag{3.134}$$

Next, one can easily see that

$$dh = C_P(T)dT = di + d\left(Pv\right) = C_v dT + RdT$$

leading to

$$R = C_p - C_v$$

Now, we can introduce γ as the ratio of specific heats [Atk94]:

$$\gamma \equiv \frac{C_P}{C_v} \tag{3.135}$$

The next level of assumption is stating that C_v is independent of temperature, i.e. the assumption of *calorically perfect gas*, leading to the following relation of the specific internal energy and temperature:

$$i(T) = i_0 + C_v\left(T - T_0\right) \tag{3.136}$$

where T_0 and i_0 are some reference constants.

With this, we can write ***"Gamma-Law"*** equation of state as

$$P\left(\rho, i\right) = \rho(\gamma - 1)\left(i - i_0 + C_v T_0\right) \tag{3.137}$$

and important thermodynamic properties are defined as

$C_v = const$	Specific heat at constant volume
$C_P = \gamma C_v$	Specific heat at constant pressure
$\beta(T) = \frac{1}{T}$	Isobaric compressibility
$\chi(P) = \frac{1}{P}$	Isothermal compressibility
$c(P,\rho) = \sqrt{\frac{\gamma P}{\rho}}$	Speed of sound
$\left(\frac{\partial P}{\partial i}\right)_\rho = \rho(\gamma-1)$	
$\left(\frac{\partial P}{\partial \rho}\right)_i = \frac{P}{\rho}$	

(3.138)

Input parameters must be specified in the material input file, which should be located in the folder

“`INPUT/Materials/`”

The following options must be set:

```
[EoSParameters]
  gamma = ... → γ
  sph = ... → C_v
  Tref = ... → T_0
  IEref = ... → i_0
[]
```

3.6.4 Stiffened Gamma Law Gas

Class `StiffenedGammaGas` is another subset of fluids; see Figure 3.35. The ***“stiffened gamma law gas”*** [GZI+76] is a simplest generalization of the gamma law gases eq.(3.137) to fluids and solids:

$$P = (\gamma-1)\,\rho\left(i - i_0 + C_v T_0\right) - \gamma\Pi \tag{3.139}$$

where Π is the so-called “*stiffening parameter*”. This equation of state is known to provide a reasonable approximation of thermodynamic processes in gases, liquids and also in solids under high pressure conditions. The values of parameters γ and

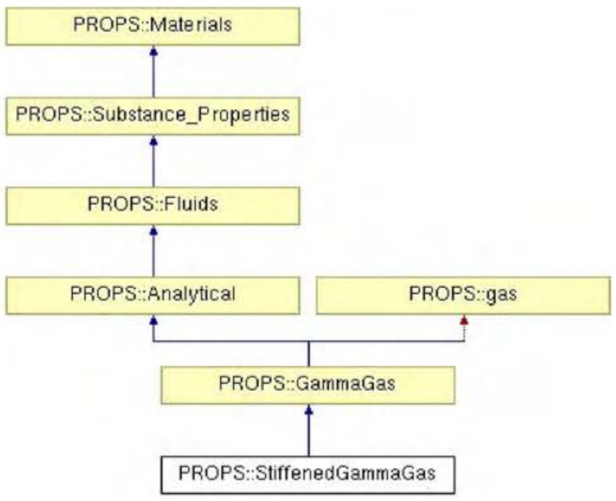

Fig. 3.35 : Inheritance tree for the Class `StiffenedGammaGas`.

Π for some gases, liquids and solids are given in [SA99].

Important thermodynamic properties are defined as

$$\begin{array}{rcl|c}
C_v & = & const & \text{Specific heat at constant volume} \\
C_P & = & \gamma C_v & \text{Specific heat at constant pressure} \\
\beta(T) & = & \frac{1}{T} & \text{Isobaric compressibility} \\
\chi(P) & = & \frac{1}{P+\gamma\Pi} & \text{Isothermal compressibility} \\
c\,(P,\rho) & = & \sqrt{\frac{\gamma(P+\Pi)}{\rho}} & \text{Speed of sound} \\
\hline
\left(\frac{\partial P}{\partial i}\right)_\rho & = & \rho(\gamma-1) & \\
\left(\frac{\partial P}{\partial \rho}\right)_i & = & \frac{P+\gamma\Pi}{\rho} &
\end{array} \tag{3.140}$$

Input parameters are the same as for the Class `GammaGas`, except for additional input:

```
[EoSParameters]
  Pi = ... → Π
[]
```

Linearized EOS (Class `LinearEoS`)

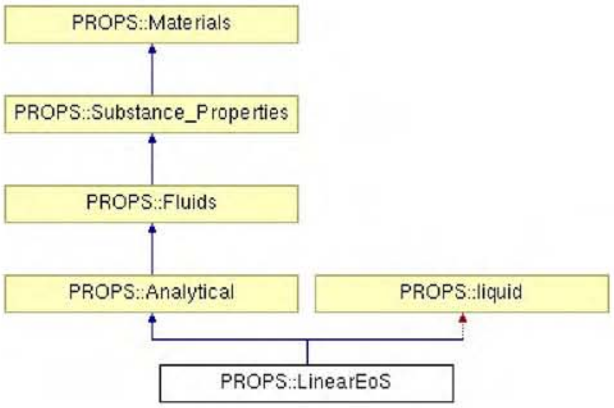

Fig. 3.36 : Inheritance tree for the Class `LinearEoS`.

(Figure 3.36). Another simple approximation for EOS is the following linearization around a fixed state (ρ_0, T_0):

$$P(\rho, T) = P_0 + P_\rho\big|_0 (\rho - \rho_0) + P_T\big|_0 (T - T_0) \tag{3.141}$$

where $P_0 = P(\rho_0, T_0)$, $P_\rho\big|_0 \equiv \left(\frac{\partial P}{\partial \rho}\right)_T (\rho_0, T_0)$ and $P_T\big|_0 \equiv \left(\frac{\partial P}{\partial T}\right)_\rho (\rho_0, T_0)$ are given constants.

Relationship $i(T)$ is defined by eq.(3.136), where C_v is a given constant.

Notably, C_P depends on both temperature and pressure:

$$C_P(T, P) = C_v + \frac{P\ P_T\big|_0}{P_\rho\big|_0 \left(\rho_0 + \frac{P - P_0 - P_T\big|_0 (T - T_0)}{P_\rho\big|_0}\right)^2} \tag{3.142}$$

Accordingly, γ is also a function of temperature and pressure, $\gamma(P, T)$.

The specific volume v as a function of pressure P and temperature T is defined as:

$$v(T, P) = \frac{P_\rho\big|_0}{\rho_0\ P_\rho\big|_0 + P - P_0 - P_T\big|_0 (T - T_0)}$$

which leads to the following *isobaric compressibility*:

$$\beta(\rho) = \frac{P_T|_0}{\rho \, P_\rho|_0} \tag{3.143}$$

and *isothermal compressibility*:

$$\chi(\rho) = \frac{1}{\rho \, P_\rho|_0} \tag{3.144}$$

Finally, the *sound speed* is defined as

$$c(P,\rho) = \sqrt{P_\rho|_0 + \frac{P \, P_T|_0}{C_v \rho^2}} \tag{3.145}$$

and

$$\left(\frac{\partial P}{\partial i}\right)_\rho = \frac{P_T|_0}{C_v}, \quad \left(\frac{\partial P}{\partial \rho}\right)_i = P_\rho|_0$$

Input parameters must be specified in the material input file, which should be located in the folder

“`INPUT/Materials/`”

The following options must be set:

```
[EoSParameters]
  Pref = ... → $P_0$
  Dref = ... → $\rho_0$
  Tref = ... → $T_0$
  IEref = ... → $i_0$
  sph = ... → $C_v$
  dPREdDEN = ... → $P_\rho|_0$
  dPREdTEM = ... → $P_T|_0$
[]
```

Tabulated $(P - i)$ EOS (Class `Tabular_PI`)

This is an important class of materials, which allows for representation of realistic material properties, defined in tabulated form, as a function of fluid pressure (P) and specific internal energy (i). In general, the class `Tabular_PI` allows for modeling of two-phase (liquid/vapor) systems as well. Here, we will focus on the algorithms used for property interpolation in single-phase regions. Property

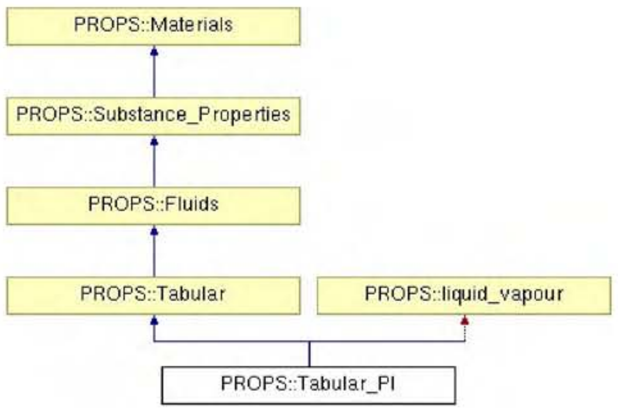

Fig. 3.37 : Inheritance tree for the Class `Tabular_PI`.

approximations in the two-phase zone are described in Section 3.6.7.

The use of the class `Tabular_PI` for tabulated property approximations is most cost-effective when both *pressure* and *specific internal energy* are primary unknowns, such as for example in components of the `Pipe_Pui` family. Class `Tabular_PI` is designed to read tabulated input data from the given file, and apply *smooth and accurate* bi-cubic interpolation on the supplied data set.

I/O (Class `Read_PI_Table`)

Actual I/O of the data is performed by the Class `Read_PI_Table`. The name of the table data file is specified under

```
[Table]
  Table_File = TableFileName
[]
```

in the material input file, located in the folder

"`INPUT/Materials/`"

`Read_PI_Table` will look for the file

"`INPUT/Materials/Tables/TableFileName.dat`"

An example of tabulated data file is given by

"`INPUT/Materials/Tables/Water_at_16MPa.dat`" which defines saturation (liquid and vapour) properties (i.e., specific internal energy $i_{\mathrm{sat,L/V}}$, temperature T_{sat}, density $\rho_{\mathrm{sat,L/V}}$, speed of sound $c_{\mathrm{sat,L/V}}$, isochoric specific heat $c_{v_{\mathrm{sat,L/V}}}$, isobaric specific heat $c_{P_{\mathrm{sat,L/V}}}$, isobaric compressibility $\beta_{\mathrm{sat,L/V}}$, thermal conductivity $\kappa_{\mathrm{sat,L/V}}$, dynamic viscosity $\mu_{\mathrm{sat,L/V}}$, and surface tension $\sigma_{\mathrm{sat,L}}$), as a function of pressure. Then, for each pressure level, one has to specify T, ρ, c, c_v, c_P, β, κ, μ, and σ for each of two single-phase zones (liquid and vapour), tabulated as a function of specific internal energy. The number of pressure levels in the table are defined by the parameter

```
[Define]
  PressureLevels = ...
[]
```

at the beginning of the table file. The number of specific internal energy levels at each pressure level is arbitrary, but must be confined in the range from i_{beg} to i_{end}, defined under

```
[Define]
  IEN_beg = ...
  IEN_end = ...
[]
```

It is generally recommended to supply at least 2-3 points for each of single-phase regions, covering the range more or less uniformly.

Bi-cubic interpolation (Classes of `Table_Props_Evaluator_PI` family)

After reading the data from the input file, Class `Read_PI_Table` does split the range of specific internal energy $\left[i_{\mathrm{beg}}...i_{\mathrm{end}}\right]$ into three zones:

Liquid: From i_{beg} to $i_{\mathrm{sat,L}}\,(P)$

Vapour: From $i_{\mathrm{sat,V}}\,(P)$ to i_{end}

Two-phase: From $i_{\mathrm{sat,L}}\,(P)$ to $i_{\mathrm{sat,V}}\,(P)$

To define boundaries between zones, we build cubic splines using discrete data for saturation specific internal energy.

Next, we can transform each of the irregular-shape zone in the $(P-i)$ formulation, into the regular rectangular shape in the $(P-\underline{x})$ formulation, where

$0 \le \underline{x} \le 1$ is the specific internal energy-based "quality" of the zone. It is defined as

$$\begin{array}{ll} \textbf{Liquid:} & \underline{x} \equiv \frac{i-i_{\text{beg}}}{i_{\text{sat,L}}(P)-i_{\text{beg}}} \\ \textbf{Vapour:} & \underline{x} \equiv \frac{i-i_{\text{sat,V}}(P)}{i_{\text{end}}-i_{\text{sat,V}}(P)} \\ \textbf{Two-phase:} & \underline{x} \equiv \frac{i-i_{\text{sat,L}}(P)}{i_{\text{sat,V}}(P)-i_{\text{sat,L}}(P)} \end{array} \tag{3.146}$$

Fig. 3.38 : Inheritance tree for the Class `Table_Props_Evaluator_PI`.

It is rather straightforward to demonstrate that the transformation of the derivatives in the $(P-i)$ formulation into the derivatives in the $(P-\underline{x})$ formulation is accomplished using the following formulas:

$$\begin{array}{lll} \left.\frac{\partial \Phi}{\partial P}\right|_{\underline{x}} & = & \left.\frac{\partial \Phi}{\partial P}\right|_{i} + \left.\frac{\partial \Phi}{\partial i}\right|_{P} \left.\frac{\partial i}{\partial P}\right|_{\underline{x}} \\ \left.\frac{\partial \Phi}{\partial \underline{x}}\right|_{P} & = & \left.\frac{\partial \Phi}{\partial i}\right|_{P} \left.\frac{\partial i}{\partial \underline{x}}\right|_{P} \end{array} \tag{3.147}$$

where Φ is a scalar, while $\left.\frac{\partial i}{\partial P}\right|_{\underline{x}}$ and $\left.\frac{\partial i}{\partial \underline{x}}\right|_{P}$ can be computed using definition eq.(3.146).

Property evaluations in each of these three zones are accomplished using Classes of `Table_Props_Evaluator_PI` family, Fig. 3.38. Here, we will summarize interpolation in the single-phase liquid and vapour zones. Property evaluation in two-phase zone is described in Section 3.6.7.

There are five material properties which must be smoothly interpolated as functions of P and $\underline{x}(i, P)$:

1. density, $\rho(P, \underline{x})$,
2. temperature, $T(P, \underline{x})$,
3. dynamic viscosity, $\mu(P, \underline{x})$,
4. thermal conductivity, $\kappa(P, \underline{x})$, and
5. surface tension, $\sigma(P, \underline{x})$ (relevant only for liquid).

The rest important thermodynamic quantities are derivatives of $\rho\,(P,\underline{x})$ and $T\,(P,\underline{x})$:

$$c = \sqrt{\frac{1}{\left(\frac{\partial \rho}{\partial P}\right)_i}\left(1 - \frac{P\left(\frac{\partial \rho}{\partial i}\right)_P}{\rho^2}\right)} \tag{3.148}$$

$$C_v = \frac{1}{\left(\frac{\partial T}{\partial i}\right)_P - \frac{\left(\frac{\partial T}{\partial P}\right)_i\left(\frac{\partial \rho}{\partial i}\right)_P}{\left(\frac{\partial \rho}{\partial P}\right)_i}} \tag{3.149}$$

$$C_P = \frac{1 - \frac{P}{\rho^2}\left(\frac{\partial \rho}{\partial i}\right)_P}{\left(\frac{\partial T}{\partial i}\right)_P} \tag{3.150}$$

and

$$\beta = -\frac{1}{\rho}\frac{\left(\frac{\partial \rho}{\partial i}\right)_P}{\left(\frac{\partial T}{\partial i}\right)_P} \tag{3.151}$$

Note that the above-given formulas imply thermodynamic consistency and rely on the fact that interpolations of density and temperature are smooth, allowing to accurately compute derivatives $\left(\frac{\partial \rho}{\partial \underline{x}}\right)_P$, $\left(\frac{\partial \rho}{\partial P}\right)_{\underline{x}}$, $\left(\frac{\partial T}{\partial \underline{x}}\right)_P$ and $\left(\frac{\partial T}{\partial P}\right)_{\underline{x}}$, and via eq.(3.147) $\left(\frac{\partial \rho}{\partial i}\right)_P$, $\left(\frac{\partial \rho}{\partial P}\right)_i$, $\left(\frac{\partial T}{\partial i}\right)_P$ and $\left(\frac{\partial T}{\partial P}\right)_i$.

There are two options for smooth interpolations of properties $\Phi = \rho, T, \mu, \kappa$, or σ.

Bi-Cubic spline

This option is activated by setting

```
[Table]
  interpolation = 0
[]
```

in the material input file, located in the folder

"INPUT/Materials/"

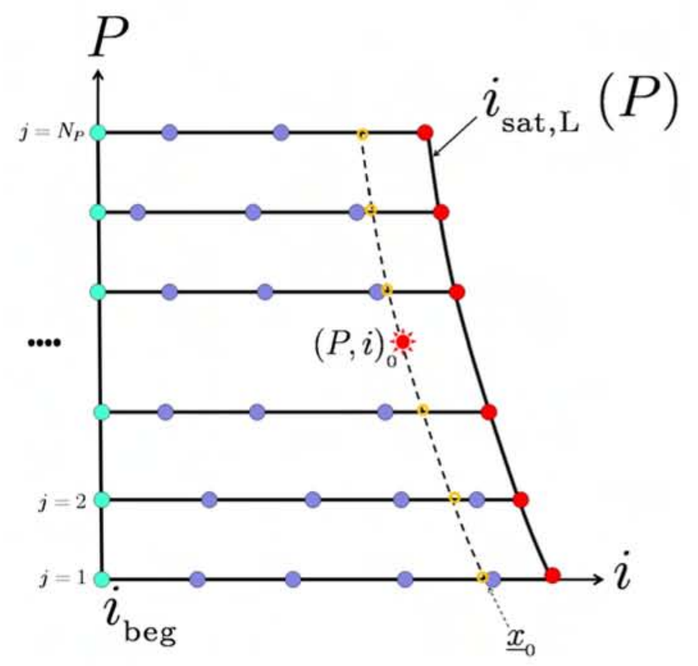

Fig. 3.39 : On bi-cubic spline interpolation of properties.

.

The algorithm is explained by Figure 3.39, illustrating the case for single-phase liquid zone.

1. For each pressure level P_j, we construct and store cubic spline of $\Phi_j\,(i)$ (see [PTVF99] for details of cubic spline interpolation).

2. Suppose we are looking to interpolate properties at $(P, i)_0$.

   - First, compute $\underline{x}_0$.
   - Second, using available splines for each pressure level, compute property Φ and $\left(\frac{\partial\Phi}{\partial i}\right)_P$ for each intersection of the dashed line $\underline{x} = \underline{x}_0$ with each pressure level (orange circles in Figure 3.39).
   - Third, construct cubic splines for evaluation of Φ and $\left(\frac{\partial\Phi}{\partial i}\right)_P$ along the dashed line.
   - Next, using these splines, evaluate Φ_0, $\left.\left(\frac{\partial\Phi}{\partial i}\right)_P\right|_0$ and $\left.\left(\frac{\partial\Phi}{\partial P}\right)_{\underline{x}}\right|_0$.
   - Finally, using $\left.\left(\frac{\partial\Phi}{\partial i}\right)_P\right|_0$ and $\left.\left(\frac{\partial\Phi}{\partial P}\right)_{\underline{x}}\right|_0$ and eq.(3.147), compute $\left.\left(\frac{\partial\Phi}{\partial P}\right)_i\right|_0$

This option is rather expensive, as it involves N_P spline evaluations (including table search with bisection), construction of two new splines along the pressure direction, and finally two spline evaluations. A cheaper option is to utilize bi-cubic interpolation.

Bi-Cubic interpolation

This option is activated by setting

```
[Table]
  interpolation = 1
[]
```

The basic idea is to re-tabulate data in the rectangular-shape $(P - \underline{x})$ domain. The size of this new table is defined by parameters

```
[Table]
  bi3_pre_axis = ...
  bi3_x__axis = ...
[]
```

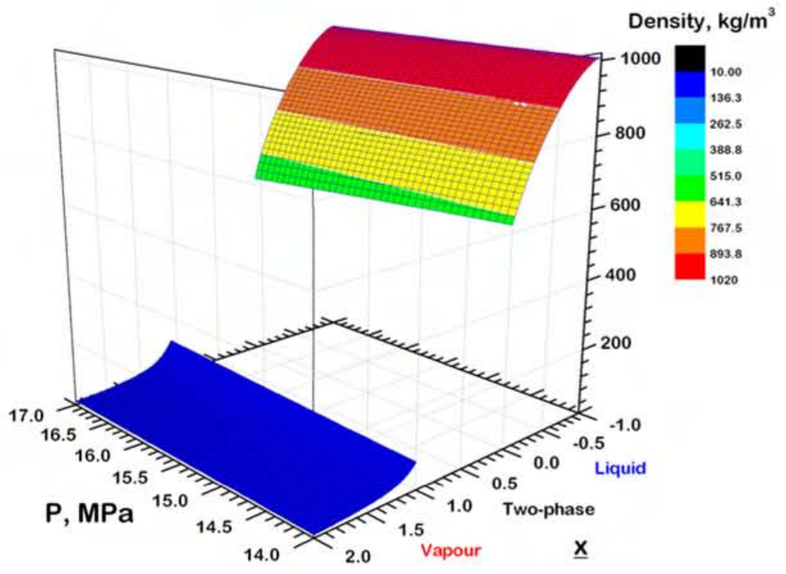

Fig. 3.40 : Example of the `Tabular_PI` table-interpolated density (high-pressure water/steam) as a function of pressure and specific internal energy.

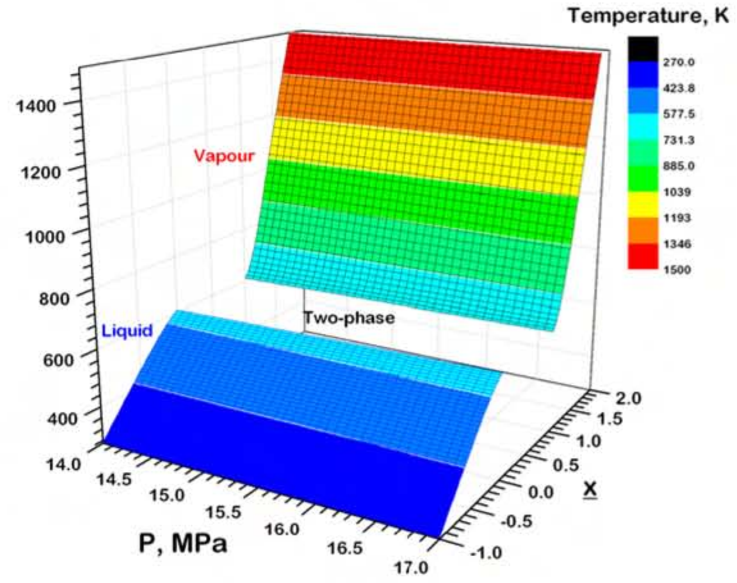

Fig. 3.41 : Example of the `Tabular_PI` table-interpolated temperature (high-pressure water/steam) as a function of pressure and specific internal energy.

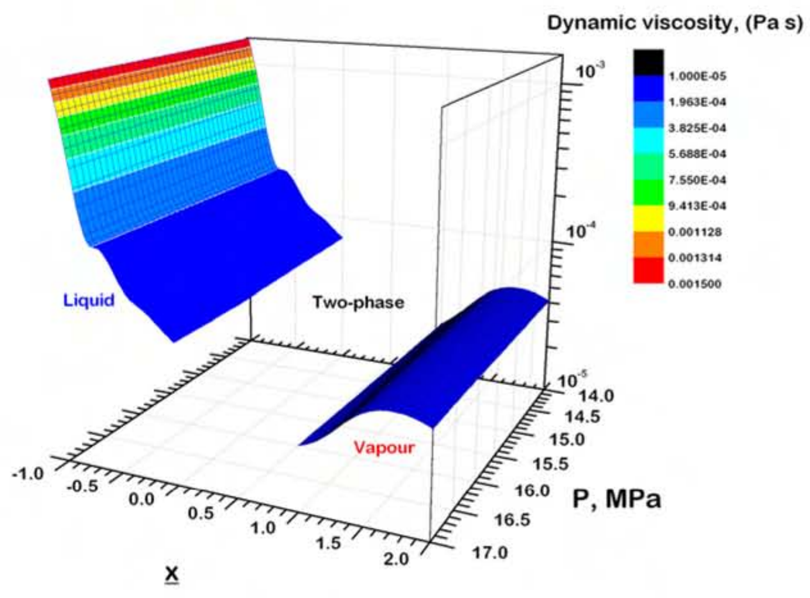

Fig. 3.42 : Example of the `Tabular_PI` table-interpolated dynamic viscosity (high-pressure water/steam) as a function of pressure and specific internal energy.

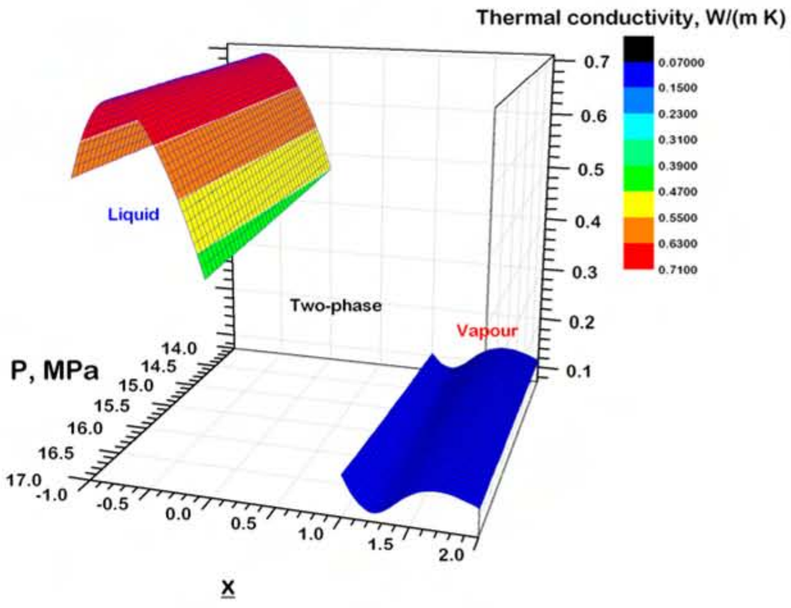

Fig. 3.43 : Example of the `Tabular_PI` table-interpolated thermal conductivity (high-pressure water/steam) as a function of pressure and specific internal energy.

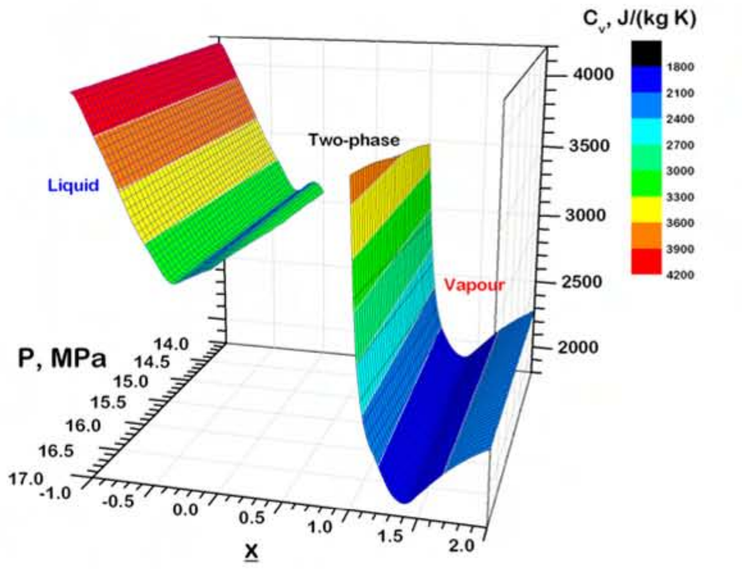

Fig. 3.44 : Example of the `Tabular_PI` table-interpolated isochoric specific heat (high-pressure water/steam) as a function of pressure and specific internal energy.

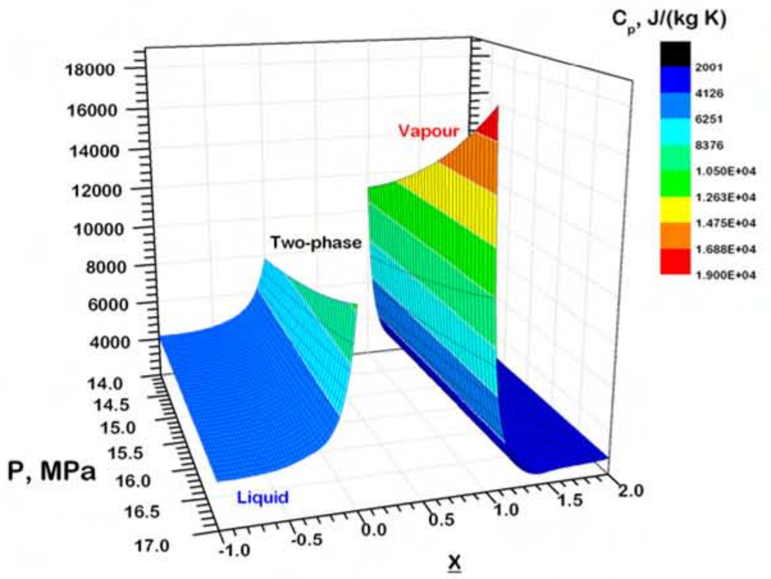

Fig. 3.45 : Example of the `Tabular_PI` table-interpolated isobaric specific heat (high-pressure water/steam) as a function of pressure and specific internal energy.

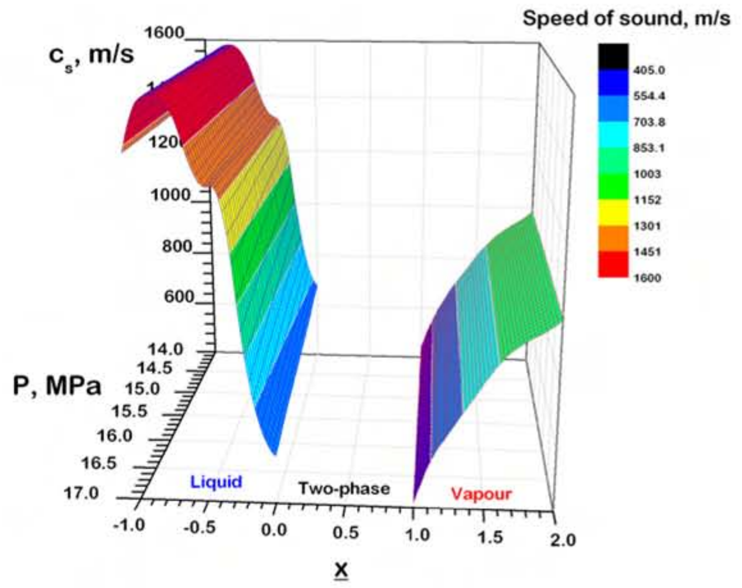

Fig. 3.46 : Example of the `Tabular_PI` table-interpolated speed of sound (high-pressure water/steam) as a function of pressure and specific internal energy.

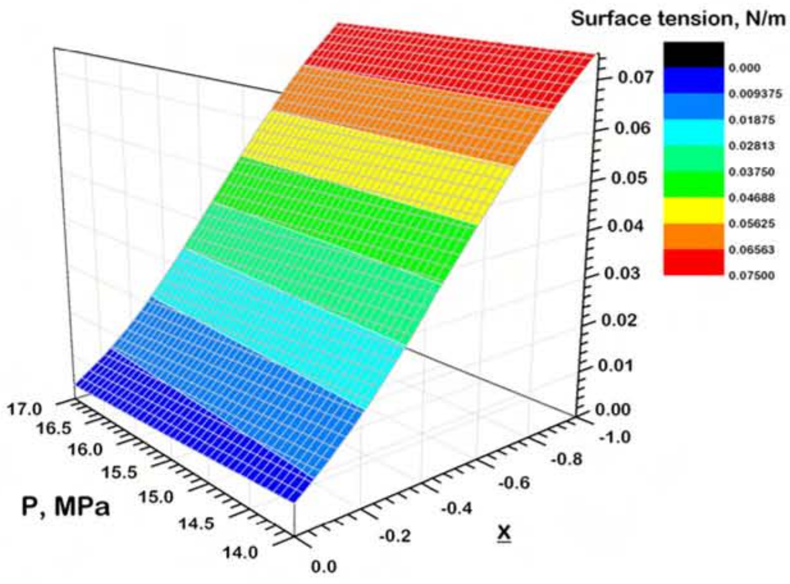

Fig. 3.47 : Example of the `Tabular_PI` table-interpolated surface tension (high-pressure water) as a function of pressure and specific internal energy.

At each point of this new table, we compute Φ using bi-cubic spline algorithm described above. For bi-cubic interpolation, we need also derivatives $\left(\frac{\partial\Phi}{\partial P}\right)_{\underline{x}}$, $\left(\frac{\partial\Phi}{\partial\underline{x}}\right)_{P}$ and cross-derivative $\left(\frac{\partial^2\Phi}{\partial\underline{x}\partial P}\right)$. To compute these derivatives, we perturb point values with some small ΔP and $\Delta\underline{x}$, and use second order finite differencing to estimate point values of the derivatives. This is done at the initiation stage of the class `Table_Props_Evaluator_PI_1phase_Bi3`.

Then, we can use bi-cubic interpolation algorithm described in [PTVF99]. Importantly, for efficiency purposes, 1) we use "hunt" table search algorithm [PTVF99], storing and later providing initial guess each time property evaluation has been invoked. 2) First time we hit an element of the $(P-\underline{x})$ table, we dynamically allocate memory for sixteen coefficients $c_{ij}, i, j = 1, .., 4$ needed for bi-cubic interpolation (see details in [PTVF99]), compute them and store for future use. These coefficients are rather expensive to compute, so we store them for the next time we hit the same element. Then, any subsequent evaluation of interpolated value and its derivative is rather inexpensive sum of the type

$$\begin{aligned}
\Phi\left(P,\underline{x}\right) &= \sum_{i=1}^{4}\sum_{i=1}^{4} c_{ij} t^{i-1} u^{j-1} \\
\Phi_{,P}\left(P,\underline{x}\right) &= \sum_{i=1}^{4}\sum_{i=1}^{4} (i-1) c_{ij} t^{i-2} u^{j-1} \tfrac{dt}{dP} \\
\Phi_{,\underline{x}}\left(P,\underline{x}\right) &= \sum_{i=1}^{4}\sum_{i=1}^{4} (j-1) c_{ij} t^{i-1} u^{j-2} \tfrac{dt}{d\underline{x}}
\end{aligned}$$

(see [PTVF99] for explanation of local element coordinates u and t).

We found that this bi-cubic interpolation is roughly 2.5 times faster than bi-cubic spline algorithm.

Examples of property evaluations using this bi-cubic interpolation algorithm are shown in Figures 3.40-3.47, for tabulated water near 16MPa.

3.6.5 Implicit property evaluations

The above-described algorithms do rely on the fact that pressure P and specific internal energy i are given. Sometimes, it is also necessary to solve inverse problem, for example, given density and pressure, compute specific internal energy,

$i\left(\rho_{0}, P_{0}\right)$. In this case, we use Newton-Raphson algorithm, solving the following nonlinear problem

$$\mathcal{F}(i)=\rho_{0}-\rho\left(P_{0}, i\right)$$

iteratively, where $\rho\left(P_{0}, i\right)$ is coming from the above-described bi-cubic interpolations. Parameters for Newton iterations are specified as

```
[Table]
  MAXIT = The maximum number of iterations
  tol_IEN = Absolute non-linear tolerance
[]
```

For initial guess, we employ a variation of the `zbrak` algorithm from [PTVF99], "looking inward" on an initial "root bracketing" interval. Resolution for initial guess search in this bracketing strategy is defined by parameter

```
[Table]
  search_res = ...
[]
```

Similar algorithm is used when we need $i\left(T, P\right)$, $P\left(i, \rho\right)$, $P\left(i, T\right)$, etc.

3.6.6 Eigensystems

Eigensystem of the hyperbolic part of governing equations in the form eq.(3.1) can be derived by analyzing Jacobian matrices

$$\mathbb{J} \equiv \frac{\partial \vec{\mathcal{H}}}{\partial \vec{\mathcal{U}}} \tag{3.152}$$

Using the eigensystem decomposition of governing equations, one can transform physical variables into the characteristic fields, which allows to efficiently use upwinding schemes.

Euler equations, 1D, Class `EigS_Euler1D`

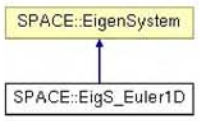

Fig. 3.48 : Inheritance tree for the Class `EigS_Euler1D`.

(Figure 3.48). A general form of the Jacobian matrix for 1D Euler equations in conservative formulation is:

$$\mathbb{J} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{\left(\frac{\partial P}{\partial i}\right)_\rho}{\rho^2}\left(\rho u^2 - E\right) + \left(\frac{\partial P}{\partial \rho}\right)_i - u^2 & u\left(2 - \frac{\left(\frac{\partial P}{\partial i}\right)_\rho}{\rho}\right) & \frac{\left(\frac{\partial P}{\partial i}\right)_\rho}{\rho} \\ u\left(\frac{\left(\frac{\partial P}{\partial i}\right)_\rho}{\rho^2}\left(\rho u^2 - E\right) + \left(\frac{\partial P}{\partial \rho}\right)_i - H\right) & H - \frac{\left(\frac{\partial P}{\partial i}\right)_\rho u^2}{\rho} & u\left(1 + \frac{\left(\frac{\partial P}{\partial i}\right)_\rho}{\rho}\right) \end{bmatrix} \tag{3.153}$$

where $E = \rho\left(i + \frac{u^2}{2}\right)$ and $H = \frac{E+P}{\rho}$. Then, the eigenvalue matrix of the system eq.(3.153) is:

$$\Lambda = \begin{bmatrix} u-c & 0 & 0 \\ 0 & u & 0 \\ 0 & 0 & u+c \end{bmatrix} \tag{3.154}$$

where the sound speed is defined by eq.(3.130).

The Left and the Right eigenvector matrices[20] are:

$$\mathbb{R} = \begin{bmatrix} \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{u-c}{2} & u & \frac{u+c}{2} \\ \frac{H-cu}{2} & H-\Gamma & \frac{H+cu}{2} \end{bmatrix} \tag{3.155}$$

and

$$\mathbb{L} = \begin{bmatrix} 1 + \frac{u}{c} + \frac{u^2-H}{\Gamma} & -\frac{u}{\Gamma} - \frac{1}{c} & \frac{1}{\Gamma} \\ \frac{H-u^2}{\Gamma} & \frac{u}{\Gamma} & -\frac{1}{\Gamma} \\ 1 - \frac{u}{c} + \frac{u^2-H}{\Gamma} & -\frac{u}{\Gamma} + \frac{1}{c} & \frac{1}{\Gamma} \end{bmatrix} \tag{3.156}$$

respectively. The parameter Γ is defined as $\Gamma = \frac{\rho c^2}{\left(\frac{\partial P}{\partial i}\right)_\rho}$. Thus, the vector of characteristic variables is

$$\vec{\Phi} \equiv \mathbb{L}\vec{\mathcal{U}} = \begin{bmatrix} \rho - \frac{P}{\Gamma} \\ \frac{P}{\Gamma} \\ \rho - \frac{P}{\Gamma} \end{bmatrix} \tag{3.157}$$

3.6.7 Multi-phase fluids

Tabulated $(P - i)$ EOS (Class `Tabular_PI`)

This class of material representation allows for modeling two-phase boiling/condensing systems. As discussed in Section 3.6.4, the $(P - i)$ space is split into

[20]These are defined as $\mathbb{JR} = \mathbb{R}\Lambda$, $\mathbb{LJ} = \Lambda\mathbb{L}$, and $\mathbb{L} = \mathbb{R}^{-1}$.

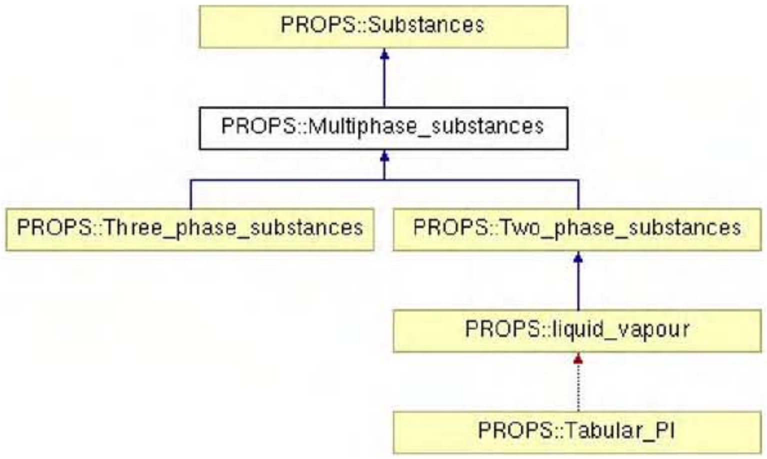
PROPS::Substances
PROPS::Multiphase_substances
PROPS::Three_phase_substances
PROPS::Two_phase_substances
PROPS::liquid_vapour
PROPS::Tabular_PI

Fig. 3.49 : Inheritance tree for the Class `Multiphase_substances`.

three zones: *liquid*, *vapour* and *two-phase*. The first two zones are explained in details in Section 3.6.4. Here, we will be focusing on the *two-phase zone*. In this case, `Tabular_PI` uses property evaluators of the `Table_Props_Evaluator_PI_2phase` family, Figure 3.50.

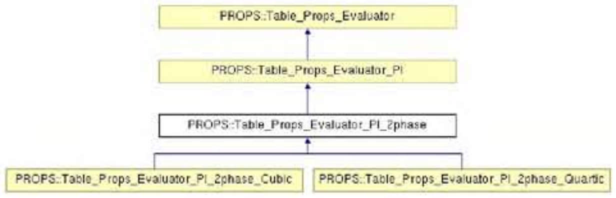
PROPS::Table_Props_Evaluator
PROPS::Table_Props_Evaluator_PI
PROPS::Table_Props_Evaluator_PI_2phase
PROPS::Table_Props_Evaluator_PI_2phase_Cubic
PROPS::Table_Props_Evaluator_PI_2phase_Quartic

Fig. 3.50 : Inheritance tree for the Class `Table_Props_Evaluator_PI_2phase`.

Temperature

Temperature is independent of specific internal energy, and a function of only pressure,

$$T\left(P\right) = T_{\text{sat}}\left(P\right)$$

As a consequence,

$$\begin{aligned} c_P &= \infty \\ c_v &= \infty \end{aligned}$$

Viscosity and Thermal Conductivity

Dynamic viscosity μ and thermal conductivity κ are assumed to be functions of pressure and thermodynamic quality, defined as

$$x\left(P,i\right) \equiv \frac{i - i_{\text{sat,L}}\left(P\right)}{i_{\text{sat,V}}\left(P\right) - i_{\text{sat,L}}\left(P\right)} \tag{3.158}$$

Saturation specific internal energies $i_{\text{sat,L}}\left(P\right)$ and $i_{\text{sat,V}}\left(P\right)$ are computed using cubic splines.

There are two models for $\phi = \mu, \kappa$, currently implemented in the class `Table_Props_Evaluator_PI_2phase`:

1. *Linear-average*,

$$\phi\left(x\left(P,i\right)\right) = \phi_{\text{sat,L}}(1-x) + \phi_{\text{sat,v}}x$$

and

2. *Harmonic-mean*,

$$\phi\left(x\left(P,i\right)\right) = \frac{1}{\frac{x}{\phi_{\text{sat,V}}} + \frac{1-x}{\phi_{\text{sat,L}}}}$$

Density

Definition of effective density in the two-phase zone requires special attention, as it is closely related to the definition of speed of sound,

$$c^2 = \frac{1}{\left(\frac{\partial\rho}{\partial P}\right)_i}\left(1 - \frac{P\left(\frac{\partial\rho}{\partial i}\right)_P}{\rho^2}\right) \tag{3.159}$$

The naive approach of setting density to be linear-average of saturation properties, using thermodynamic quality as a weighting kernel function

$$\rho\left(x\left(P,i\right)\right) = \rho_{\text{sat,L}}(1-x) + \rho_{\text{sat,v}}x$$

turns out to be a bad choice, since it is not only generates discontinuity in sound speed at saturation lines, but also generates non-hyperbolic system with $c^2 < 0$ at $x \geq 0.5$.

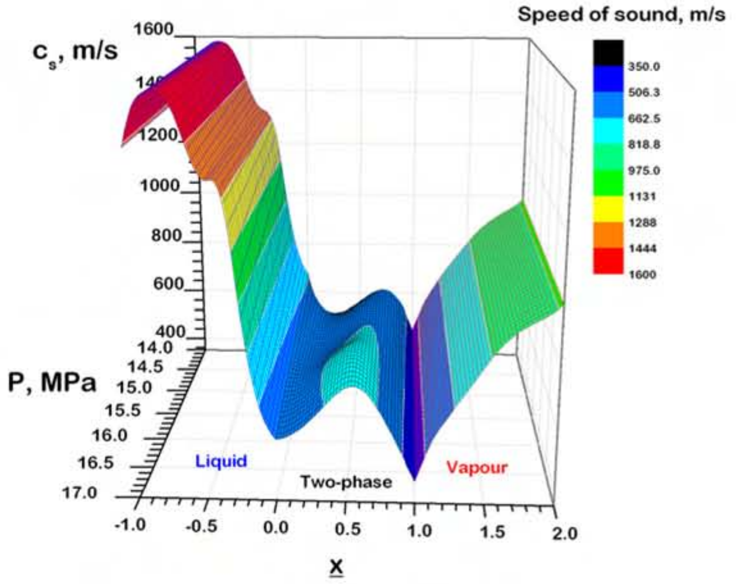

Fig. 3.51 : Example of the `Tabular_PI` table-interpolated speed of sound (high-pressure water/steam) as a function of pressure and specific internal energy. Based on cubic approximation of density in the two-phase zone, $\rho\,(x, P) = \rho_3\,(x, P)$.

A better choice is to use cubic or higher-order approximations of type

$$\begin{aligned}\rho\left(x,P\right) \;=\; & \underbrace{\rho_{\mathrm{sat,L}}\left(x-1\right)^2\left(1+2x\right)+ +x\left[\begin{array}{c}\rho_{\mathrm{sat,V}}x\left(3-2x\right)-\left(i_{\mathrm{sat,L}}-i_{\mathrm{sat,V}}\right)\left(x-1\right)\\ \left(\left(x-1\right)\left.\frac{\partial\rho}{\partial i}\right|_{P,\mathrm{L}}+x\left.\frac{\partial\rho}{\partial i}\right|_{P,\mathrm{V}}\right)\end{array}\right]}_{\text{Cubic term (ensuring continuity of density and sound speed), }\equiv\rho_3(x,P)} \\ & \underbrace{+\sum_{n=4}^{N}a_n\left(P\right)x^2\left(\left(n-3\right)-\left(n-2\right)x+x^{n-2}\right)}_{\text{Quartic}^{+}\text{ correction, helping to ``shape'' speed of sound, }\equiv\rho_{>3}(x,P)}\end{aligned} \tag{3.160}$$

With this, the sound speed behaves smoothly, as demonstrated in Figure 3.51 for water near 16 MPa.

Chapter 4

Testing the Developmental Version

IN this Chapter, we present examples of using RISMC β-2 to perform numerical simulations that help develop a basic understanding of the code performance, particularly with respect to architectural features and solution algorithms implemented in this code version.

4.1 Blowdown test (accuracy, efficiency)

4.1.1 Problem formulation

Let consider a single-phase blowdown problem. A vessel of 100 m^3 in volume, filled with air at 300 K and 100 bars, is placed in a containment of 10^4 m^3 in volume, under 300 K and 1 bar. A pipe, 5m-long and 15 cm in diameter connects the vessel and containment as shown in Figure 4.1. Both the vessel and the containment are modeled by the `Tank1f` components. The pipe is represented by the `Pipe` component. The characteristics-based `Pipe2Tank1f` interface is used to connect the vessel to the pipe and the pipe to the containment. Air is modeled as gamma-gas (Section 3.6.3), with $\gamma = 1.4$, specific heat $C_v = 1000 \frac{\mathrm{J}}{\mathrm{kg\ K}}$, $T_0 = 0$, $i_0 = 0$ and constant dynamic viscosity $\mu = 1.983 \cdot 10^{-5} \frac{\mathrm{kg}}{\mathrm{m\ s}}$. Three test-cases are considered:

- **Inviscid.**
- **Viscid, Filonenko-based friction**, with Class `Friction_pipe_1phase_Filonenko` as friction closure, and

- **Viscid, enhanced friction**, with Class `Friction_pipe_1phase_const`, $\xi = 1$.

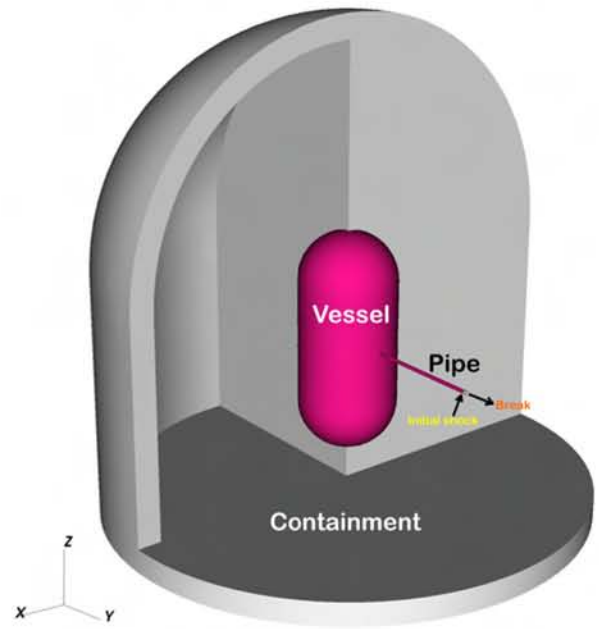

Fig. 4.1 : Outline of the blowdown problem.

As set, this problem represents a harmonic oscillator, i.e., some kind of "manometer".

- In the first stage, there is a **rarefaction** wave propagating towards the vessel.
- Next, the flow **"chokes"** at the exit of the pipe. Characteristics-based interface between the pipe and the containment naturally captures choking, without special "modeling" usually utilized in legacy codes. The gas is discharged from the vessel to the containment, within approximately 40 seconds equalizing pressures in both. During this process, the vessel cools down significantly, while the containment pressure slightly increased.
- If there is no any dissipative mechanisms introduced in the problem (numerical or physical), the pressure in the vessel "overshoots" due to inertia, and the harmonic **oscillations** begin. These oscillations with continue indefinitely in the inviscid case and high-order space-time discretizations.

- In the case of friction, these oscillations will be damped, until **no-flow steady-state** is established.
- Under very strong dissipation (real case), the friction will remove oscillation completely, and the pressures in the vessel and containment will smoothly equilibrate (**without oscillations**), establishing no-flow steady state.

Since this problem involves shock dynamics, only first-order convergence in space is expected. We will use finite-volume cRDG$_0$ scheme with van Albada-2 limiter. The base grid size is set to $\Delta x = 5$cm. Richardson-extrapolation-based convergence is demonstrated in Figure 4.12.

Three fully-implicit time discretization schemes are used and compared – the `BEuler`, the `BDF2` and the `ESDIRK3`, Section 3.2.2. For dynamic time control, we use the CFL-based strategy (Section 3.2.4).

4.1.2 Computational results

Computational results are presented in Figures 4.4-4.9.

First, we show comparison of inviscid and viscid (Filonenko-based) simulations, in Figures 4.2, 4.3, 4.8, 4.9, 4.6 and 4.7. In both cases, we utilized second-order BDF$_2$ time discretization scheme. It can be seen that the inviscid case results in oscillations of both pressure and temperature (see also Figures 4.4 and 4.5). With each "overshoot" of pressure, the flow in pipe reverses, bringing warm air from the containment to the vessel. Dissipation due to the second order time discretization is very low, and since no physical dissipation introduced in the system, the oscillations will persist indefinitely. As one can see from Figure 4.10, Mach number in the pipe is also oscillatory, being able to accelerate during each flow reversal to Mach=1 and choke.

In the case of the viscid Filonenko-based simulations, the friction in the pipe effectively removes energy from the system, damping oscillations, as demonstrated in Figures 4.2, 4.3 and 4.10. Notably, the Filonenko-based friction law is based on *steady-state fully-developed* correlations, which apparently underpredict friction losses in the considered case of highly developing oscillatory flow. In the reality, the friction losses are significantly higher, due to the losses in the multidimensional shock dynamics in both vessel and containment (not represented

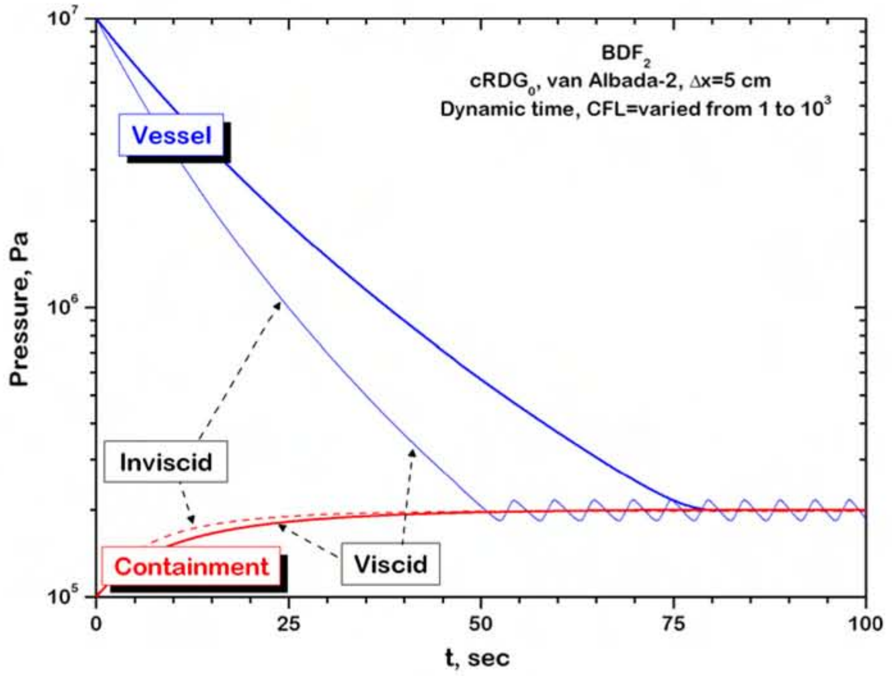

Fig. 4.2 : Inviscid vs. viscid blowdown problems. Comparison of the vessel/containment pressure histories.

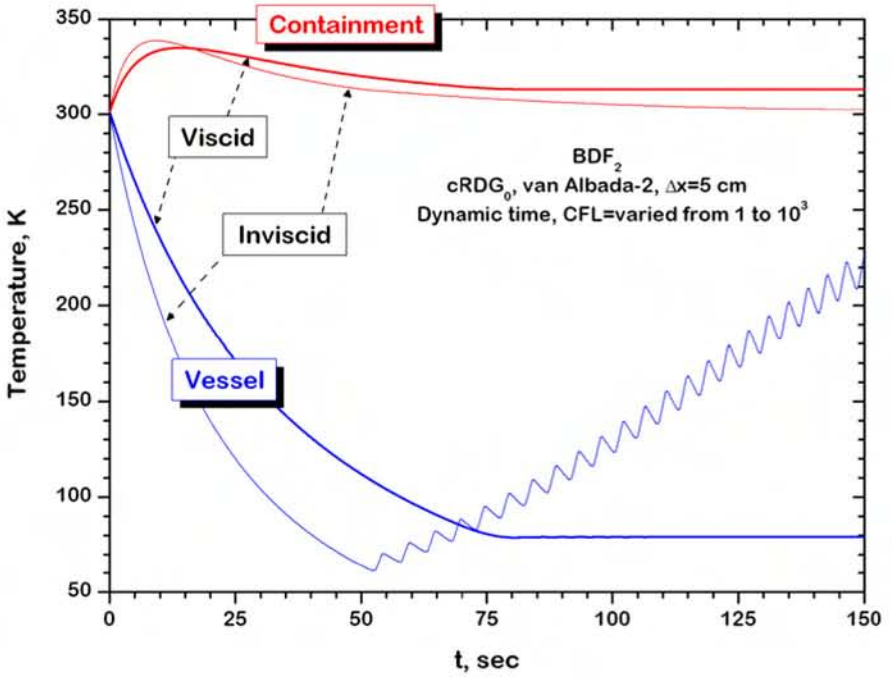

Fig. 4.3 : Inviscid vs. viscid blowdown problems. Comparison of the vessel/containment temperature histories.

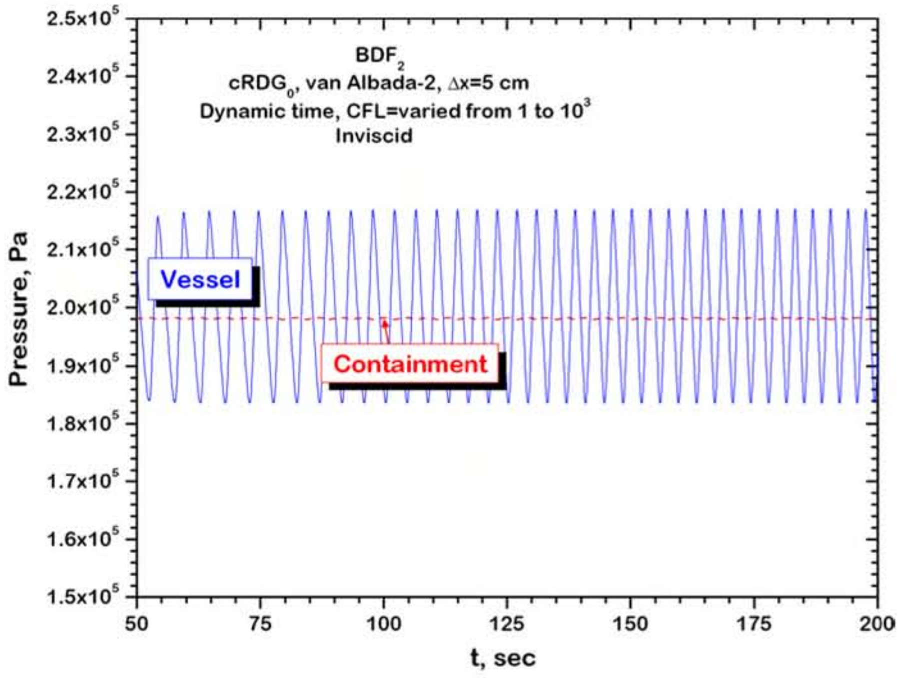

Fig. 4.4 : Dynamics of the pressure in the vessel and containment for inviscid blowdown problem.

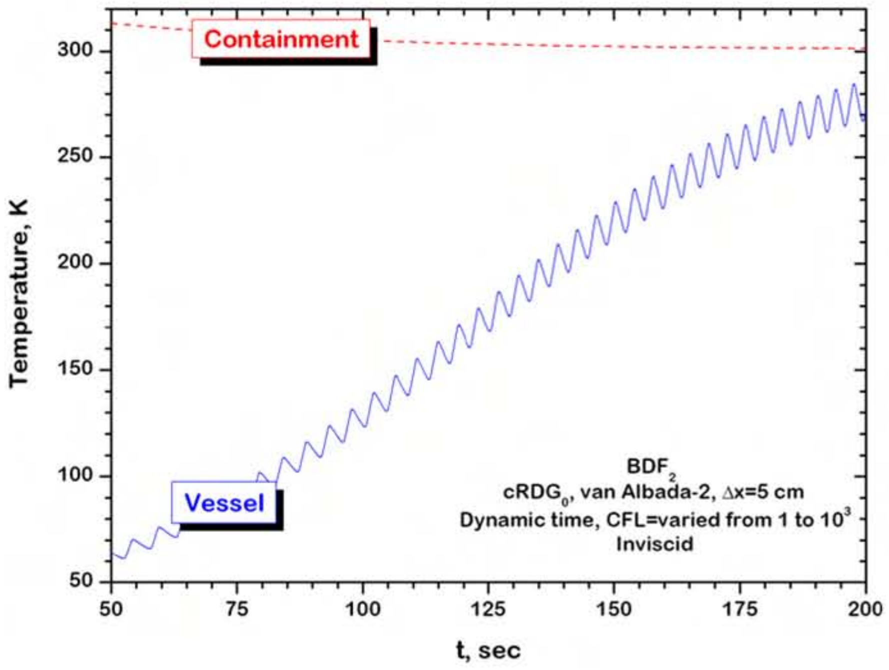

Fig. 4.5 : Dynamics of the temperature in the vessel and containment for inviscid blowdown problem.

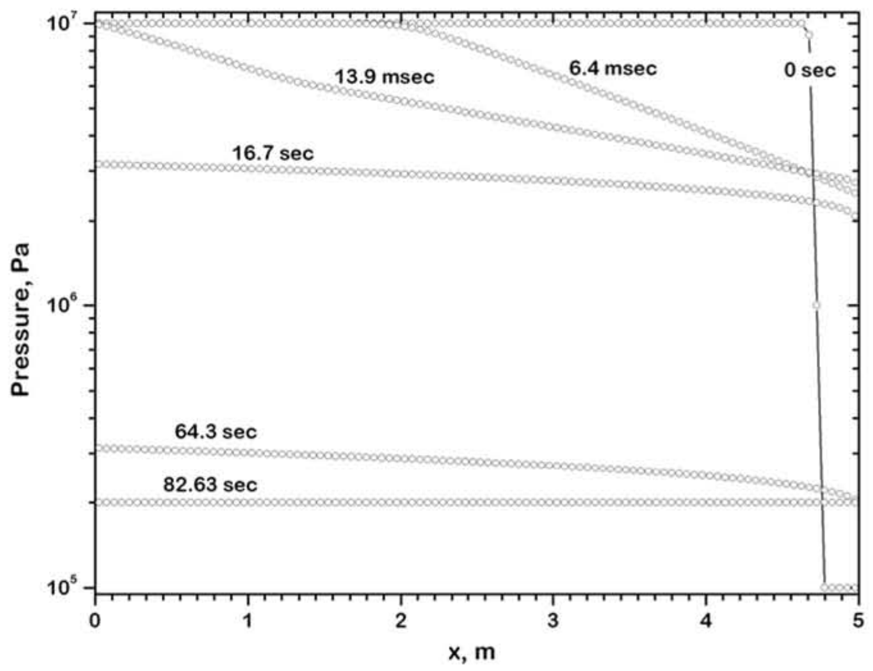

Fig. 4.6 : Dynamics of the pressure profile for Filonenko-based viscid case. van-Albada-2-limited cRDG$_0$, BDF$_2$, $\Delta x = 5$ cm.

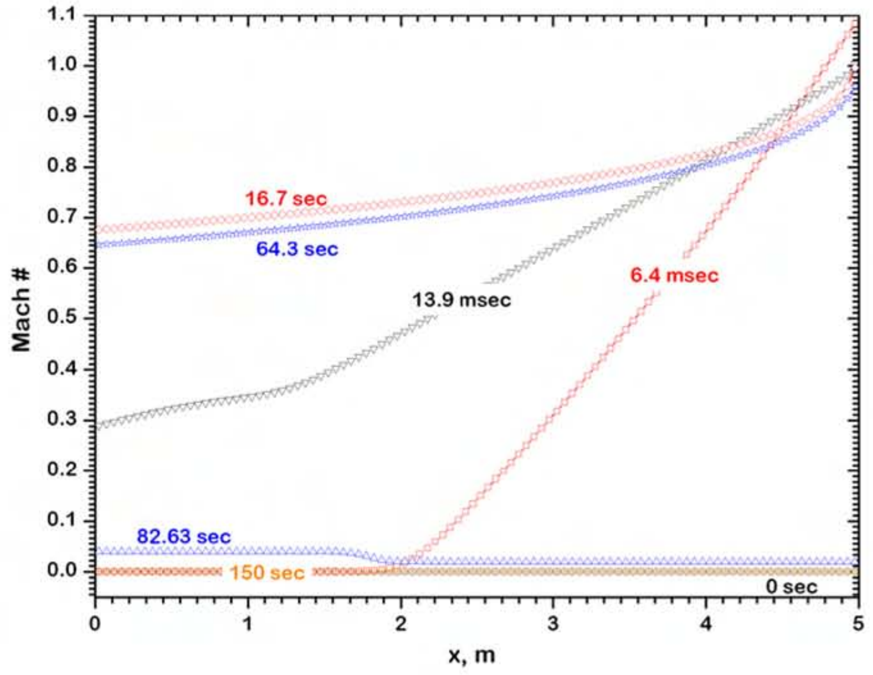

Fig. 4.7 : Dynamics of the Mach number profile for Filonenko-based viscid case. van-Albada-2-limited cRDG$_0$, BDF$_2$, $\Delta x = 5$ cm.

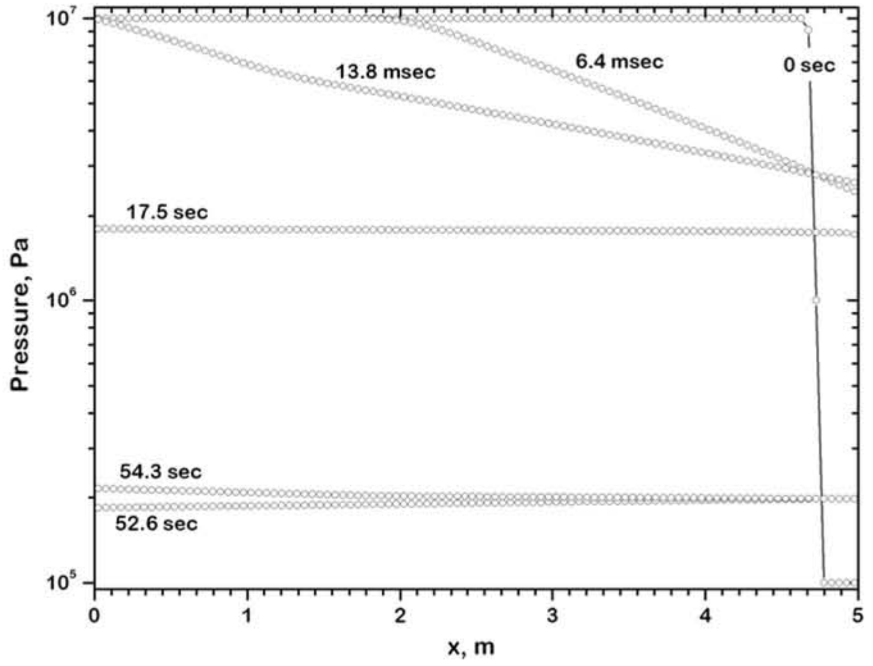

Fig. 4.8 : Dynamics of the pressure profile for inviscid case. van-Albada-2-limited cRDG$_0$, BDF$_2$, $\Delta x = 5$ cm.

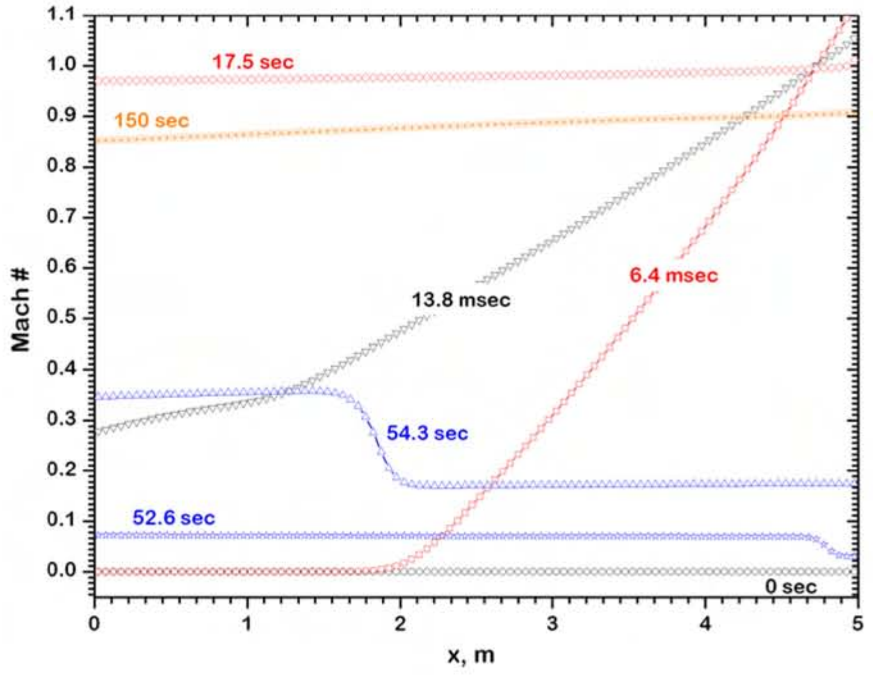

Fig. 4.9 : Dynamics of the Mach number profile for inviscid case. van-Albada-2-limited cRDG$_0$, BDF$_2$, $\Delta x = 5$ cm.

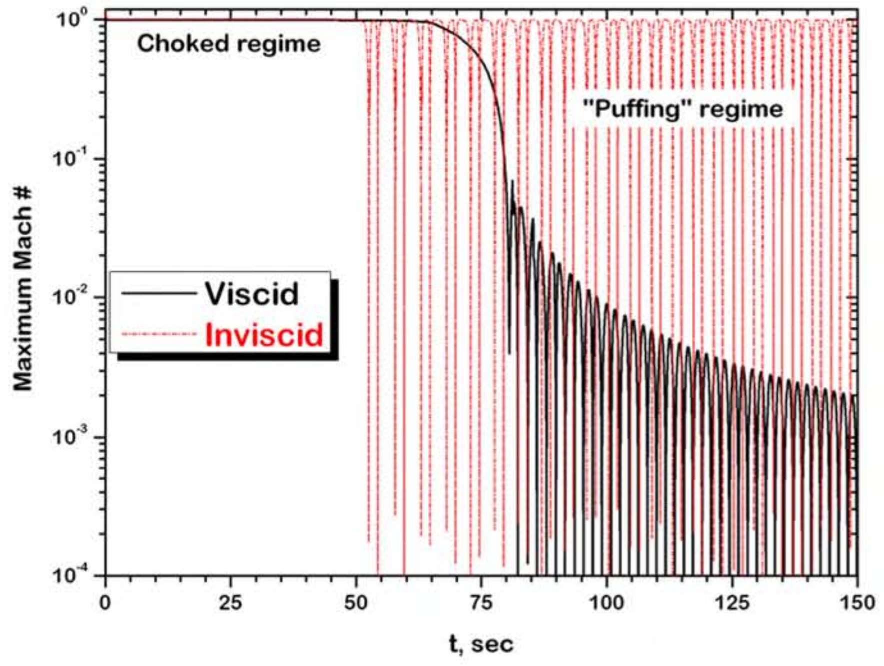

Fig. 4.10 : Dynamics of the maximum Mach number. Viscid vs. inviscid case. van-Albada-2-limited cRDG$_0$, BDF$_2$, $\Delta x = 5$ cm.

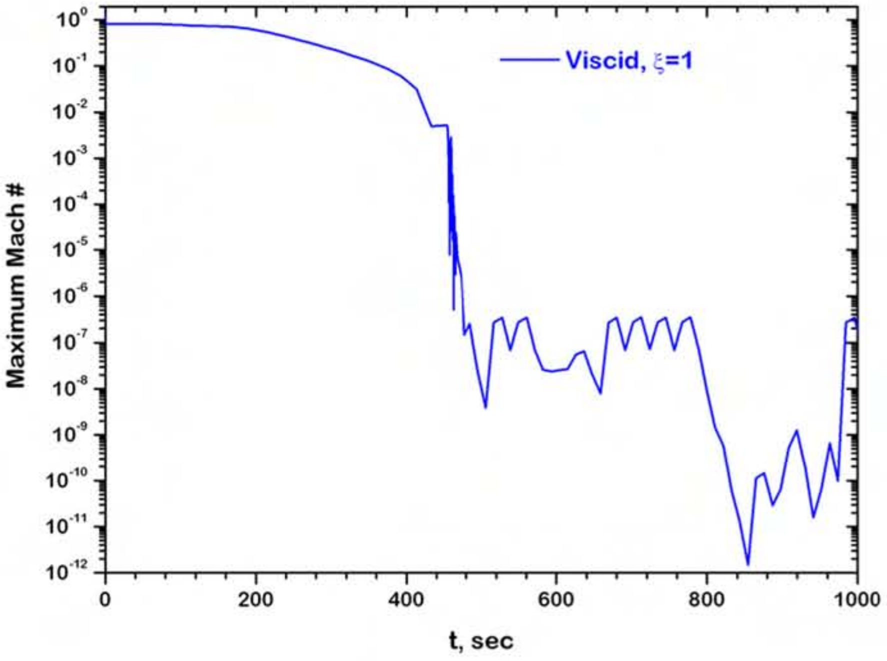

Fig. 4.11 : Dynamics of the maximum Mach number for enhanced friction. van-Albada-2-limited cRDG$_0$, BDF$_2$, $\Delta x = 5$ cm.

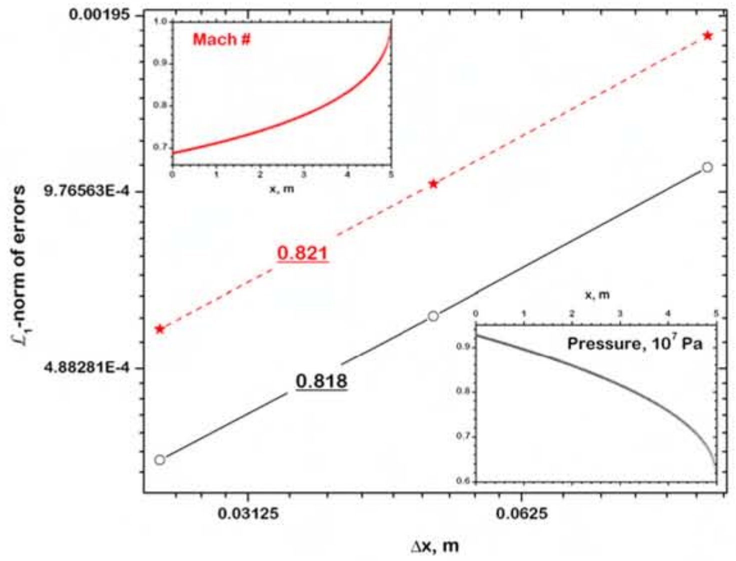

Fig. 4.12 : Space convergence study. Simulations were run with `RK3-TVD` scheme, at $\text{CFL}_{\text{acou}} \approx 1$, till $t = 1$sec. Correspondent pressure and Mach number profiles are also shown.

here at all), and the entrance effects/developing/oscillatory flows. Thus, we expect no oscillations in the real physical systems. This problem clearly exemplifies the need for more complex closures and higher-fidelity components. To demonstrate the complete damping of oscillation with significant friction losses, we run a case with constant-friction $\xi = 1$. The results shown in Figure 4.11 demonstrate almost no-oscillation transient[1].

The effects of the numerical dissipation due to low-order time discretization are shown in Figures 4.13 and 4.14. The first order `BEuler` scheme quickly damps oscillations, effectively draining the energy from the system. Obviously, this discretization scheme is not adequate when the modeling of physical instabilities is necessary (like in the BWR instability problems). Both the second-order `BDF2` and the third-order `ESDIRK3` schemes perform comparably, without draining the energy from the system oscillations.

Finally, we demonstrate efficiency of the time stepping by using implicit schemes and dynamic time control, in Figures 4.15 and 4.16. As one can see, we are able

[1]In fact, only a few very small oscillations took place at time around ≈ 435sec.

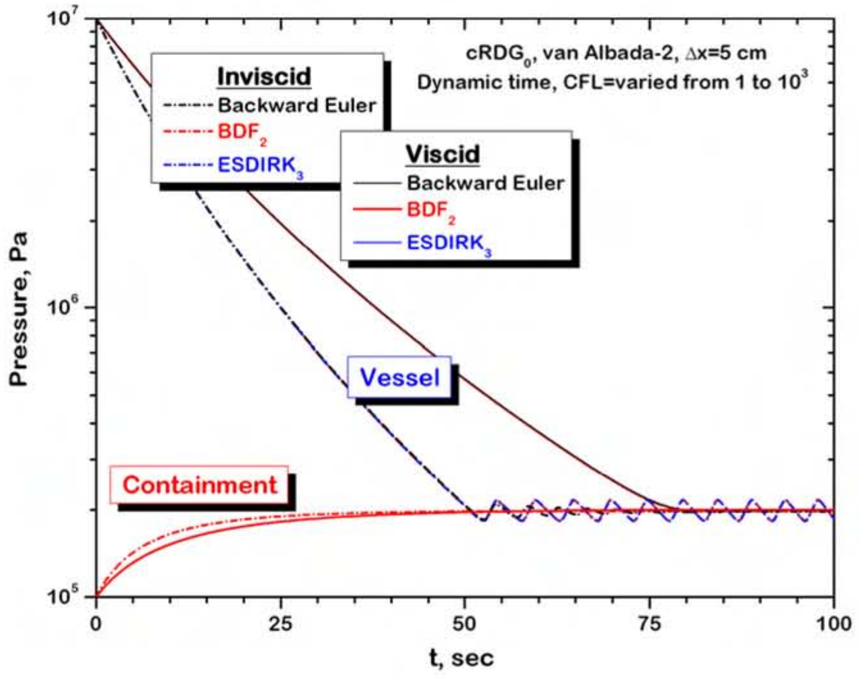

Fig. 4.13 : Effects of time discretization on the vessel/containment pressure histories.

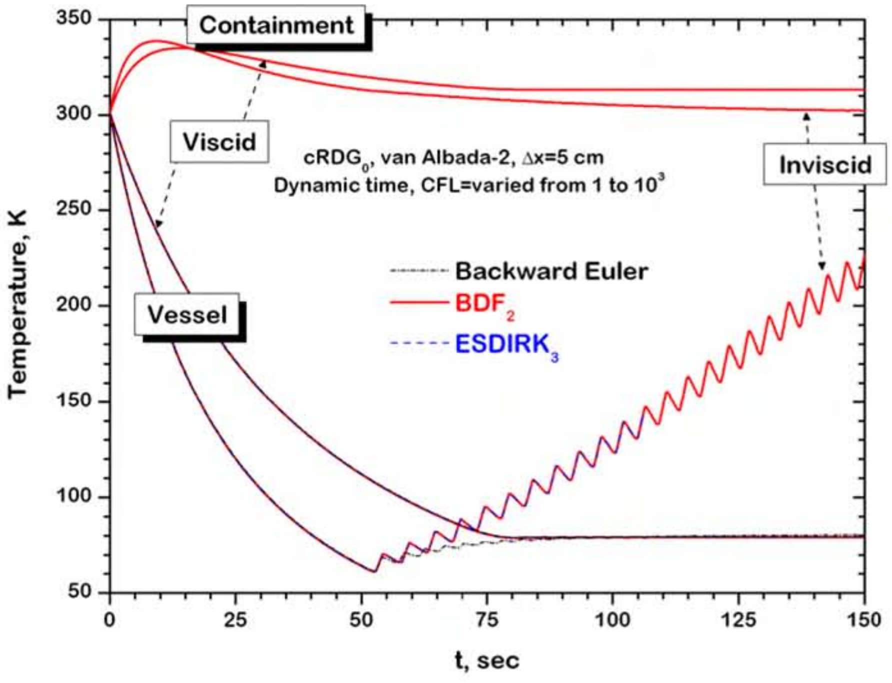

Fig. 4.14 : Effects of time discretization on the vessel/containment temperature histories.

to run robustly and accurately simulations, stepping 1000 times over the acoustic Courant limit, and up to 800 times over the material Courant limit, Figure 4.15.

In the case of enhanced friction, our time steps approximately 10^5 larger than those due to acoustic or material CFL. It is instructive to note that the $\text{CFL}_{\text{mat}} = 1$ is the theoretical stability limit for ICE-based algorithm in legacy system thermal-hydraulics codes. This implies that, for this test problem, we can take hundreds or thousands times larger time steps than RELAP5-generation codes, without numerical instability problems and any loss of accuracy.

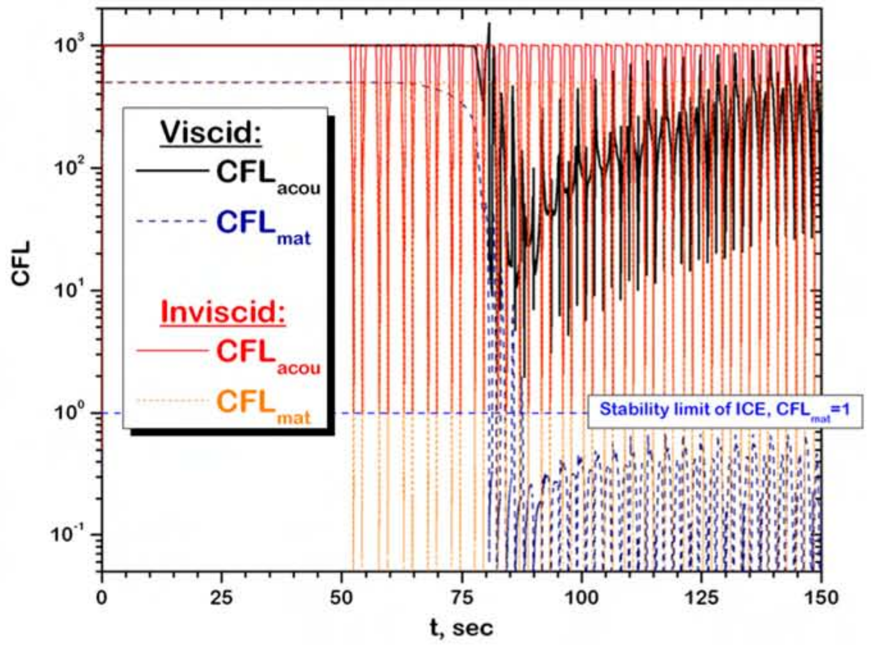

Fig. 4.15 : Dynamics of the instantaneous acoustic and material CFL numbers. Viscid vs. inviscid case. van-Albada-2-limited cRDG$_0$, BDF$_2$, $\Delta x = 5$ cm.

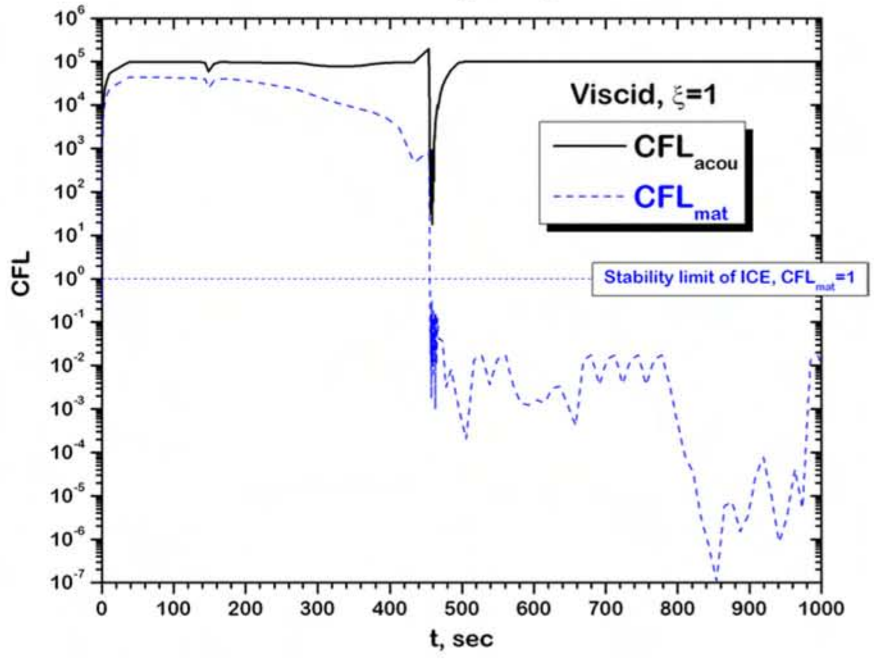

Fig. 4.16 : Dynamics of the instantaneous acoustic and material CFL numbers. Enhanced friction $\xi = 1$ case. van-Albada-2-limited cRDG$_0$, BDF$_2$, $\Delta x = 5$ cm.

4.2 Multi-Component Close-Loop Configuration

We use the term "VR_1" reactor to refer to the first, most simplified "virtual reactor" modeled in RISMC β-2 (Figure 4.17). It is a thermalhydraulics loop, 10m-tall and 10m-wide, made of 20 components: eight `Pipes`, four `Elbows`, a `Pump`, a `Pressurizer`, a core (`PipeQw` + `dummyNeutronics`), and heat exchanger (HX) `PipeTw`, and three controllers (`PumpPrescribedHistory`, `PrzPrescribedHistory` and `CRPrescribedHistory`).

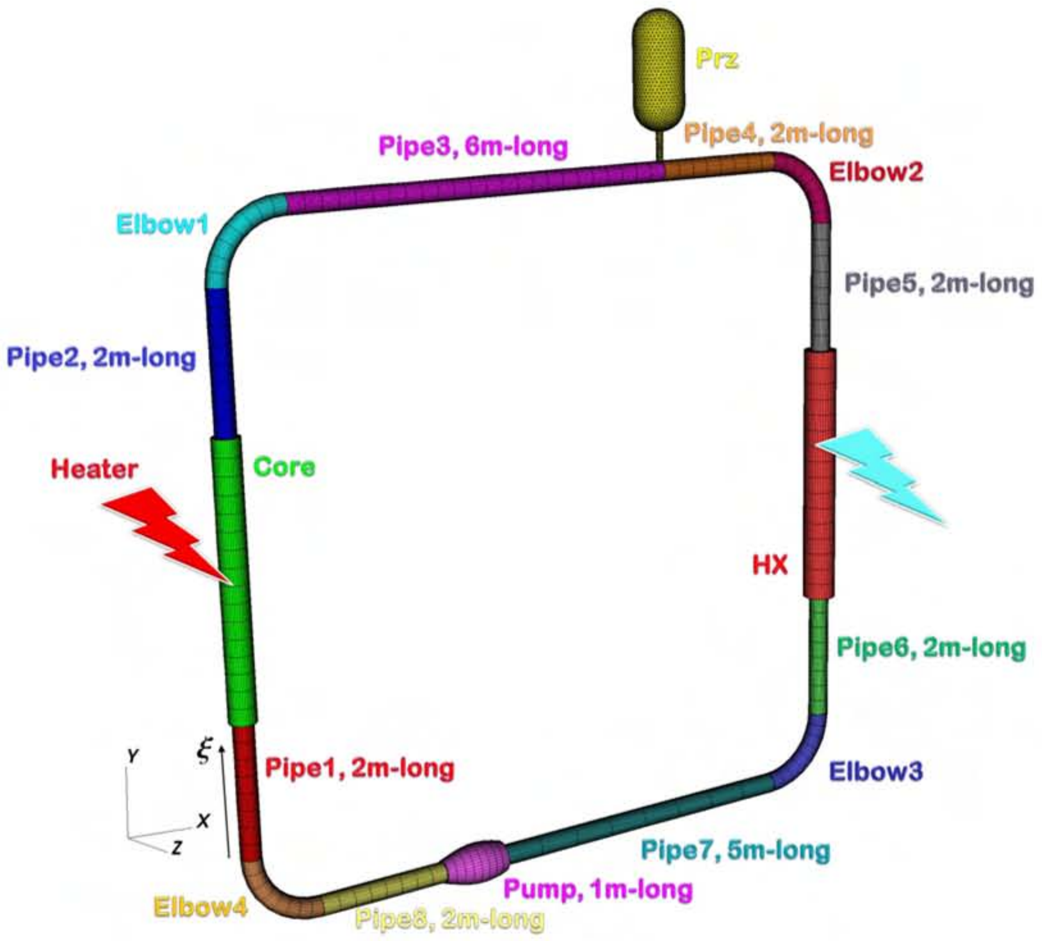

Fig. 4.17 : Formulation of the "VR_1" reactor problem.

- The elbows (type `Elbow`) are 1m in radius `_rad_torus`, nodalized uniformly along the arc coordinate, with the base mesh of 4 elements.

- The pipes are of type `Pipe`. Pipe3 is nodalized with the base mesh of 12 elements, all the rest pipes are nodalized with 4 elements.

- The core consists of two components – the heated pipe (`PipeQw`) and the heater (`dummyNeutronics`. The heated pipe is nodalized with the base mesh of 8 elements. The nominal power of the heater (Class `dummyNeutronics`) is 4 MW.

- The heat exchanger (HX) is represented by the component of type `PipeTw` and nodalized with the base mesh of 8 elements. We utilized constant-Nusselt wall heat transfer closure (Class `HTC_pipe_1phase_const`), with the constant wall temperature $T_{\text{wall}} = 600$ K, and the enhanced heat transfer rate, $Nu = 10^4$. The smoothing edge heat transfer effects are represented by $\delta^{(\text{HTC})}_{\text{edge}} = 0.1$

- The pump is 1m-long, and nodalized with the base mesh of 4 elements. The pump model is the simplest constant pump head, $\Delta P_{\text{Pump,nominal}} = 5 \cdot 10^4$ Pa, without additional pump heating, $\mathcal{S}_E^{(\text{Pump,H})} = 0$.

- The pressurizer is modeled with `dummyPrz`, designed to keep loop's pressure at nominal 16 MPa.

The working fluid is water at high-pressure, represented by a linearized EOS, described in Sections 3.6.4.

The sequential mesh refinement is provided by setting

```
[General]
  mesh_refine = 2, 4, 8, 16, 32
[]
```

corresponding to the total mesh refined from 153 to 2448 elements. All `Pipe`-based components (including Pump, Core and HX) are 15 cm in radius. The friction law was set to Filonenko-based, Class `Friction_pipe_1phase_Filonenko`. Hydraulic diameters of both Core and HX is set to 15 cm.

4.2.1 Steady-state

Initial conditions for are constant-temperature $T = 600$ K, motionless, and linear (in the y-direction) pressure profile, according to the gravity head ($g = -9.8\ \frac{\mathrm{m}}{\mathrm{s}^2}$). The pressure at the top of the loop is set to 16 MPa.

The reactor is brought to the "steady-state" in two stages:

1. <u>The first 10^4 sec</u>: the pump is on, and the heater is off.
2. <u>$10^4 \div 2 \cdot 10^2$ sec</u>: the heater is on.

4.2.2 Power transient

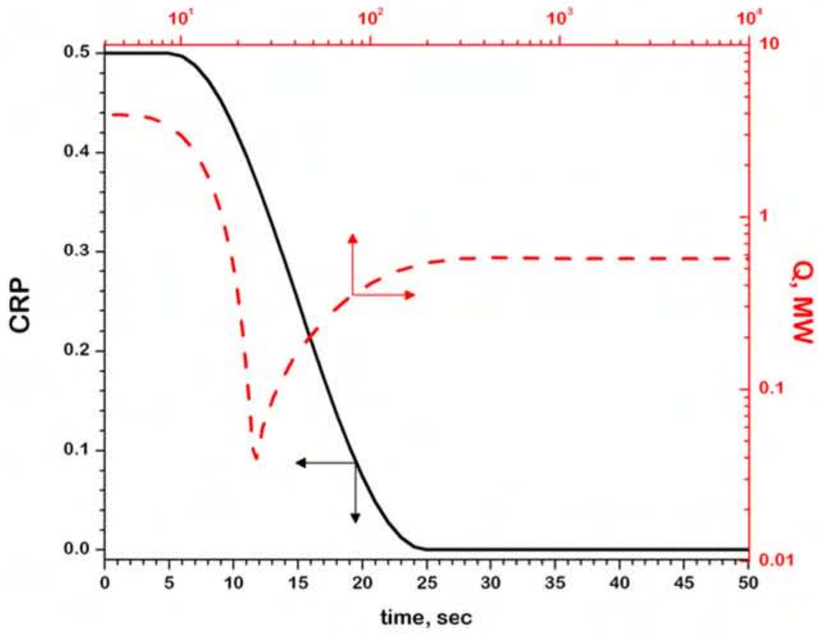

Fig. 4.18 : Dynamics of the control rod position and power for the SCRAM transient. Solution with 304 elements, $\mathrm{cRDG_0}$, and $\mathrm{BDF_2}$ schemes.

In the first transient test, we fully inserted control rods within 20 sec, as shown in Figure 4.18. The corresponding power response is shown in the same Figure in the red. The power first dropped down, during the first 5 sec of the transient, causing the coolant temperature to drop. Then, the temperature Doppler feedback is taking off, bringing the power to $\approx 14\%$ of nominal, after about 30 sec of the

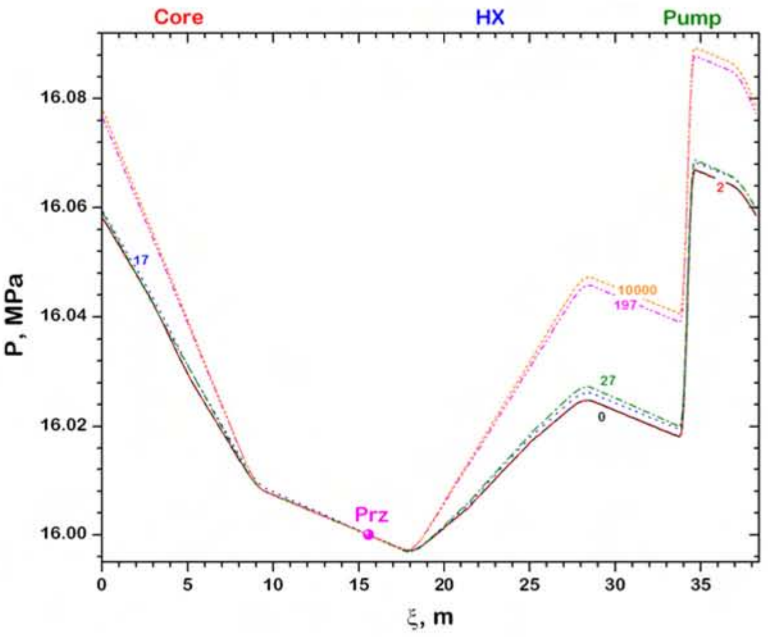

Fig. 4.19 : Pressure distribution for the SCRAM transient. Solution with 304 elements, cRDG_0, and BDF_2 schemes.

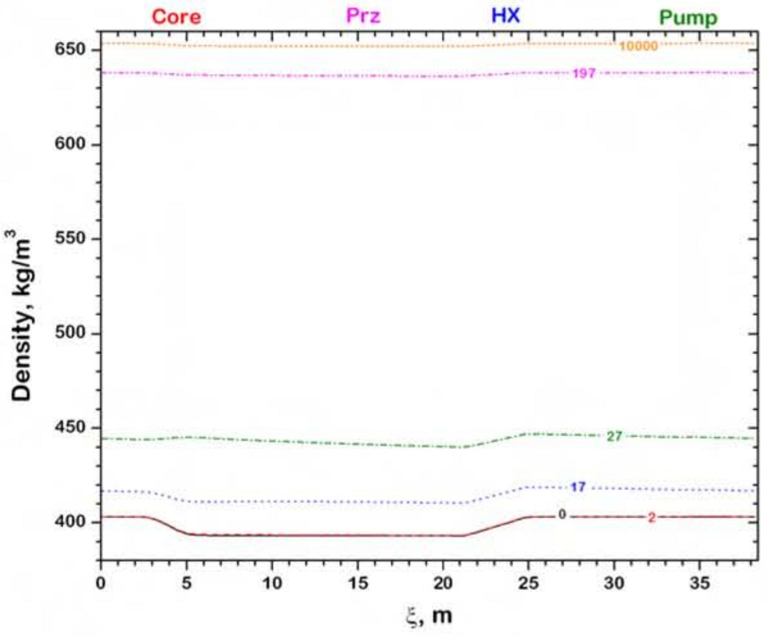

Fig. 4.20 : Density distribution for the SCRAM transient. Solution with 304 elements, cRDG_0, and BDF_2 schemes.

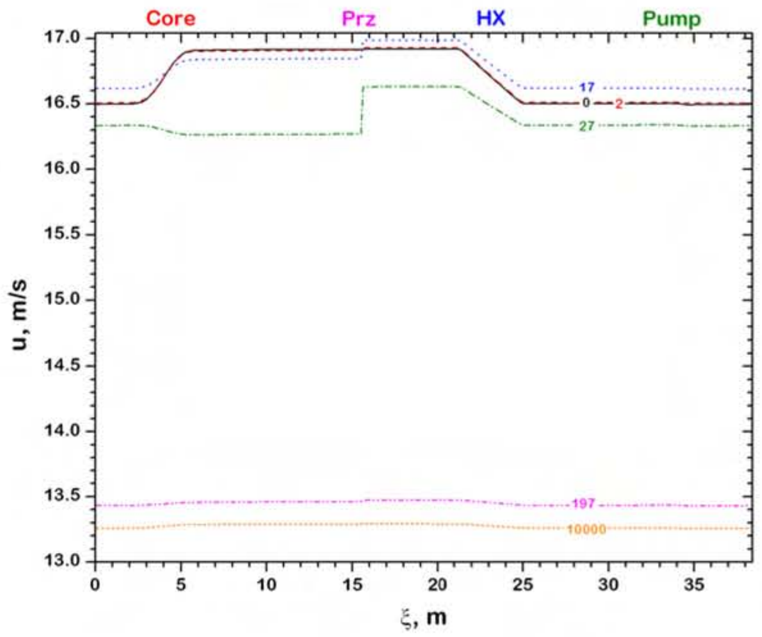

Fig. 4.21 : Velocity distribution for the SCRAM transient. Solution with 304 elements, cRDG_0, and BDF_2 schemes.

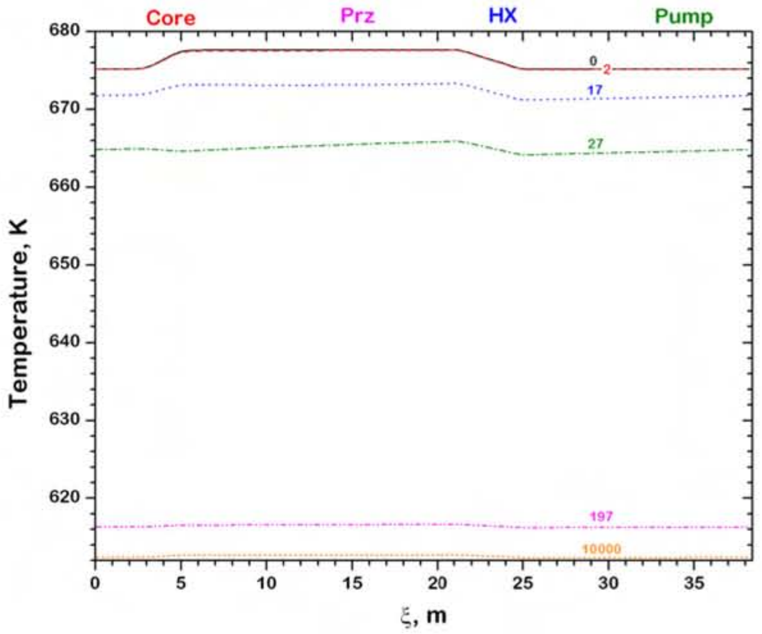

Fig. 4.22 : Temperature distribution for the SCRAM transient. Solution with 304 elements, cRDG_0, and BDF_2 schemes.

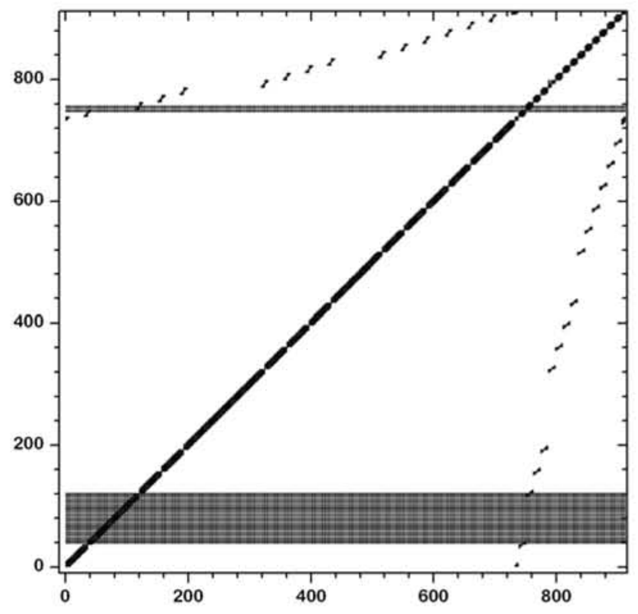

Fig. 4.23 : Jacobian matrix pattern for solution with 304 elements, cRDG$_0$, and BDF$_2$ schemes (at steady-state), when Doppler feedback is accounted for.

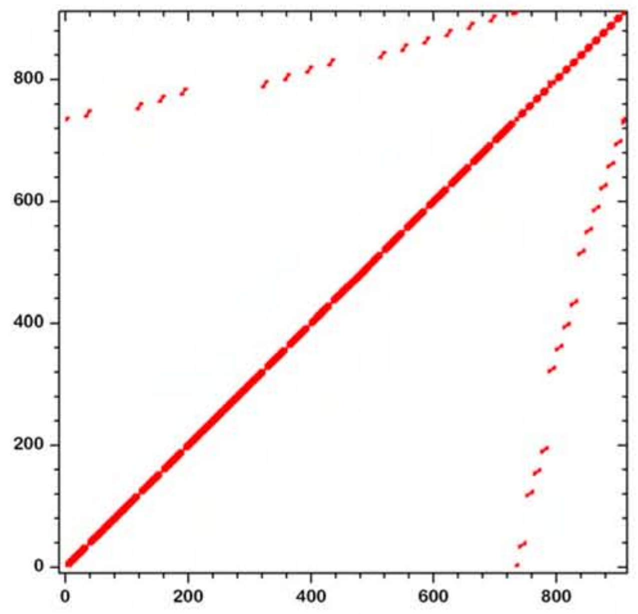

Fig. 4.24 : Jacobian matrix pattern for solution with 304 elements, cRDG$_0$, and BDF$_2$ schemes (at steady-state), when Doppler feedback is ignored.

transient. Pressure, density, velocity and temperature profiles for different times of the transient are shown in Figures 4.19, 4.20, 4.21 and 4.22, respectively.

It is instructive to note the important effect of the Doppler feedback on the structure of the Jacobian matrix, comparing Figures 4.23 and 4.24. The temperature coupling in the core causes a significant number of rows to be fully filled-in. This effect is efficiently accounted for by re-probing the Jacobian matrix during the transient.

4.2.3 Pressurizer failure transient

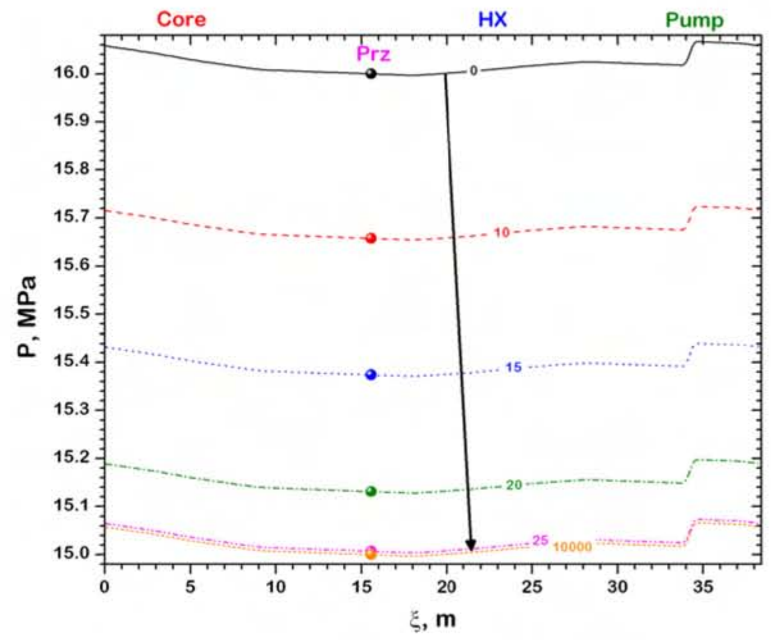

Fig. 4.25 : Pressure distribution for the pressurizer failure transient. Solution with 304 elements, cRDG$_0$, and BDF$_2$ schemes.

In the next transient, we "fail" the pressurizer, causing the loop pressure to drop from 16 MPa to 15 MPa during the first 26 sec. Both the pump and core remain at full power.

The response of the reactor's flow parameters is shown in Figures 4.25, 4.26, 4.27 and 4.28 – for pressure, density, velocity and temperature, correspondingly.

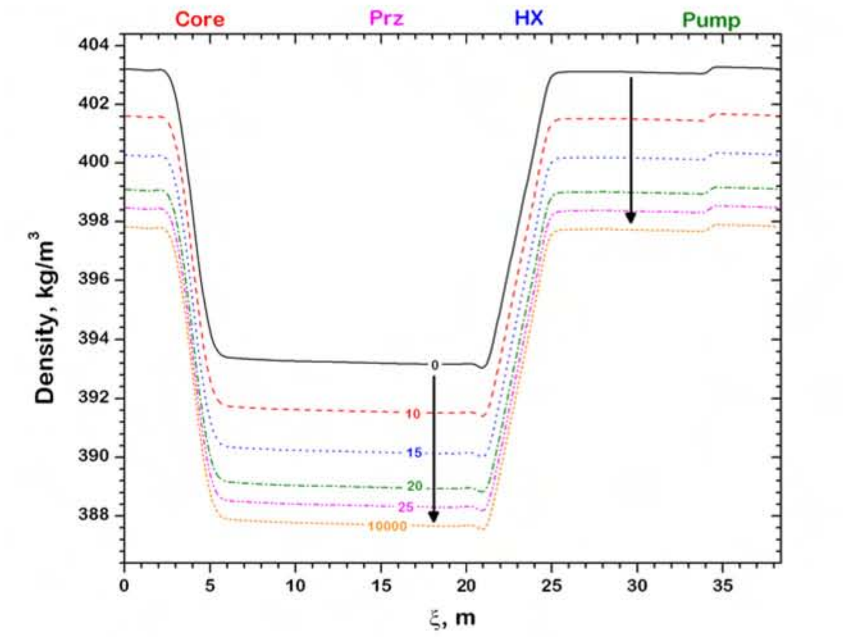

Fig. 4.26 : Density distribution for the pressurizer failure transient. Solution with 304 elements, cRDG$_0$, and BDF$_2$ schemes.

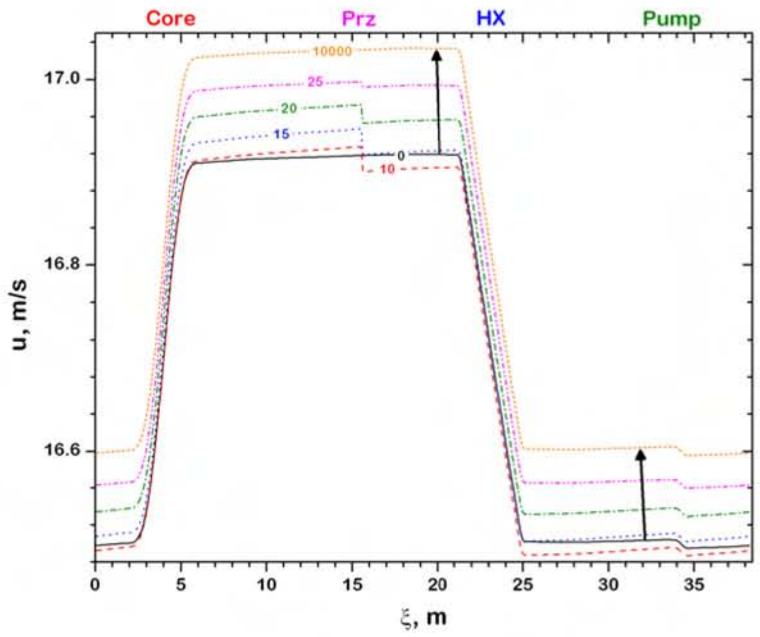

Fig. 4.27 : Velocity distribution for the pressurizer failure transient. Solution with 304 elements, cRDG$_0$, and BDF$_2$ schemes.

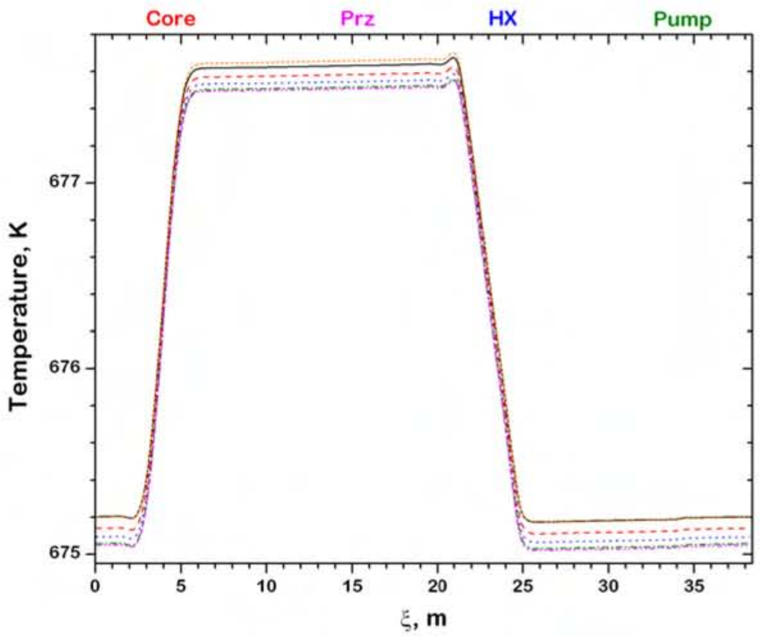

Fig. 4.28 : Temperature distribution for the pressurizer failure transient. Solution with 304 elements, cRDG$_0$, and BDF$_2$ schemes.

4.3 Thermohydraulic Loop with a I&C System

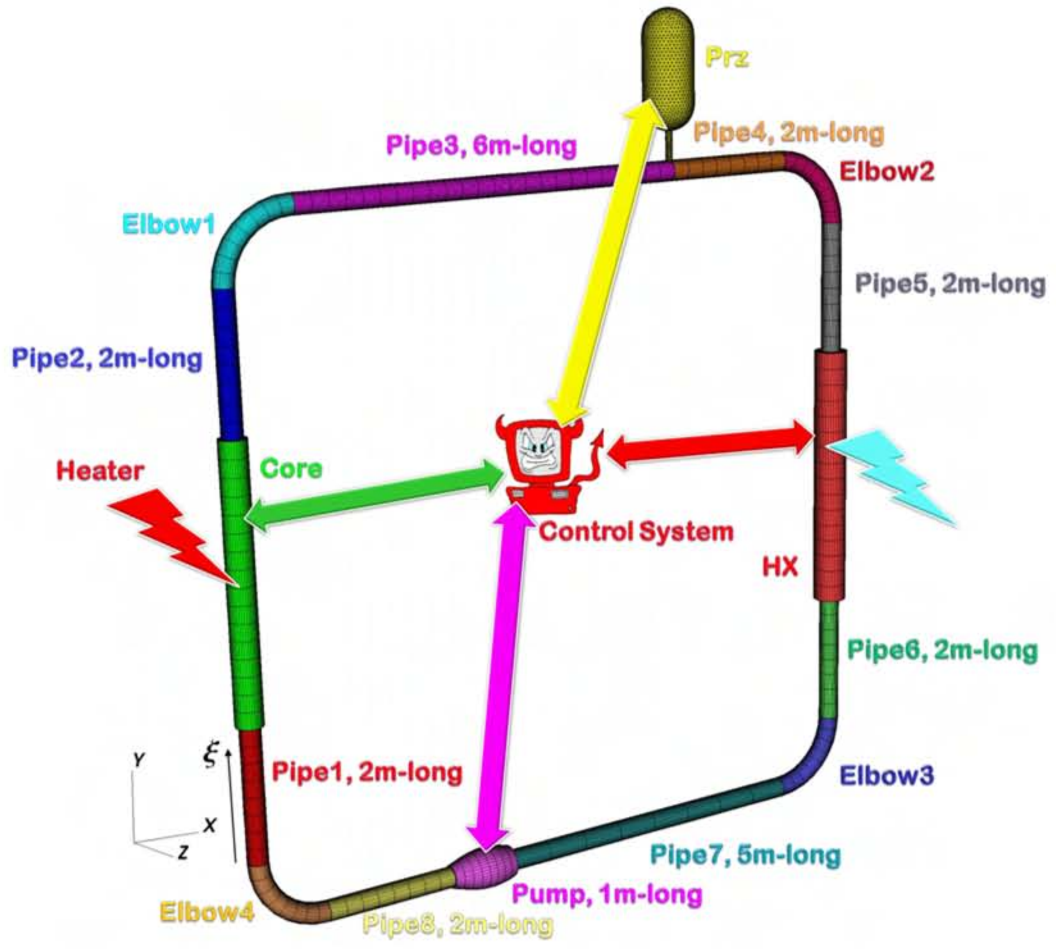

Fig. 4.29 : Formulation of the "VR_2" reactor problem.

"VR_2" is an "upgrade" of "VR_1", with the following modifications:

1. Nominal power of the heater is increased to 6 MW. In the following exercise, we will ignore Doppler feedback effects.
2. Wall temperature in heat exchanger is reduced to 530 K.
3. Both Core and HX induce additional friction due to some internal flow obstructions, effectively represented by hydraulic diameter set to 5 cm,

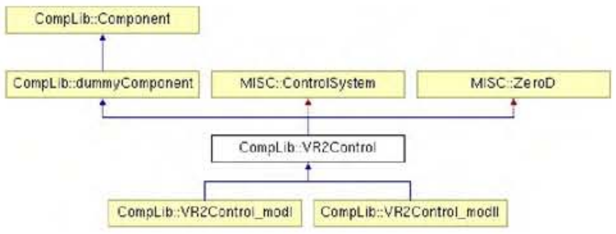

Fig. 4.30 : Inheritance tree for Class `VR2Control`.

smoothly transitioned to the effective diameter of 15 cm, using sine smoothing functions with $\delta_{\text{smoothing}} = 0.1$ at the entrance and exit of the Core and HX.

4. Nominal pump power is increased to $\Delta P_{\text{Pump,nominal}} = 2 \cdot 10^5$ Pa.

5. Control system is added (Figure 4.30), with a Instrumentation and Control (I&C) system, including "sensors" and "actuators" connected to the pump, pressurizer, core and HX, see Figure 4.29.

4.3.1 Pump failure with SCRAM

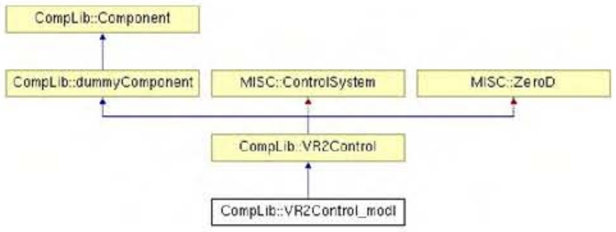

Fig. 4.31 : Inheritance tree for Class `VR2Control_modI`.

In the first example with "VR$_2$" (**`VR2Control_modI`**), we will analyze a scenario of pump failure with possible subsequent full or partial SCRAM. We start with the reactor at the nominal steady-state, and 4 sec after we initiate a *pump failure*. This will be followed by the insertion of control rods. This scenario is controlled by the component `VR2Control_modI`, Figure 4.31.

There are four variable parameters used in the present study:

- τ_{Pump}: Relaxation time of pump's phase-out, varied from 10 to 100 sec. Pump is phased out from the full power $\mathcal{P}_{\text{Pump}} = 1$ to $\mathcal{P}_{\text{Pump}} = 0$ within τ_{Pump}.
- τ_{delay}: Delay time, between pump's trip and starting SCRAM, varied from 1 to 100 sec.
- CRP_{end}: Control rod position at the end of SCRAM, varied from 0.05 to 0.55.
- τ_{CRP}: Relaxation time of the control rod system. CRs are inserted from the nominal position (set to 0.5) to CRP_{end} within τ_{CRP} sec. varied from 10 to 50 sec.

The figure of merit (FOM) for RISMC analysis of VR_2 is the *"Peak Coolant Temperature" (PCT)*.

Inputs for components of type `VR2Control_modI` are specified in

"`INPUT/DefineComponents/ComponentName.inp`"

files, where "`ComponentName`" is a unique name of the control system component, as defined in the file "`INPUT/ListOfComponents.inp`". To declare the component "`ComponentName`" to be of the type `VR2Control_modI`, one must set

```
[Define]
  type = VR2Control_modI
[]
```

The pump is controlled using the following input parameters:

```
[PumpHistory]
  event_start =→ Start of the pump failure
  event_duration = $\tau_{\text{Pump}}$
  pow_end = $\mathcal{P}_{\text{Pump}}$
  resolve =→ During the event, the maximum time step $\Delta t$ is set to $\frac{\tau_{\text{Pump}}}{\texttt{resolve}}$
[]
```

The control system is controlled using the following input parameters:

```
[CRPHistory]
  pump_End_act =→ $\mathcal{P}_{\text{Pump}}$ causing the SCRAM
  event_delay = $\tau_{\text{delay}}$
  event_duration = $\tau_{\text{CRP}}$
  CRP_End = $\text{CRP}_{\text{end}}$
  resolve =→ During the event, the maximum time step $\Delta t$ is set to $\frac{\tau_{\text{CRP}}}{\texttt{resolve}}$
[]
```

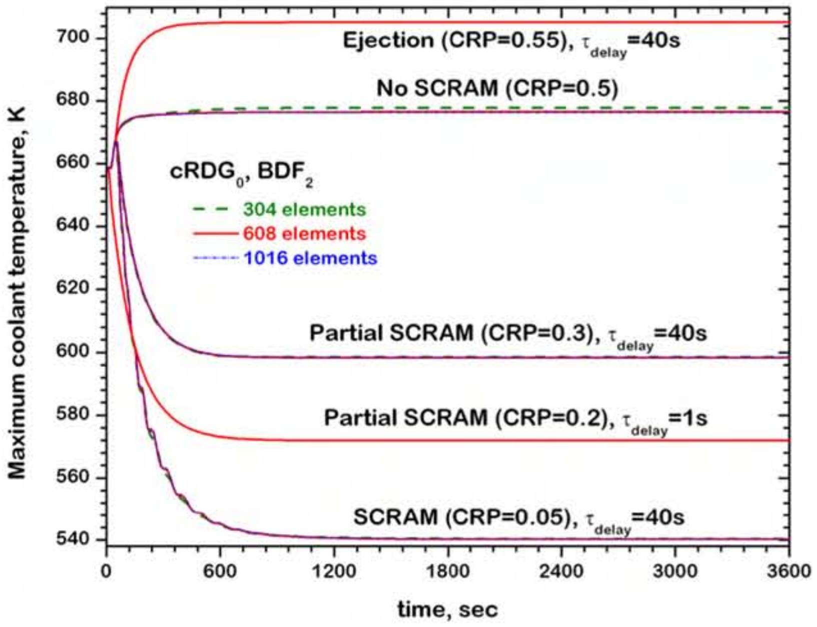

Fig. 4.32 : Dynamics of the maximum coolant temperature for selected transients. Full phase-out of pump within 26 sec, and SCRAM applied within 20 sec, started τ_{delay} sec after start of the pump trip.

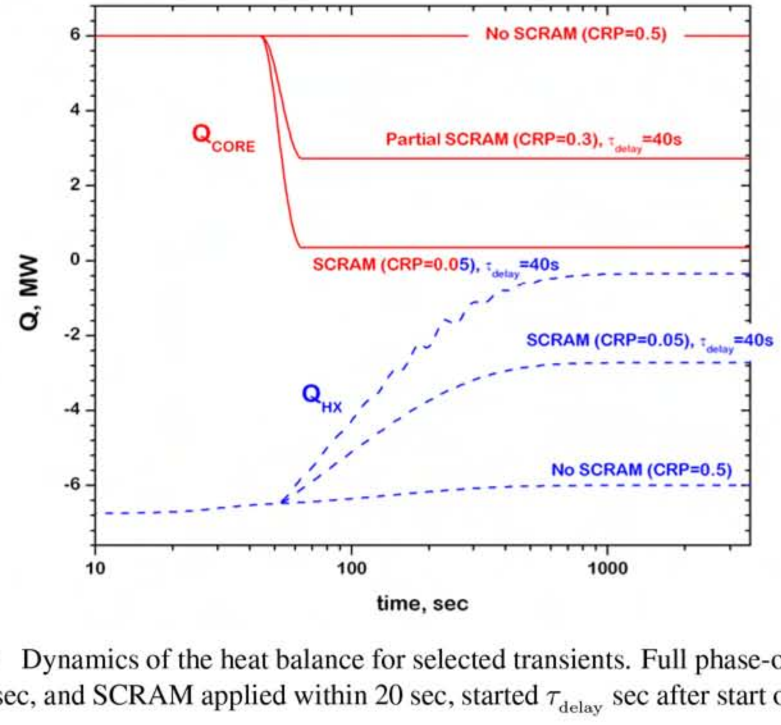

Fig. 4.33 : Dynamics of the heat balance for selected transients. Full phase-out of pump within 26 sec, and SCRAM applied within 20 sec, started τ_{delay} sec after start of the pump trip.

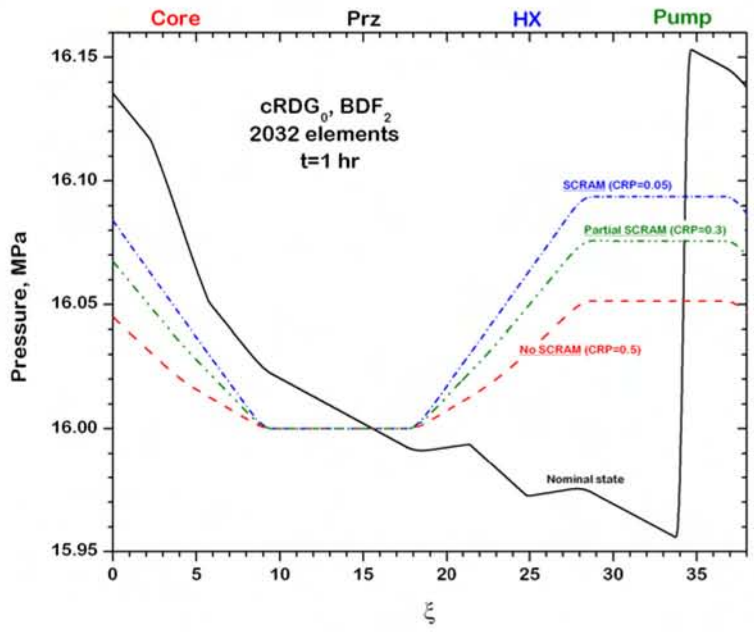

Fig. 4.34 : Pressure distributions 1 hr after the starts of selected transients. Solutions with 2032 elements, cRDG$_0$, and BDF$_2$ schemes.

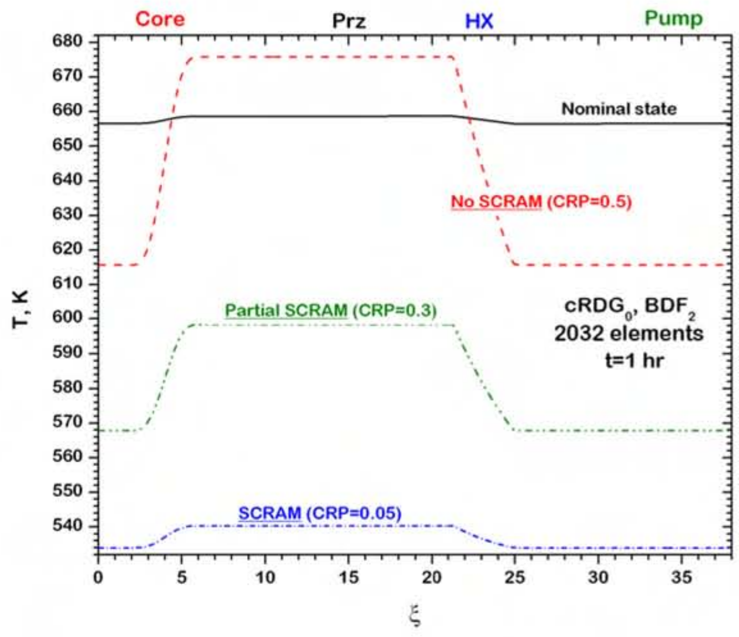

Fig. 4.35 : Temperature distributions 1 hr after the starts of selected transients. Solutions with 2032 elements, cRDG$_0$, and BDF$_2$ schemes.

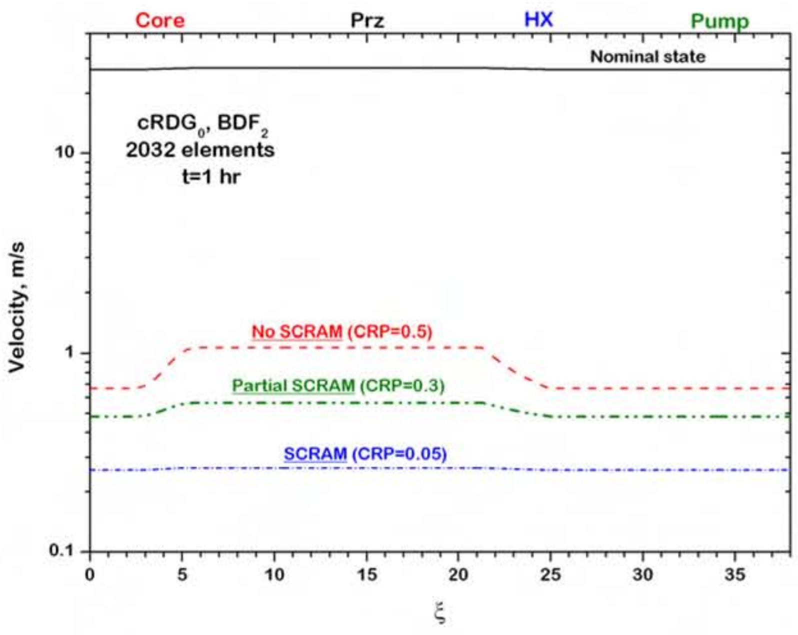

Fig. 4.36 : Velocity distributions 1 hr after the starts of selected transients. Solutions with 2032 elements, cRDG$_0$, and BDF$_2$ schemes.

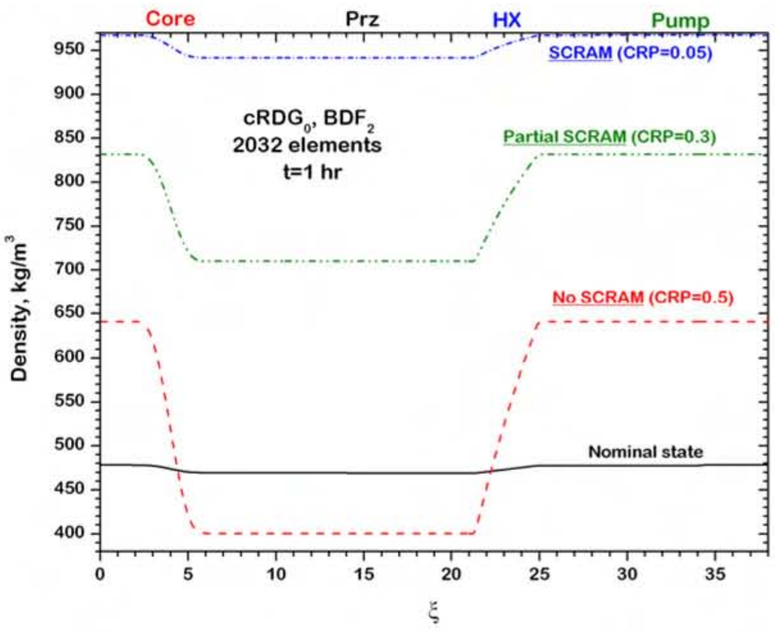

Fig. 4.37 : Density distributions 1 hr after the starts of selected transients. Solutions with 2032 elements, cRDG$_0$, and BDF$_2$ schemes.

Figures 4.32-4.37 show results from the selected parameter set. Figures 4.32 and 4.33 demonstrate the history of PCT and energy balance in the system. Spatial distributions of pressure, temperature, velocity and density are presented in Figures 4.34, 4.35, 4.36 and 4.37, respectively.

To compute RISMC "logo", we utilize the SNL's DAKOTA software. Each of RISMC β-2 runs was set for the maximum $t = 600$ s of accident transient time[2], which executed in parallel within 1.5-2 min on each CPU of 2×4-core 2.8GHz Intel Xeon Linux desktop. The total of 10,000 MC runs took ≈ 40 clock hours.

Each of four uncertain parameters are sampled from the corresponding range, using equal probability. Figure 4.38 shows PDF and CDF for PCT, using 100, 1,000, 10,000 and 100,000 random samples.

4.3.2 Pressurizer failure transient

In the second example with "VR$_2$" (**`VR2Control_modII`**), we analyze a scenario with pressurizer failure, followed by pump trip and initiation of SCRAM. This scenario is controlled by the component `VR2Control_modII`, Figure 4.39.

There are seven variable parameters used in the present study:

- $P^{\text{end}}_{\text{Prz}}$: final pressurizer's pressure at the end of transient, varied from 15 to 15.5 MPa.

- τ_{Prz}: Time (duration) of the pressurizer failure transient, varied from 10 to 50 sec.

- $P^{\text{Trip}}_{\text{Pump}}$: Minimum pressure in the pump, causing the trip, varied from 15.6 to 15.7 MPa.

- τ_{Pump}: Relaxation time of pump's phase-out, varied from 10 to 100 sec. Pump is phased out from the full power $\mathcal{P}_{\text{Pump}} = 1$ to $\mathcal{P}_{\text{Pump}} = 0$ within τ_{Pump}.

[2]As one can see from Figure 4.32, the PCT occurs at the very beginning of the transient.

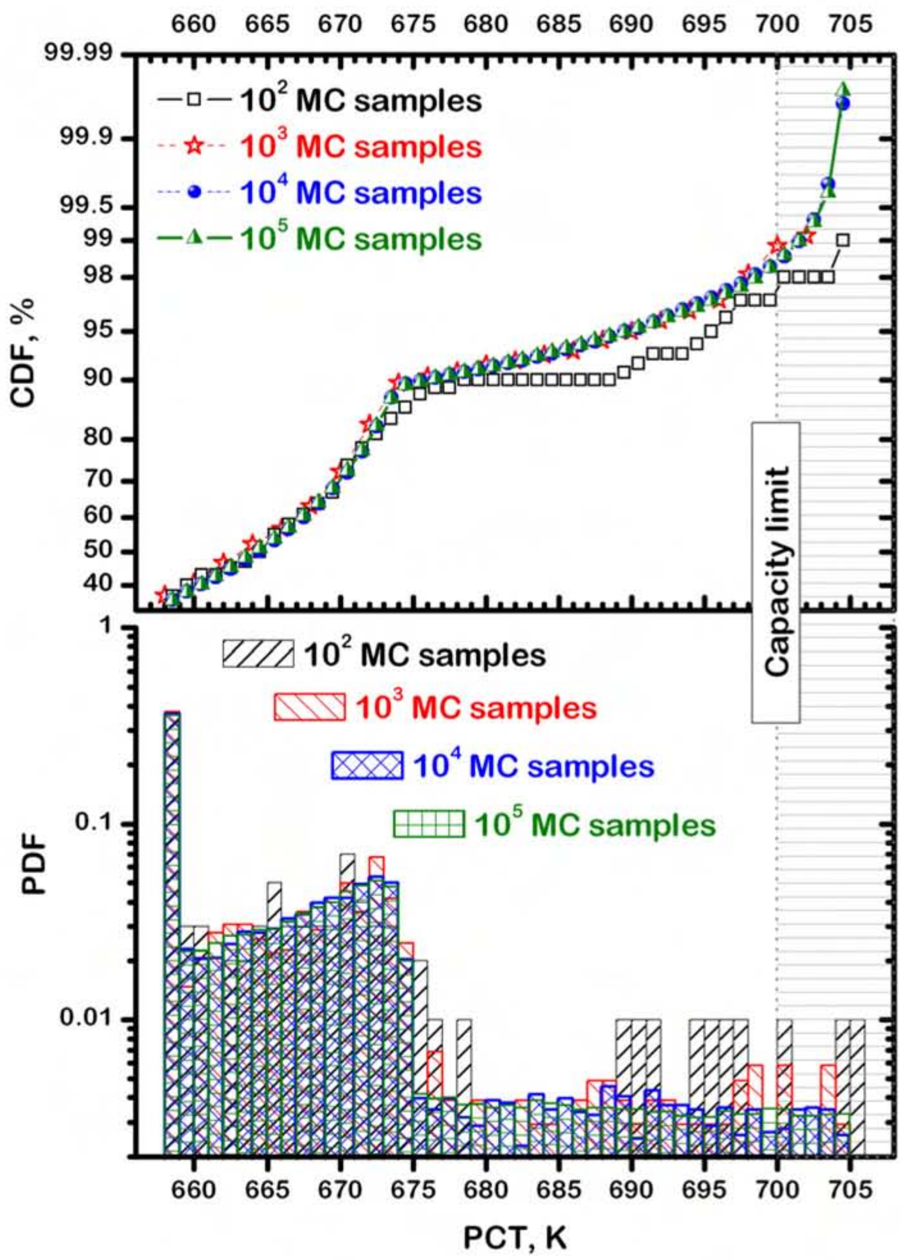

Fig. 4.38 : RISMC "capacity-loading" plot for "pump failure with SCRAM", generated using Monte-Carlo sampling.

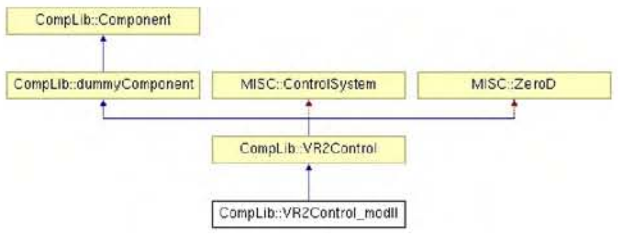

Fig. 4.39 : Inheritance tree for Class `VR2Control_modII`.

- $T_{\max}^{\mathrm{SCRAM}}$: Maximum temperature in the system causing the SCRAM, varied from 665 to 675 K.
- $\mathrm{CRP}_{\mathrm{end}}$: Control rod position at the end of SCRAM, varied from 0.05 to 0.55.
- τ_{CRP}: Relaxation time of the control rod system. CRs are inserted from the nominal position (set to 0.5) to $\mathrm{CRP}_{\mathrm{end}}$ within τ_{CRP} sec. varied from 10 to 50 sec.

The figure of merit (FOM) is the *"Peak Coolant Temperature" (PCT).*

Inputs for components of type `VR2Control_modII` are specified in

"`INPUT/DefineComponents/ComponentName.inp`"

files, where "`ComponentName`" is a unique name of the control system component, as defined in the file "`INPUT/ListOfComponents.inp`". To declare the component "`ComponentName`" to be of the type `VR2Control_modII`, one must set

```
[Define]
  type = VR2Control_modII
[]
```

The pressurizer is controlled using the following input parameters:

```
[PrzHistory]
  event_start     =  → Start of the pressurizer failure
  event_duration  =  → τ_Prz
  pre_End         =  → P_Prz^end
  resolve         =  → During the event, the maximum time step Δt is set to τ_Prz / resolve
[]
```

The pump is controlled using the following input parameters:

```
[PumpControl]
  Trip_pressure   =  → $P^{\rm Trip}_{\rm Pump}$
  Pump_relax      =  → $\tau_{\rm Pump}$
  pow_end         =  → $\mathcal{P}_{\rm Pump}$
  resolve         =  → During the event, the maximum time step $\Delta t$ is set to $\frac{\tau_{\rm Pump}}{\tt resolve}$
[]
```

The control system is controlled using the following input parameters:

```
[SCRAM]
  SCRAM_temperature   =  → $T^{\rm SCRAM}_{\rm max}$
  CRP_relax           =  → $\tau_{\rm CRP}$
  CRP_End             =  → ${\rm CRP}_{\rm end}$
  resolve             =  → During the event, the maximum time step $\Delta t$ is set to $\frac{\tau_{\rm CRP}}{\tt resolve}$
[]
```

Each of RISMC β-2 runs was set for the maximum $t = 1,200$ s of accident transient time. The total of 10,000 MC runs took ≈ 1.7 clock hours on 300 CPUs of INL's **IceStorm** supercomputer. Each of seven uncertain parameters are sampled from the corresponding range, using equal probability. Figure 4.40 shows PDF and CDF for PCT, using 100, 1,000, 10,000 and 100,000 random sampling.

4.3.3 High-Performance Parallel Computation

In the next example (**VR2Control_modII** with 20 uncertainty parameters), in addition to seven scenario-uncertainty parameters of Section 4.3.2, we add 1 more scenario-, 11 modeling- and 1 discretization-uncertainty parameters:

---

1. $P^{\rm end}_{\rm Prz}$: final pressurizer's pressure at the end of transient, varied from 15 to 15.5 MPa.

2. $\tau_{\rm Prz}$: Time (duration) of the pressurizer failure transient, varied from 10 to 50 sec.

3. $P^{\rm Trip}_{\rm Pump}$: Minimum pressure in the pump, causing the trip, varied from 15.6 to 15.7 MPa.

4. $\tau_{\rm Pump}$: Relaxation time of pump's phase-out, varied from 10 to 100 sec. Pump is phased out from the full power $\mathcal{P}_{\rm Pump} = 1$ to $\mathcal{P}_{\rm Pump} = 0$ within $\tau_{\rm Pump}$.

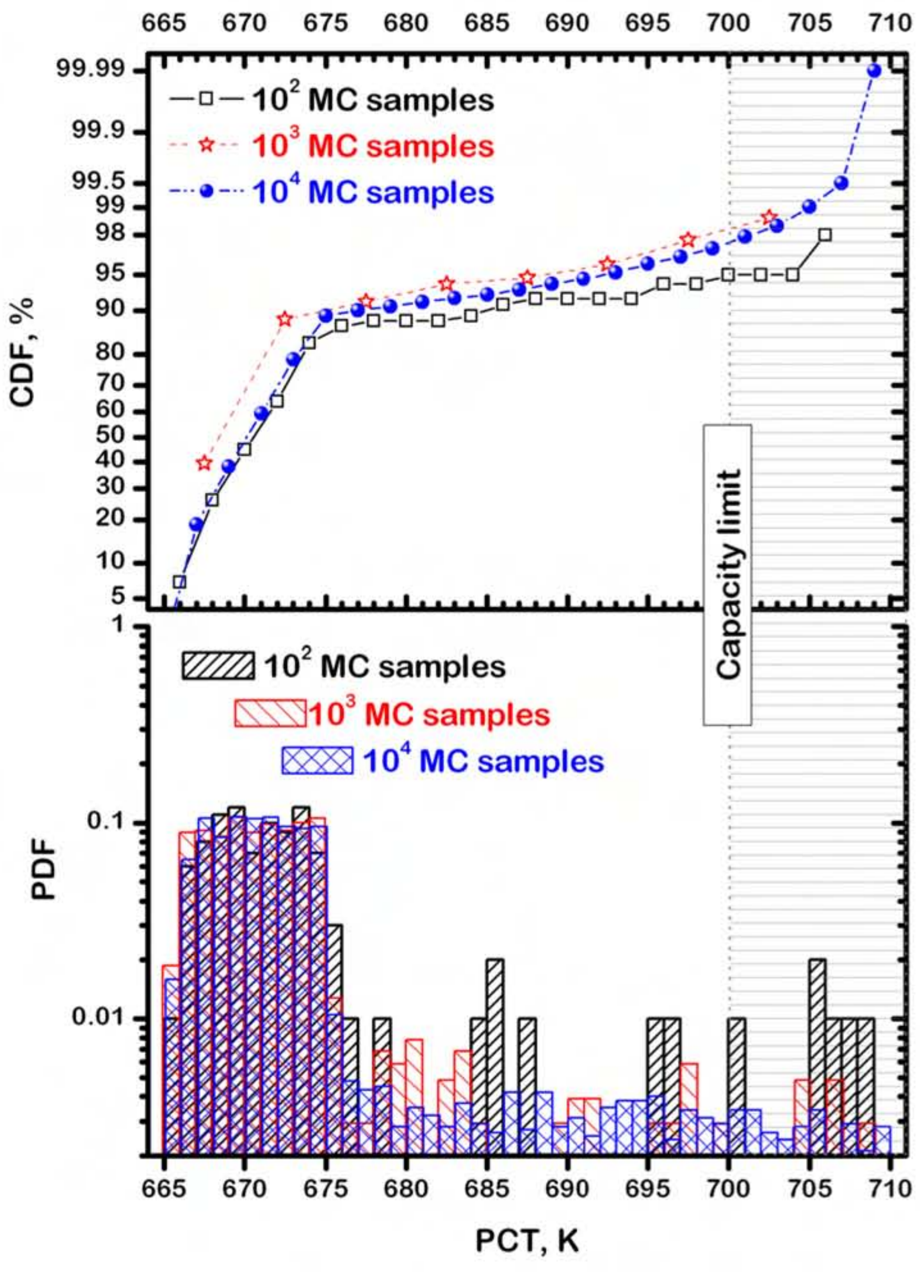

Fig. 4.40 : RISMC "capacity-loading" plot for "pressurizer failure scenario", generated using Monte-Carlo sampling.

5. $T_{\text{max}}^{\text{SCRAM}}$: Maximum temperature in the system causing the SCRAM, varied from 665 to 675 K.

6. CRP_{end}: Control rod position at the end of SCRAM, varied from 0.05 to 0.55.

7. τ_{CRP}: Relaxation time of the control rod system. CRs are inserted from the nominal position (set to 0.5) to CRP_{end} within τ_{CRP} sec varied from 10 to 50 sec.

8. End power of the pump, varied from 0 to 0.1.

9. Parameter α in LLF scheme (see Section 3.3.1), varied from 0.9 to 1.

10. Hydraulic radius in the core, varied from 0.05 to 0.06 m.

11. Hydraulic radius in the HX, varied from 0.05 to 0.06 m.

12. Wall temperature in the HX, varied from 528 to 532 K.

13. Nominal power of the core, varied from 5.98 to 6.02 MW.

14. Nominal pump head, varied from 1.98 to 2.02 Bar.

15. Uncertainty in turbulent friction law, varied from -$\frac{1}{2}$ to $\frac{1}{2}$ %.

16. Uncertainty in HX' Nusselt number, varied from -1 to 1 %.

17. Uncertainty in specific heat, varied from 3.05613 to 3.05813 $\frac{kJ}{kg\ K}$.

18. Uncertainty in dynamic viscosity, varied from 70.14547 to 70.16547 $\frac{kg}{m\ s}$.

19. Uncertainty in $\left.\frac{\partial P}{\partial \rho}\right|_{(\rho_0,T_0)}$, varied from 0.1851 to 0.1853 $\frac{MPa}{kg} m^3$.

20. Uncertainty in $\left.\frac{\partial P}{\partial T}\right|_{(\rho_0,T_0)}$, varied from 0.7389 to 0.7391 $\frac{MPa}{K}$.

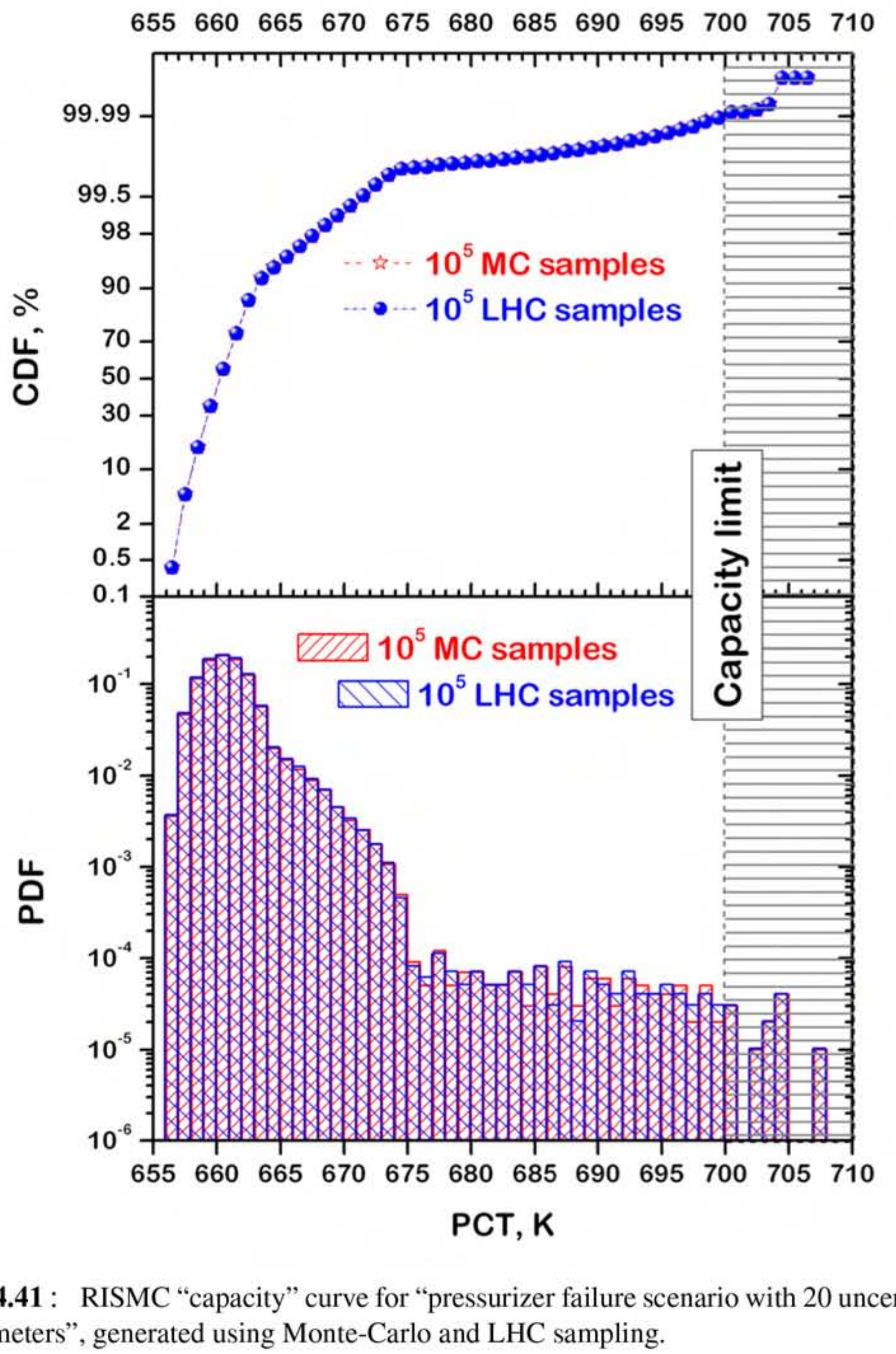

Fig. 4.41 : RISMC "capacity" curve for "pressurizer failure scenario with 20 uncertainty parameters", generated using Monte-Carlo and LHC sampling.

Monte-Carlo Analysis

Each of twenty uncertain parameters are sampled from the corresponding range, using equal probability. Computational results for Monte-Carlo (MC) and Latin Hypercube Sampling (LHS) with 10^5 samples are shown in Figure 4.41. The total of 10^5 code runs were executed successfully (without single crash) within ≈ 16 clock hours using 512 CPUs on **`IceStorm`** [3].

[3] **`IceStorm`** specs:

- SGI Altix ICE 8200 distributed memory blade cluster.
- 256 compute blades with two quad core Intel Xeon processors each.
- 2,048 compute cores total, 2.66 GHz clock speed.
- 2 login nodes, each with 8 cores.
- 2 GB memory per core, 4 TB memory total.
- DDR 4X InfiniBand interconnect network.
- Operating System: SUSE Linux Enterprise Server 10.
- 70 TB disk capacity.

4.4 Loading Calculation and Smart Sampling

The VR_2 reactor is "upgraded" to allow for testing methods of sensitivity analysis and uncertainty quantification. In particular, two techniques of uncertainty propagation methods are applied, which address the estimation of tail probabilities in a more efficient way than a pure sampling strategy does.

Analytic reliability methods operate by transforming the uncertainty quantification problem to an optimization one, where the goal is to find the "most probable point" of failure (MPP). In a standardized normal space, one can think of the most probable point as the distance between the origin and the failure region. This distance is called the reliability index, beta, and represents how far the failure surface is from nominal or mean behavior. For example, a beta value of 2 represents a failure region that is 2 standard deviations from mean conditions, and a beta value of 3 (where failure is three standard deviations away from nominal) is a more reliable design. Analytic reliability methods do require gradient calculations (gradients of the response with respect to the uncertain variables) but they tend to be extremely efficient at finding response levels according to percentiles of output distributions, especially if the response is not too nonlinear.

Stochastic expansion methods develop an approximation of a random response function in terms of finite-dimensional series expansions. We will examine two classes of expansion methods: *polynomial chaos expansion (PCE)* and *stochastic collocation (SC)*.

In PCE, the output is represented as a sum of orthogonal polynomial basis functions which are chosen based on the distribution type of the random inputs (e.g. normal inputs use Hermite polynomials, uniforms use Legendre polynomials, etc.) The uncertainty propagation problem becomes one of obtaining the coefficients of the polynomials in the expansion. There are a variety of methods to obtain the coefficients, including multidimensional integration based on tensor-product quadrature or sparse grids, multidimensional integration via random sampling, or linear regression.

SC is very similar to PCE except that the polynomials are replaced by Lagrange polynomial interpolants. Once the expansions have been developed (e.g. the coefficients calculated), one has an analytic form of the stochastic input-output mapping, and one can sample that mapping extensively to build up a CDF of the output (for example) and obtain tail probability estimates. Research has shown that CDF estimates using stochastic expansion can converge much faster than sampling [cite Eldred].

These methods will be implemented using the DAKOTA software framework, which has several algorithmic variations implemented for both reliability and stochastic expansion methods.

Two VR_2 cases are considered, with six and twenty one variable parameters, randomly selected from uniform distributions.

Six-parameter case

1. $P^{\mathrm{Trip}}_{\mathrm{Pump}}$: Minimum pressure in the pump, causing the trip, varied from 15.6 to 15.7 MPa.
2. τ_{Pump}: Relaxation time of pump's phase-out, varied from 10 to 100 sec. Pump is phased out from the full power $\mathcal{P}_{\mathrm{Pump}} = 1$ to $\mathcal{P}_{\mathrm{Pump}} = 0$ within τ_{Pump}.
3. End power of the pump, varied from 0 to 0.4.
4. $T^{\mathrm{SCRAM}}_{\mathrm{max}}$: Maximum temperature in the system causing the SCRAM, varied from 625 to 635 K.
5. $\mathrm{CRP}_{\mathrm{end}}$: Control rod position at the end of SCRAM, varied from 0.025 to 0.24.
6. τ_{CRP}: Relaxation time of the control rod system. CRs are inserted from the nominal position (set to 0.5) to $\mathrm{CRP}_{\mathrm{end}}$ within τ_{CRP} sec. varied from 10 to 50 sec.

Twenty-one-parameter case

1. $P^{\mathrm{end}}_{\mathrm{Prz}}$: final pressurizer's pressure at the end of transient, varied from 15 to 15.5 MPa.
2. τ_{Prz}: Time (duration) of the pressurizer failure transient, varied from 10 to 50 sec.

3. $P_{\text{Pump}}^{\text{Trip}}$: Minimum pressure in the pump, causing the trip, varied from 15.6 to 15.7 MPa.
4. τ_{Pump}: Relaxation time of pump's phase-out, varied from 10 to 100 sec. Pump is phased out from the full power $\mathcal{P}_{\text{Pump}} = 1$ to $\mathcal{P}_{\text{Pump}} = 0$ within τ_{Pump}.
5. End power of the pump, varied from 0 to 0.4.
6. $T_{\text{max}}^{\text{SCRAM}}$: Maximum temperature in the system causing the SCRAM, varied from 625 to 635 K.
7. CRP_{end}: Control rod position at the end of SCRAM, varied from 0.025 to 0.24.
8. τ_{CRP}: Relaxation time of the control rod system. CRs are inserted from the nominal position (set to 0.5) to CRP_{end} within τ_{CRP} sec. varied from 10 to 50 sec.
9. Parameter α in LLF scheme (see Section 3.3.1), varied from 0.9 to 1.
10. Hydraulic radius in the core, varied from 0.05 to 0.06 m.
11. Hydraulic radius in the HX, varied from 0.05 to 0.06 m.
12. Wall temperature in the HX, varied from 510 to 515 K.
13. Nominal power of the core, varied from 14 to 16 MW.
14. Nominal pump head, varied from 1.98 to 2.02 Bar.
15. Uncertainty in turbulent friction law, varied from -10 to 10 %.
16. Uncertainty in HX' Nusselt number, varied from -15 to 15 %.
17. Uncertainty in specific heat, varied from 3.05513 to 3.05513 $\frac{kJ}{kg\ K}$.
18. Uncertainty in dynamic viscosity, varied from 71.4547 to 71.6547 $\frac{kg}{m\ s}$.
19. Uncertainty in thermal conductivity, varied from 0.4476655 to 0.4876655 $\frac{W}{m\ K}$.
20. Uncertainty in $\left.\frac{\partial P}{\partial \rho}\right|_{(\rho_0,T_0)}$, varied from 0.185 to 0.1854 $\frac{MPa}{kg}m^3$.
21. Uncertainty in $\left.\frac{\partial P}{\partial T}\right|_{(\rho_0,T_0)}$, varied from 0.7388 to 0.7392 $\frac{MPa}{K}$.

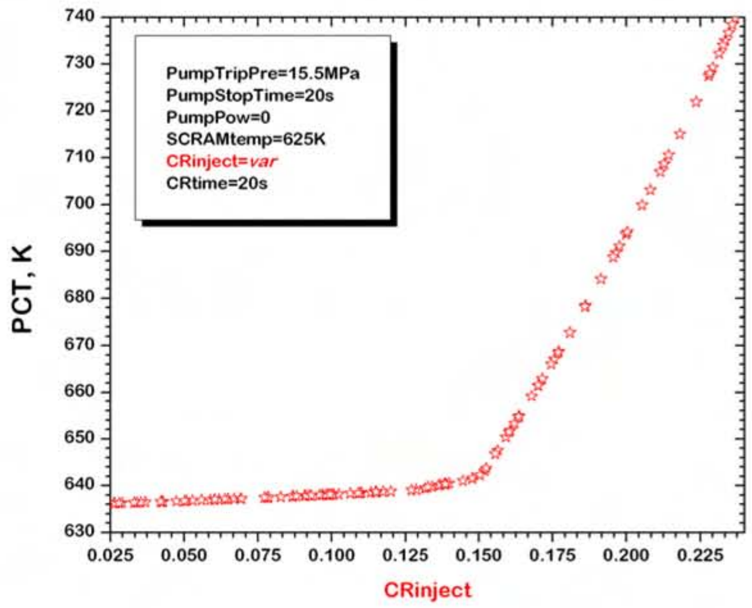

Fig. 4.42 : Sensitivity to the end control rod position, CRP_{end} (6-parameter case).

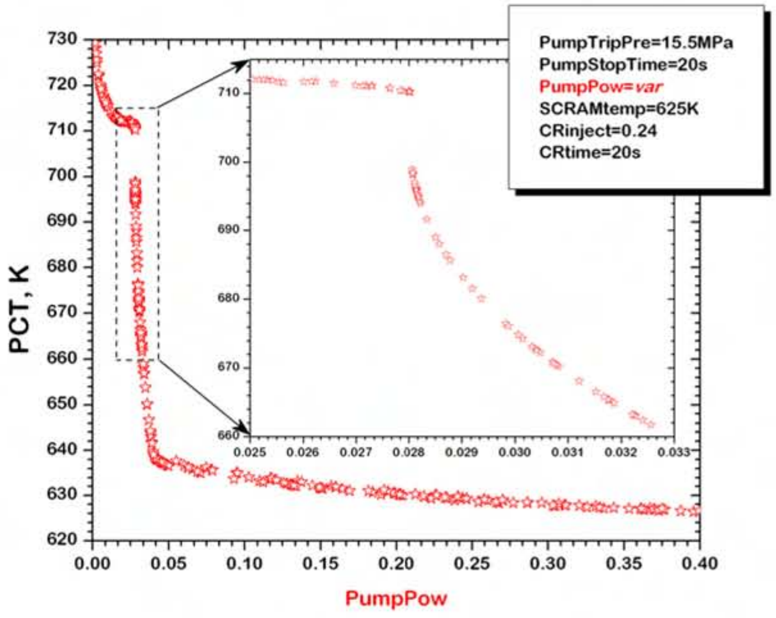

Fig. 4.43 : Sensitivity to the end pump power (6-parameter case).

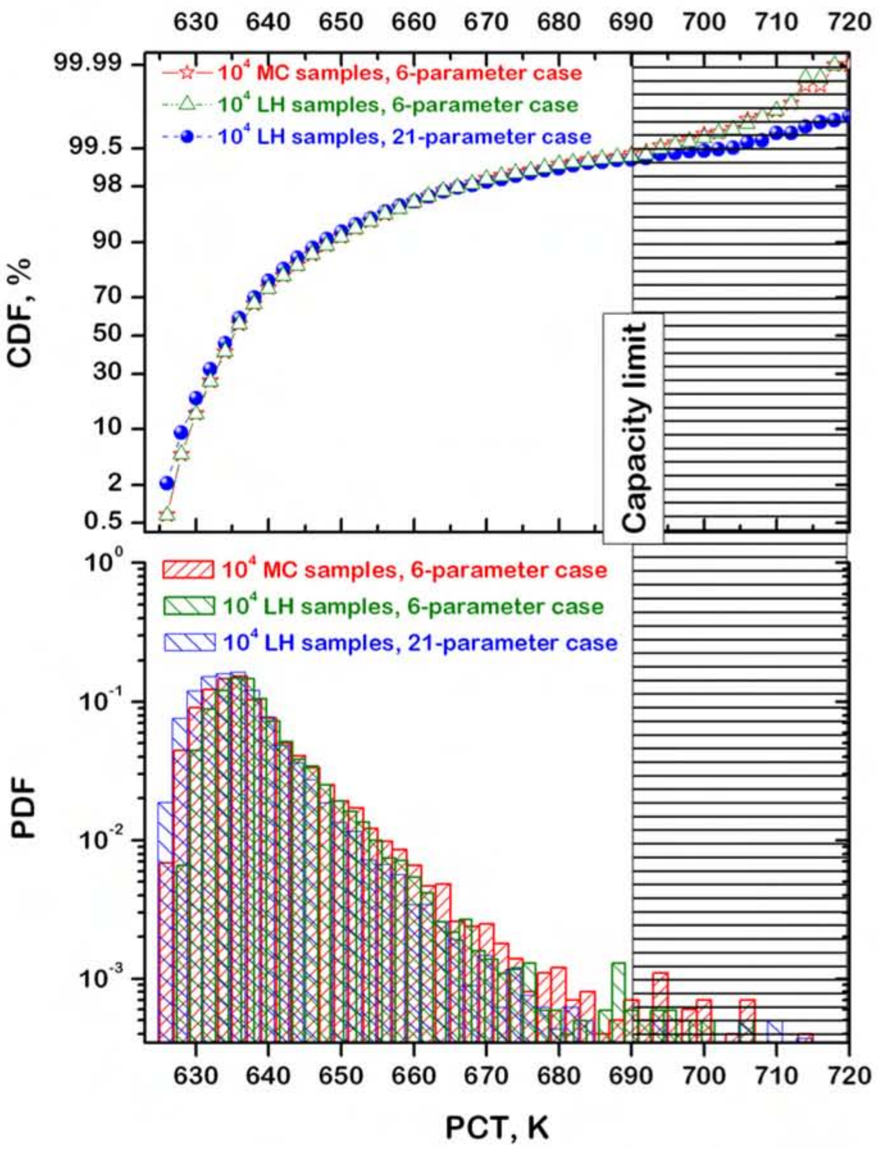

Fig. 4.44 : RISMC "capacity" curve for "pressurizer failure scenario with 6 uncertainty parameters", generated using Monte-Carlo sampling.

Figures 4.42 and 4.43 demonstrate sensitivity of the PCT to the end control rod position, $\mathrm{CRP}_{\mathrm{end}}$, and to the end pump power. It can be seen that the sensitivity plots are discontinuous.

Capacity curve generated with 10^4 Monte-Carlo runs is shown in Figure 4.44.

Figures 4.45 and 4.46 compare LHC sampling and PCE approach. It can be seen that for a smaller sample, PCE converges faster than LHC. The two methods produce similar results.

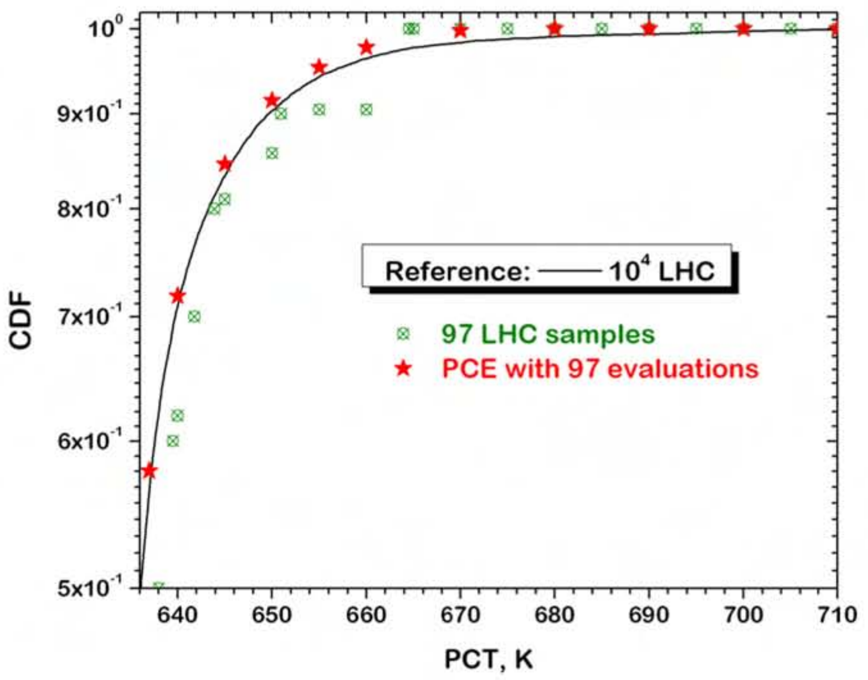

Fig. 4.45 : CDF for "pressurizer failure scenario with 6 uncertainty parameters", generated using 97 samples. LHC sampling versus polynomial chaos expansion (PCE).

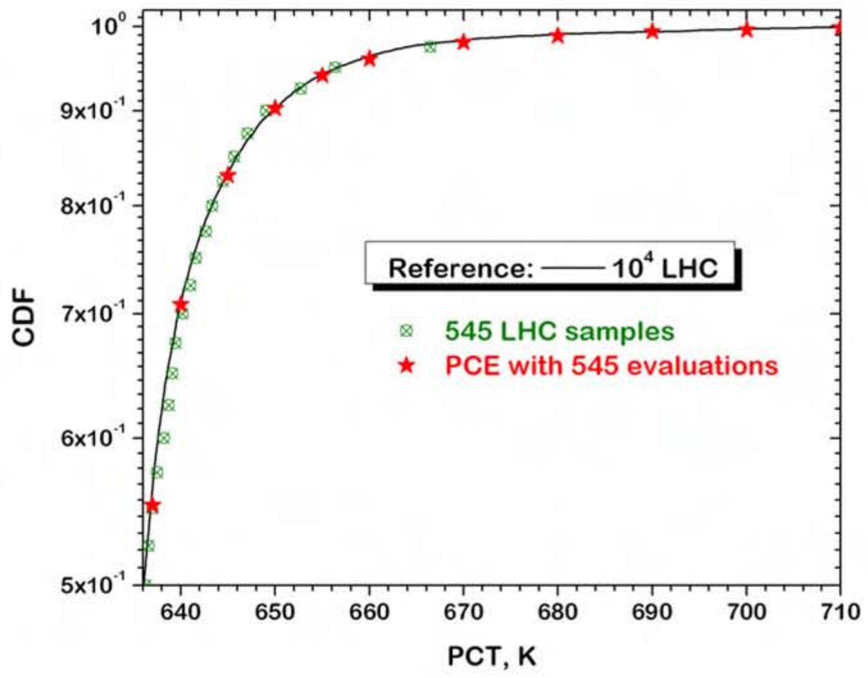

Fig. 4.46 : CDF for "pressurizer failure scenario with 6 uncertainty parameters", generated using 545 samples. LHC sampling versus polynomial chaos expansion (PCE).

4.5 3D Loop Configuration

This example is related to a more complex configuration modeled with the code β_2 version. A PWR-type primary coolant system is rendered in 3D (Figure 4.47), whereas the reactor vessel model includes a core (heated channel), a downcomer, a metal vessel, and a control rod (Figure 4.48).

A steady-state solution is shown in Figures 4.49-4.50, and space convergence in Figure 4.51. A model for core neutronics, vessel wall fluence and thermo-mechanics, vessel material degradation will be implemented and used to characterize the aging of the reactor pressure vessel. The model will be extended for study safety margins in "feed-and-bleed" scenarios including "aging-accentuated feed-and-bleed" scenarios.

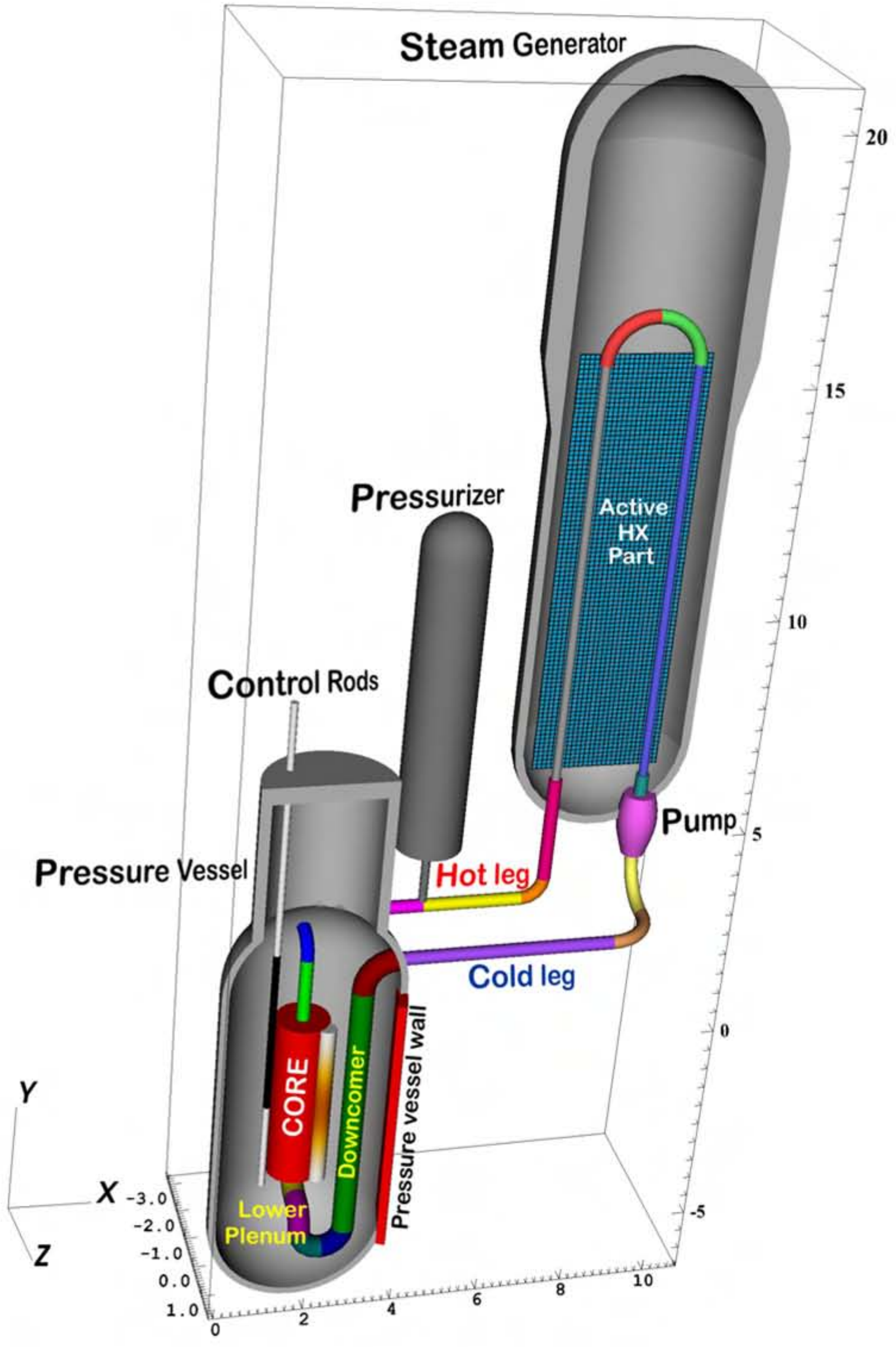

Fig. 4.47 : Configuration setup for PWR_{60}: overall look.

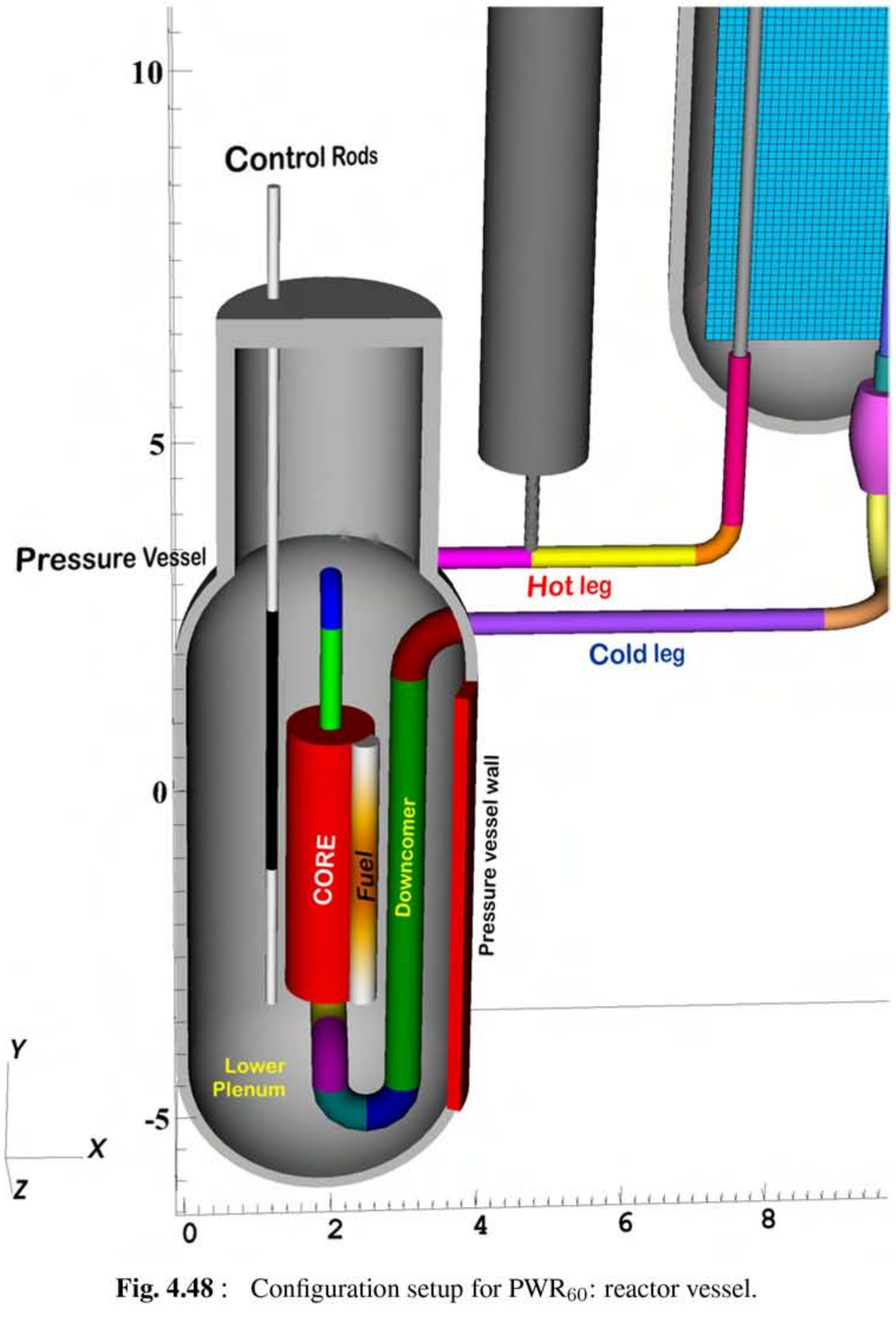

Fig. 4.48 : Configuration setup for PWR_{60}: reactor vessel.

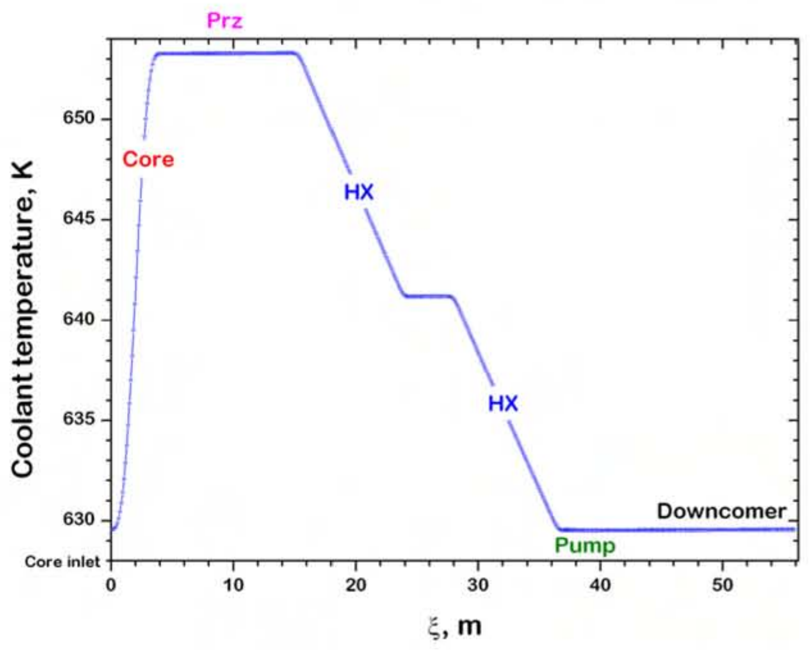

Fig. 4.49 : Steady-state coolant temperature solution.

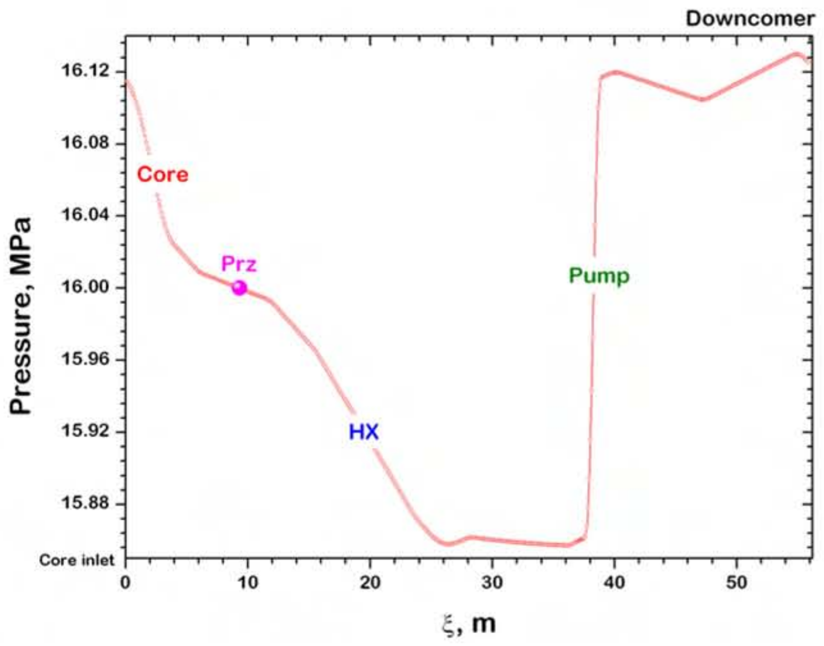

Fig. 4.50 : Steady-state pressure solution.

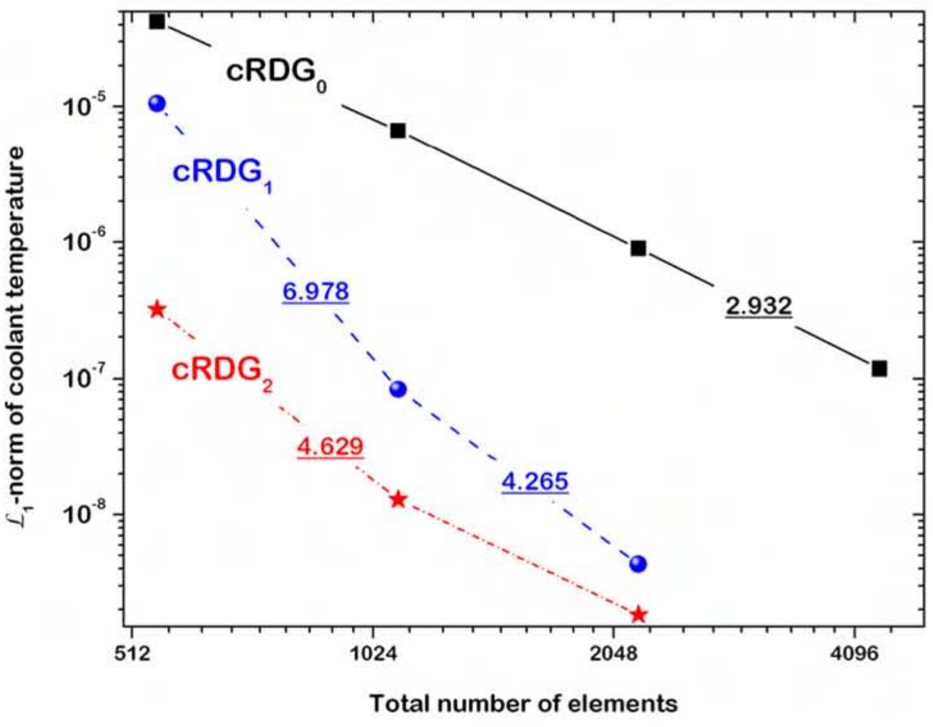

Fig. 4.51 : Convergence in space for coolant temperature in the primary system, using different space discretization schemes.

Simulation of plant operation (including startups and shutdowns over refueling cycles) using this loop configuration is also performed. Figures 4.52-4.53 depicts solutions for the plant operation over a time period of five years. Such a simulation is possible thanking to the code's computational efficiency. It is manifested by the time stepping used that is several orders of magnitude larger than that determined by the CFL limit characteristic of semi-implicit schemes in legacy system codes.

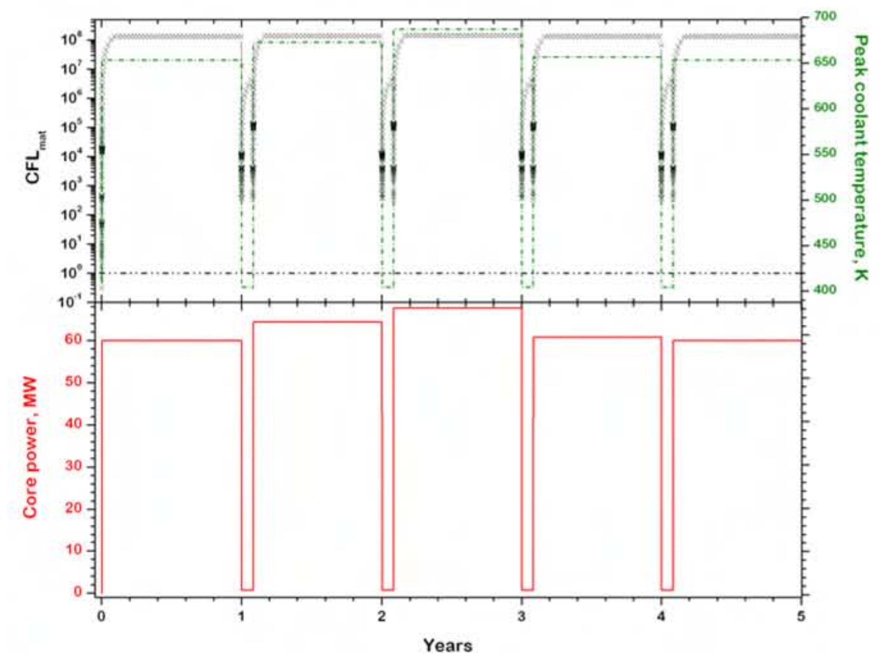

Fig. 4.52 : Solution for 5-year transient. Dynamics of material CFL number vs. core power and peak coolant temperature.

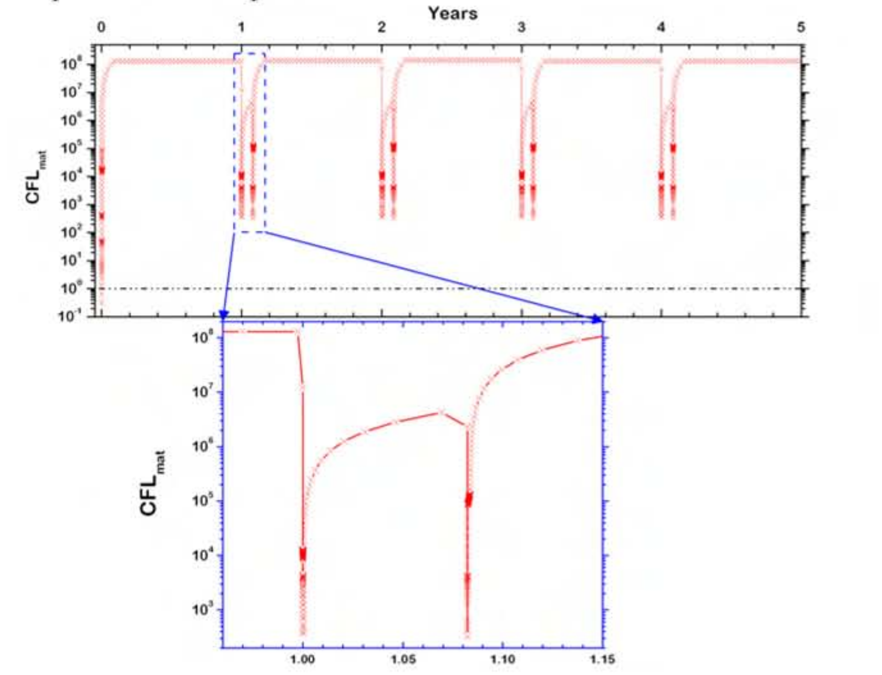

Fig. 4.53 : Solution for 5-year transient. Details of dynamic time control and CFL number.

This Page is Intentionally Left Blank

Chapter 5

Concluding Remarks

WITH its high-order accurate (in both time and space) numerical schemes, the RISMC code β_2 versiuon is computationally efficient, allowing it to be used in both "plant aging" and "safety transient" regimes (with vastly different time scales), and in both deterministic mode and probabilistic mode. Yet, with the introduction of 3D domains in the deterministic mode and the large number of runs required in the probabilistic mode, speed remains a practical challenge. Furthermore, when the plant system modeled grows in complexity, connectivity and volume, advances in multi-physics solution algorithms are central to maintaining the computational efficiency demonstrated for single-phase thermal-hydraulics tests shown in this report.

The speed requirement also varies for different classes of prospective users. More importantly, the perception of speed changes with the character and expected value of the application, and with the availability of computing resources to code users. As computing resources become affordable to utility, the cost of computation would constitute a small fraction of the resources needed to set up the analysis and to process the vast amount of information the code produce [1].

Given the above-discussed variables, in lieu of a strict "numerical" requirement on speed, we suggest the following (Table 5.1) as guidance for development. Input from stakeholders and feedback from early users will be taken into account in revision. The research in LWR-S/RISMC project on high performance comput-

[1] A simulation run estimated 1000 processor x hour - six weeks on a single-processor computer - takes a day in a cluster with 40 processors.

ing and on data aggregation and forecasting will help establish a technical basis to guide further formulation and implementation this requirement in the future version of the RISMC code.

Table 5.1 : The RISMC code's "speed" requirement given by the code run time measured in processor x hour (PxH). Also: DM: deterministic mode; PM: probabilistic mode; RCS: primary reactor coolant system; SCS: secondary coolant system; CON: containment; ICE: instrumentation, control, electrical system.

Version Year	**0D / 1D system**	**0D / 1D system with 3D zones**	Systems included in modeling
β_2 2010	DM: 0.2 PM: 300	N/A	RCS, limited ICE
β_3 2012	DM: 1 PM: 500	DM: 10 PM: 5000	RCS, SCS, limited ICE
β_4 2014	DM: 1 PM: 1000	DM: 20 PM: 10000	RCS, SCC, ICE, limited CON
β_5 2016	DM: 1 PM: 2000	DM: 40 PM: 20000	RCS, SCC, ICE, CON

For the RISMC code β_2 version - which has been developed in the spirit of "VU-assessed" code - verification has accompanied the development in every step. Also greatly helped are the high-order accurate schemes, which are very sensitive to algorithmic implementation error. We document the work on verification including a set of (now 180+) regression tests in a separate report. Most notably, the method of manufactured solutions (MMS) has proven instrumental for verification of fluid flow solver. As we move toward a multi-physics simulation capability, it is critical that the MMS be extended to verify implementation of solution for multi-physics problems. More subtle, and much harder to find out, are erroneous data. The issue commands the need to develop a formalized approach to data storage and data analysis.

Bibliography

[Atk94] P. Atkins. *Physical Chemistry*. Freeman, New York, 5th edition, 1994.

[BBE$^+$04] Satish Balay, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2004.

[BBG$^+$01] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc Web page, 2001. http://www.mcs.anl.gov/petsc.

[BCVK02] H. Bijl, M.H. Carpenter, V.N. Vatsa, and C.A. Kennedy. Implicit time integration schemes for the unsteady compressible Navier-Stokes equations: Laminar flow. *Journal of Computational Physics*, 179:313–329, 2002.

[BGMS97] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[BS90] P.N. Brown and Y. Saad. Hybrid Krylov methods for nonlinear systems of equations. *SIAM Journal of Science and Statistical Computing*, 11:450–480, 1990.

[Cho67] A. J. Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of Computational Physics*, 2:12–26, 1967.

[CHS90] B. Cockburn, S. Hou, and C. W. Shu. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws IV: The multidimensional case. *Mathematics of Computation*, 54:545–581, 1990.

[CKB$^+$05] M.H. Carpenter, C.A. Kennedy, H. Bijl, S.A. Vilken, and V.N. Vatsa. Fourth-order Runge-Kutta schemes for fluid mechanics applications. *SIAM Journal of Scientific Computing*, 25:157–194, 2005.

[CLS89] B. Cockburn, S.-Y. Lin, and C. W. Shu. TVB Runge-Kutta local projection Discontinuous Galerkin finite element method for conservation laws III: One-dimensional systems. *Journal of Computational Physics*, 84:90–113, 1989.

[Coc89] B. Cockburn. *Introduction to Discontinuous Galerkin method for Convection-Dominated Problems*, volume 1697 of *Lecture Notes in Mathematics*. Springer, Berlin/Heidelberg, 1989.

[CS89] B. Cockburn and C. W. Shu. TVB Runge-Kutta local projection Discontinuous Galerkin finite element method for conservation laws II: General framework. *Mathematics of Computation*, 52:411–435, 1989.

[CW84] P. Colella and P. R. Woodward. The Piecewise Parabolic Method (PPM) for gas-dynamical simulations. *Journal of Computational Physics*, 54:174–201, 1984.

[DNT03] T. Dinh, R. Nourgaliev, and T. Theofanous. Understanding the ill-posed two-fluid model. In *The 10th International Topical Meeting on Nuclear Reactor Thermal Hydraulics*, 2003.

[DS83] Jr. Dennis, J.E and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1983.

[EW96] S.C. Eisenstat and H.F. Walker. Choosing the forcing terms in an inexact Newton method. *SIAM Journal of Science and Statistical Computing*, 17:16–32, 1996.

[FMDO98] R. Fedkiw, B. Merriman, R. Donat, and S. Osher. The penultimate scheme for systems of conservation laws: Finite difference ENO with Marquina's flux splitting. In M. Hafez, editor, *Progress in Numerical Solutions of Partial Differential Equations*, Arcachon, France, July 1998.

[Got05] S. Gottlieb. On high order strong stability preserving Runge-Kutta and multi step discretizations. *SIAM Journal of Scientific Computing*, 25(1/2):105–128, 2005.

[GZI$^+$76] S.K. Godunov, A.V. Zabrodin, M.Ya. Ivanov, A.N. Kraiko, and G.P. Prokorov. *Numerical Modeling of Multidimensional Gas Dynamics Problems*. Nauka, Moscow, 1976. In Russian.

[HA71] F. H. Harlow and A. A. Amsden. A numerical fluid dynamics calculation method for all flow speeds. *Journal of Computational Physics*, 8:197–213, 1971.

[JS96] G.-S. Jiang and C.-W. Shu. Efficient implementation of Weighted ENO schemes. *Journal of Computational Physics*, 126:202–228, 1996.

[KK98] George Karypis and Vipin Kumar. MeTiS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, version 4.0. Technical report, University of Minnesota, Department of Computer Science and Engineering, Minneapolis, MN 55455, 1998.

[KK04] D. A. Knoll and D. Keyes. Jacobian-free Newton-Krylov methods: A survey of approaches and applications. *Journal of Computational Physics*, 193:357–397, 2004.

[KMCR05] D.A. Knoll, V.A. Mousseau, L. Chacon, and J. M. Reisner. Jacobian-free Newton-Krylov methods for the accurate time integration of stiff wave systems. *SIAM Journal of Scientific Computing*, 25:213–230, 2005.

[Kri07] L. Krivodonova. Limiters for high-order discontinuous Galerkin methods. *Journal of Computational Physics*, 226:879–896, 2007.

[KSK02] George Karypis, Kirk Schloegel, and Vipin Kumar. ParMeTiS Web page, 2002. http://glaros.dtc.umn.edu/gkhome/.

[KSK03] George Karypis, Kirk Schloegel, and Vipin Kumar. ParMeTiS: Parallel graph partitioning and sparse matrix ordering library, Version 3.1. Technical report, University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN 55455, 2003.

[Lio06] M. Liou. A sequel to AUSM, Part II: AUSM+-up for all speeds. *Journal of Computational Physics*, 214:137–170, 2006.

[LLNM09] H. Luo, L. Luo, R.R. Nourgaliev, and V. Mousseau. A Reconstructed Discontinuous Galerkin method for the compressible Euler equations on arbitrary grids. *Journal of Computational Physics*, 2009. Submitted.

[LvL09] M. Lo and B. van Leer. Analysis and implementation of Recovery-based Discontinuous Galerkin for diffusion. In *19th AIAA Computational Fluid Dynamics Conference*, San Antonio, Texas, USA, June 22-25 2009. AIAA 2009-3786.

[MKR00] V.A. Mousseau, D. A. Knoll, and W. J. Rider. Physics-based preconditioning and the Newton-Krylov method for nonequilliburium radiation diffusion. *Journal of Computational Physics*, 160:743–765, 2000.

[MKR04] V.A. Mousseau, D. A. Knoll, and W. J. Rider. New physics-based preconditioning of implicit methods for nonequilibrium radiation diffusion. *Journal of Computational Physics*, 190:42–51, 2004.

[MSGH84] J.J. Moré, D.C. Sorenson, B.S. Garbow, and K.E. Hillstrom. The MINPACK project. In W.R. Cowell, editor, *Sources and Development of Mathematical Software*, pages 88–111, 1984.

[NDT04] R.R. Nourgaliev, T.N. Dinh, and T.G. Theofanous. The characteristics-based matching method for compressible flow with moving boundaries and interfaces. *ASME Journal of Fluids Engineering*, 125:586–604, 2004.

[NPM09] R.R. Nourgaliev, H.-K. Park, and V. A. Mousseau. Recovery Discontinuous Galerkin Jacobian-free Newton-Krylov method for multiphysics problems. *Computational Fluid Dynamics Review 2008*, 2009. In press.

[Pat80] S. Patankar. *Numerical Heat Transfer and Fluid Flow*. Taylor & Francis, 1980.

[PTVF99] W.H. Press, S. A. Teukilsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipies in C*. Cambridge University Press, 2nd edition, 1999.

[PW98] M. Pernice and H.F. Walker. NITSOL: A Newton iterative solver for nonlinear systems. *SIAM Journal of Science and Statistical Computing*, 19:302–318, 1998.

[RH73] W. H. Reed and T. R. Hill. Triangular mesh methods for the neutron transport equation. Technical Report LA-UR-73-479, Los Alamos Scientific Laboratory Report, 1973.

[RMWK05] M. Reisner, V. A. Mousseau, A. A. Wyszogrodzki, and D. A. Knoll. A fully implicit hurricane model with physics-based preconditioning. *Monthly Weather Review*, 133:1003–1022, 2005.

[RP77] W. C. Reynolds and H. C. Perkins. *Engineering Thermodynamics*. McGraw-Hill, Inc., 2nd edition, 1977.

[RWMK03] J. Reisner, A. Wyszogrodzki, V.A. Mousseau, and D. A. Knoll. An efficient physics-based preconditioner for the fully implicit solution of small-scale thermally driven atmospheric flows. *Journal of Computational Physics*, 189(1):30–44, 2003.

[SA99] R. Saurel and R. Abgrall. A simple methods for compressible multifluid flows. *SIAM Journal of Scientific Computing*, 21(3):1115–1145, 1999.

[Saa03] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2nd edition, 2003.

[SO89] C.-W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *Journal of Computational Physics*, 77:439–471, 1989.

[SS86] Y. Saad and M.H. Schultz. GMRES: a Generalized Minimal Residual algorithm for solving linear systems. *SIAM Journal of Science and Statistical Computing*, 7:856, 1986.

[Tor99] E. F. Toro. *Riemann solvers and numerical methods for fluid dynamics. A practical Introduction.* Springer, Berlin/Heidelberg, 2nd edition, 1999.

[vH97] Dimitri van Heesch. Doxygen Web page, 1997. http://www.stack.nl/ dimitri/doxygen/index.html.

[vLL09] B. van Leer and M. Lo. Unification of Discontinuous Galerkin methods for advection and diffusion. In *47th AIAA Aerospace Sciences Meeting and Exhibit*, Orlando, Florida, USA, 2009. AIAA 2009-0400.

[vLLR07] B. van Leer, M. Lo, and M. Raalte. Discontinuous Galerkin for diffusion based on recovery. In *18th AIAA Computational Fluid Dynamics Conference*, Miami, Florida, USA, June 26-28 2007. AIAA 2007-4083.

[vLN05] B. van Leer and S. Nomura. Discontinuous Galerkin for diffusion. In *17th AIAA Computational Fluid Dynamics Conference*, Toronto, Ontario, Canada, June 6-9 2005. AIAA 2005-5108.

[vRvL08] M. va Raalte and B. van Leer. Bilinear forms for the recovery-based Discontinuous Galerkin method for diffusion. *Communications in Computational Physics*, 5:683–693, 2008.

Glossary

constraint
The range of values, or limits, for a metric that are judged to be acceptable. Performance constraints are assigned to physical variables characterizing device behavior at each critical design point. 82

equations of state
The equations, in algebraic, graphical, or tabular form, which relate the intensice thermodynamic properties of any substance. 107, 111

reconstruction
Using *strong* interpolation for building a polynomial, whose functions and/or its derivatives are matched in *points*. 66, 97

recovery
A *weak* interpolation technique for locally building a polynomial from the given on a computational net piecewise-discontinuous polinomials in such a way, so the $\mathcal{L}_2$-*projections* of the "recovered" and original polynomials coincide. 66, 97

This Page is Intentionally Left Blank

Index

This Page is Intentionally Left Blank