INL/EXT-07-13490

Coupling Schemes for Multiphysics Reactor Simulation

V. S. Mahadeven
J. C. Ragusa

November 2007

The INL is a U.S. Department of Energy National Laboratory
operated by Battelle Energy Alliance

INL/EXT-07-13490

Coupling Schemes for Multiphysics Reactor Simulation

V. S. Mahadeven
J. C. Ragusa

[1]Texas A & M

November 2007

Idaho National Laboratory
Idaho Falls, Idaho 83415

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

Contents

Executive Summary

This report documents the progress of the student Vijay S. Mahadevan from the Nuclear Engineering Department of Texas A&M University over the summer of 2007 during his visit to the INL. The purpose of his visit was to investigate the physics-based preconditioned Jacobian-free Newton-Krylov method applied to physics relevant to nuclear reactor simulation. To this end he studied two test problems that represented reaction-diffusion and advection-reaction. These two test problems will provide the basis for future work in which neutron diffusion, nonlinear heat conduction, and a two-phase flow model will be tightly coupled to provide an accurate model of a BWR core.

1 General Introduction

Reliable and accurate simulations of physical phenomena often require the simultaneous description of several physics components. In most cases, these physics are coupled in a non-linear fashion, making it intricate to find the solution efficiently and accurately. Examples of such phenomena include radiation diffusion, where the radiation energy is strongly coupled with the temperature field, nuclear reactor analysis, where the neutronics and the power are strongly coupled with the thermal-hydraulics field.

Any such multi-physics transient problem involving solution to a nonlinear Partial Differential Equation (PDE) will typically contain 4 different loops in the solution procedure. These are

1. Time stepping loop, which can either be performed using a constant or adaptive time stepping strategy,
2. Nonlinear iteration, which can be performed using a Picard or Newton-type iteration,
3. Linear iteration to solve a linearized system given by $Ax = b$,
4. Preconditioner iteration, where a problem of type $Px = b$ is solved with P, a preconditioner matrix, encompassing some features of matrix A.

Based on these four embedded loops, this report addresses each topic separate sections in order to finally produce and present a consistent scheme to solve nonlinear problems: The Adaptive time stepped, Jacobian-Free Newton Krylov. The details of the algorithm are discussed below in the context of solving some physics problems commonly encountered in reactor analysis.

2 Automated Time Step Control

Coupled multi-physics problems solve different physics with time scales varying by orders of magnitude. In order to correctly resolve the solution fields from each physics accurately, a minimum of all the time scales need to be found and used for advancement. This poses a problem if constant time steps are used because the dynamic time scales of the physics change during a transient and, hence, an algorithm to capture this would be necessary to improve efficiency and to avoid overkilling the problem accuracy, in certain regions of the transient.

The current report only uses 2 simple temporal discretization methods namely, Backward Euler (B.E) $O(h)$ and Crank Nicholson (C.N) $O(h^2)$ by means of a parameterized θ scheme. The discretization using θ scheme for a simple system of equations can be written as

$$\begin{aligned} \frac{\partial y}{\partial t} &= F(y, x) \\ \frac{y^{n+1} - y^n}{\Delta t} &= \theta F^{n+1} + (1 - \theta) F^n \end{aligned} \tag{1}$$

where $\theta = 1$ yields the B.E scheme and $\theta = 0.5$ yields the C.N scheme.

The current work is focused mainly on understanding the dynamical time scales of the problem to adapt time steps, rather than adaptive time stepping based on numerical truncation error which is based on a slightly different philosophy. Some thoughts regarding time adaptation by these different methods and future endeavors necessary are discussed below.

2.1 Time Stepping Strategies

The possible flaw with an adaptive scheme that depends only on the dynamical time scale of the problem is that it fails to address the question of accuracy and reliability of the solution, which is very much dependent on the type of discretization method used. Indeed, a 1-st order and a 4-th order method need not use the same time step based on dynamical time scale to achieve a given accuracy. Previous work [1] in the context of multi-physics coupling using Local Truncation Error (LTE) for higher order methods using standard and predictive time step controller, have proved to be effective in this respect. But there is the lack of information on the physical time scale of the problem from numerical LTE for a particular discretization method since these are buried in the constants preceding the higher order derivatives in LTE. Also, certain assumptions made in deriving the standard controller such as smoothness of the solution and constancy of the local order of the method might fail in several transient scenarios and hence possibly leading to a different estimate of the LTE as compared to the true local error. The basic equations for both adaptive time stepping approaches are shown below:

Dynamical time scale:

$$\tau_{dyn} = \frac{S}{\left|\frac{1}{\phi}\frac{\partial \phi}{\partial t}\right|} \tag{2}$$

where ϕ is the solution field for current physics; S is a safety factor

Standard controller:

$$\tau_{num} = S\left[\frac{\epsilon}{LTE}\right]^{\frac{1}{p}} \tag{3}$$

where ϵ is a user-specified accuracy tolerance, LTE is the local truncation error based on the numerical scheme and p is the local order of accuracy.

Future work on combining these two different techniques to create a hybrid time adaptation algorithm might yield the best results since it would account for the local accuracy of the solution and while following the time scale of the problem evolution itself, thereby resolving the solution accurately. Then, stringent criteria to make the hybrid controller predict optimal time steps can be ensured by using information on convergence of the nonlinear iteration, linear iteration and conditions such as $\frac{\epsilon}{LTE} - 1 > 0$, if $(\tau_{num}(p) < \tau_{num}(p-1))$ to ensure that we are in the asymptotic regime. The usage of such predictive capability and success of these methods still remains to be tested for higher dimensional, real world transients but nevertheless, the lessons learnt and results from smaller problems are assuring that we are heading in the right direction. Additionally, it is also imperative to note that the global error in a transient calculation should also be bounded and appropriate temporal schemes with such inherent properties need to be used.

3 Jacobian-Free Newton Krylov (JFNK) Algorithm

3.1 Introduction

The JFNK method, as the name suggests, uses Newton's method as the nonlinear solver and Krylov methods as the linear solver to solve a set of nonlinear equations effectively and accurately. The Jacobian-Free approximation implies that the algorithm can be implemented without explicitly building the matrix needed in the linear solve. Often, building the Jacobian matrix can be costly in CPU time and memory, especially when different physic components reside in multiple codes. On the other hand, a Krylov method may require a certain number of basis vector to be stored in order to find an accurate solution. The Krylov space size and the overall computing time can be significantly reduced by the use of an appropriate preconditioner. As mentioned before, the JFNK algorithm consists of 3 levels of iterations: 1) Newton iteration 2) Krylov

iteration 3) Preconditioner iteration. Let us now briefly look into the equations involved in implementing the algorithm.

3.2 Newton's Method

Let us consider a system of nonlinear equations of the form $F(y) = 0$ (e.g., obtained by implicit time differentiation of (1)). Expanding the discrete equation using Taylor series, and neglecting higher order terms, we can then obtain

$$F(y^{k+1}) = F(y^k) + \frac{\partial F(y^k)}{\partial y}(y^{k+1} - y^k) \tag{4}$$

If we define $J(y^k) = \frac{\partial F(y^k)}{\partial y}$ as the Jacobian matrix of the system, then the Newton update at every iteration is simply given by

$$J(y^k)\delta y = -F(y^k) \tag{5}$$

with $\delta y = y^{k+1} - y^k$.

It is quite clear that the above equation requires forming the Jacobian matrix explicitly in order to solve the system for δy. This can be expensive in terms of computational time if a finite difference procedure by perturbing $F(y)$ is used to find J element by element (numerical Jacobian). Alternately, a different algorithm can be employed that does not require the Jacobian matrix itself but rather only its action on a given vector. This operation will be required during the linear solve of (5). For a given vector v, the action of the Jacobian on the vector can be computed using the following equation

$$Jv \approx \frac{F(y + \epsilon v) - F(y)}{\epsilon} \tag{6}$$

where ϵ is a parameter used to control the magnitude of perturbation. Given this introduction, let us look at a particular Krylov method, namely the GMRES method and analyze how the JFNK framework can be implemented.

3.3 Krylov Method: GMRES

The GMRES (Generalized Minimum RESidual) method, introduced by Saad and Schultz [2], is a popular and efficient Krylov subspace method used to solve nonsymmetric system of equations. The GMRES algorithm generates a sequence of orthogonal vectors, and because the matrix being inverted is not symmetric, short recurrence relations cannot be used as in the case of the Conjugate Gradient algorithm. Instead, all previously computed vectors in the orthogonal sequence have to be retained. One matrix-vector product is required per iteration and the matrix-free approximation introduced earlier in (6) can be used to create the Jacobian-Free framework. Detailed information on the exact numerics and implementation of GMRES in the JFNK framework has been shown in [3]. Optimal equation for choosing the perturbation parameter ϵ has also been derived in reference paper [3].

3.4 Preconditioners for the Linear Iteration

As introduced earlier, highly nonlinear problems typically possess a preconditioning loop inside every GMRES iteration. A right preconditioning technique applied to the above Jacobian-Free Newton framework yields

$$(JP^{-1})(P\delta y) = -F(y^k) \tag{7}$$

where P is the preconditioning matrix.
The preconditioner P is usually a good approximation of the Jacobian but should be easier to form and solve as compared to the Jacobian matrix itself. Then the inherently two-step process shown above requires the computation of the action of P^{-1} on any vector v, rather than actually forming the preconditioning matrix itself. Such an algorithm would strictly be "matrix-free" and future studies for higher dimensional real world problems will be conducted to determine the efficiency of this approach.

Quoting from Knoll and Keyes [2], the preconditioning algorithm can be explained as follows.
The operation of Eq.(7) is done once per GMRES iteration and is performed in two steps:

1. Preconditioning: Solve (approximately) for x in $Px = v$
2. Perform the matrix-free product Jx $(= JP^{-1}v)$ by using (6)

Only the matrix elements required for the action of P^{-1} are formed. There are two primary choices to be made:

1. What linearization should be used to form the matrices required in P^{-1}?

   Based on the physics, depending on which modes are dominant, the linearization should implicitly handling the dominant modes while treating the other modes in a linearized fashion. This will reduce the spread of the eigenvalues of JP^{-1} thereby creating the desired effect in the Krylov iteration (reduce the total number of required iterations).

2. What linear iterative method should be used for $y = P^{-1}v$?

   For nontrivial multidimensional problems, iterative methods would be the best option instead of a direct solve using Banded solvers or by Gaussian elimination. An obvious option would be to use GMRES again to solve the preconditioner problem in a Matrix free way in which case, we would only require action of P on a vector v. This form is recursive and care is needed to implement this framework in a modular fashion.

More discussion on physics-based preconditioning will be included when dealing with individual physics components.

4 Physical Phenomena

In the current report, two specific types of physics will be further discussed within the scope of JFNK applications. They are:

1. a nonlinear heat conduction for reaction-diffusion dominated physics and,
2. a nonlinear fluid flow simulation for reaction-advection dominated physics.

We will look each of these in more detail and discuss the usage of JFNK methodology and preconditioners in this context.

4.1 Reaction-Diffusion: 1-D Nonlinear Thermal Conduction

The general unsteady nonlinear thermal conduction equation can be written in the following form

$$\frac{\partial T(x)}{\partial t} - \frac{\partial}{\partial x}\left(k(T)\frac{\partial}{\partial x}T(x)\right) = f(T) \tag{8}$$

where $k(T)$ is the temperature-dependent conductivity of the material.

It is obvious that (8) is a parabolic equation which can be solved with a consistent spatial and temporal discretization scheme. For spatial discretization, a traditional Finite Element Method with arbitrary order piecewise Lagrange polynomials as basis functions was used to obtain the solution. Several experiments to observe the effectiveness of Preconditioning techniques were also simulated by controlling the nonlinearity arising from $k(T)$. The results on the order of accuracy and efficiency of the devised numerical schemes are discussed in the following sections for different test cases.

4.1.1 Thermal Wave Problem

Let us assume that $k(T) = 1$, then the diffusion equation (8) has the following form

$$\frac{\partial T(x)}{\partial t} - \frac{\partial}{\partial x}\left(\frac{\partial}{\partial x}T(x)\right) = f(T) \tag{9}$$

This test problem was previously introduced in [4]. Then, by assuming a particular solution for $T(x)$, a suitable forcing function can be obtained by substituting in (9), a method more commonly known as the Method of Manufactured Solutions (MMS). Let us consider a test problem where a thermal wave is travelling with a constant velocity c across the domain with a wave width of δ. Assume the exact solution to be of the form

$$T(x,t) = \frac{1}{2}\left(1 - \tanh(\frac{x - ct}{\delta})\right) \tag{10}$$

with $c = 2$ and $\delta = 1$ with $T \in [0, 1]$ for a specific case. Then the forcing function can be obtained after substitution in (9) as

$$f(T) = \frac{2T(1-T)(c\delta - 2 + 4T)}{\delta^2} \tag{11}$$

and with $c = 2$ and $\delta = 1$, the forcing function reduces to

$$f(T) = 8T^2(1 - T) \tag{12}$$

Using equation (10) and (11), numerical simulation can be performed to test the validity of the discretization schemes implemented. In other words, using the exact solution as the reference, the spatial and temporal truncation errors can be measured and checked if they are consistent with theoretical orders of accuracy.

4.1.2 Spatial Accuracy

In the current study, as mentioned before, FEM spatial discretization is used with piecewise continuous Lagrange polynomials as basis functions. The code is flexible enough to accommodate polynomials of arbitrary order. For the current work, since only a second order spatial accuracy is a requirement, the order of convergence for $p = 1$ is shown in Fig. 1 (using a constant step size of $\Delta T = 1E - 4$ to eliminate temporal errors). Hence, the code is consistent in the treatment of the spatial derivatives and matches analytical results.

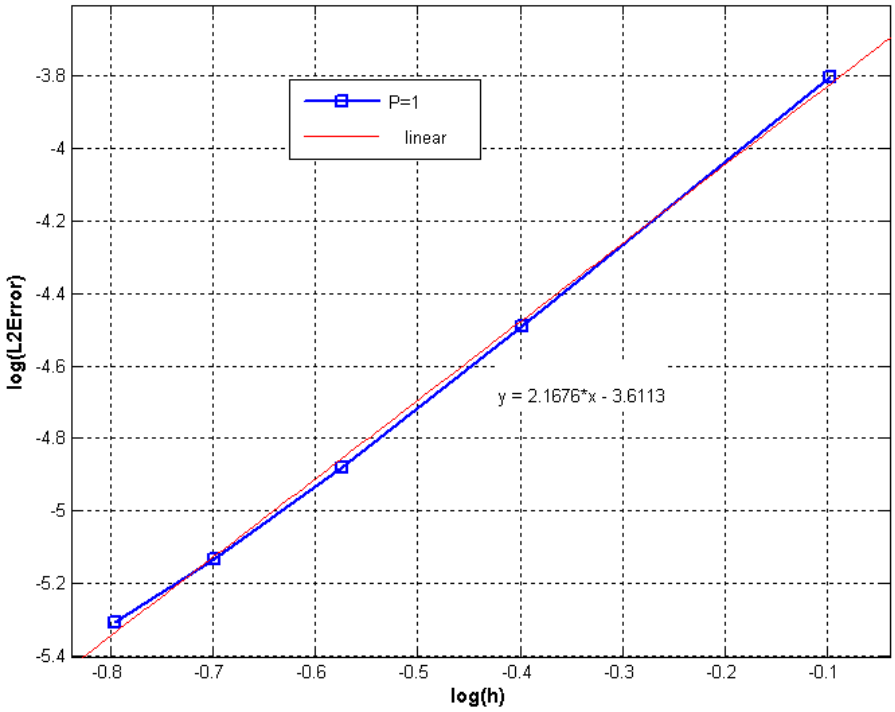


Figure 1: Spatial order of convergence with $\Delta T = 1E-4$ and $L = 40$.

4.1.3 Temporal Accuracy

In the current study, the θ time discretization is used to obtain B.E and C.N with $\theta = 1.0$ and $\theta = 0.5$ respectively. The extension of the numerical scheme to higher order methods is quite straightforward and if the nonlinearities are resolved, higher orders of temporal accuracy is achievable. A plot for the temporal order of convergence for B.E and C.N schemes is shown in Fig. 2.

Again, if the spatial mesh used is fine enough ($N \geqslant 200$ nodes), at the asymptotic limit, the orders of convergence obtained for both the time discretization schemes are as expected. Hence, the code is consistent in the treatment of the temporal derivative and matches the theoretical results.

4.1.4 Code Consistency

From the results presented above, it is quite clear that the code framework based on JFNK technique is consistent and accurate both spatially and temporally. Hence, results shown and discussed can be replicated with the given assumptions.

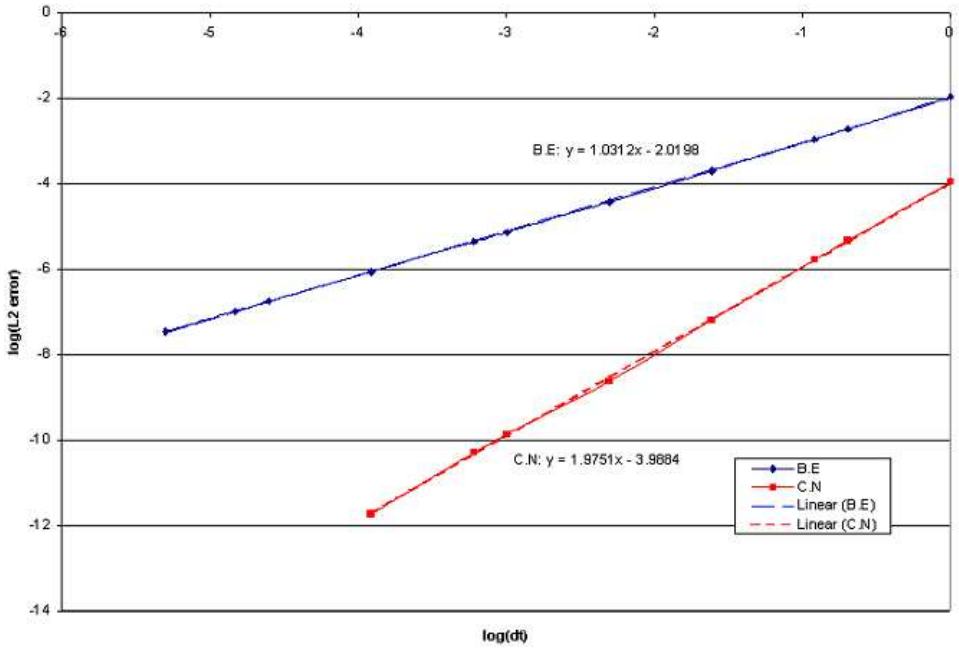


Figure 2: Temporal order of convergence with N=300.

4.1.5 Marshak Wave Problem

Consider a nonlinear diffusion equation of the following general form with $k(T) = T^\alpha$,

$$\frac{\partial T(x)}{\partial t} - \frac{\partial}{\partial x}\left(T^\alpha \frac{\partial}{\partial x} T(x)\right) = 0 \tag{13}$$

where $\alpha \geqslant 0$ is the nonlinearity index of the problem. This test case problem provides the flexibility to increase the nonlinearity in the problem and observe the cost of computation required as a function of the nonlinearity index. This also provides an opportunity to test and evaluate the effect, efficiency of the preconditioners on reducing the total number of Krylov iterations required to solve the system at each time step. For the process of simulating the problem, let us consider the following initial and boundary conditions.

Problem dimension: $L = 1$,
B.C: $T(0,t) = a, T(1,t) = b, t > 0, \;\; a = 1, b = 0.1$,
I.C: $T(x,0) = b$

It is also worthy to note that the ratio $(a/b)^\alpha = 10^\alpha$ is an intuitive measure of stiffness of the nonlinear system being simulated. Hence, this ratio can be increased along with the nonlinearity index to actually break the numerical simulation. Such experiments will hopefully provide us good insight into the JFNK machinery, which is important when the problem being solved is large (future work).

The nonlinear residual equation r_{cond} for the Marshak wave problem at $t = t^{n+1}$, in the semi-discrete form can be written as

$$r_{\text{cond}}(T^{n+1}) = \frac{T^{n+1} - T^n}{\Delta t} - \theta \frac{\partial}{\partial x}\left(\left(T^{n+1}\right)^\alpha \frac{\partial}{\partial x} T^{n+1}(x)\right) - (1-\theta)\frac{\partial}{\partial x}\left(\left(T^n\right)^\alpha \frac{\partial}{\partial x} T^n(x)\right) \tag{14}$$

In the fully discrete form, the term $\frac{\partial}{\partial x}\left(T^\alpha \frac{\partial}{\partial x} T(x)\right)$ will be replaced by a nonlinear stiffness matrix from FEM discretization that needs to be computed at every nonlinear iteration.

4.1.6 Reference Run: Temperature Profiles

The problem was simulated for various nonlinearity index values and the plot of the temperature profile for the different cases is shown in Fig. 3. The profiles give an idea of the complexity of the evolution of the solution which will become clear in the next section when we look at the increase in computational cost with the nonlinearity. It is intuitive that the problem becomes more and more diffusive as α increases and this explains why the profiles evolve the way they do in the plot shown in Fig. 3. Plot of temperature profiles for Marshak wave problem $\alpha \in [0, 5]$.

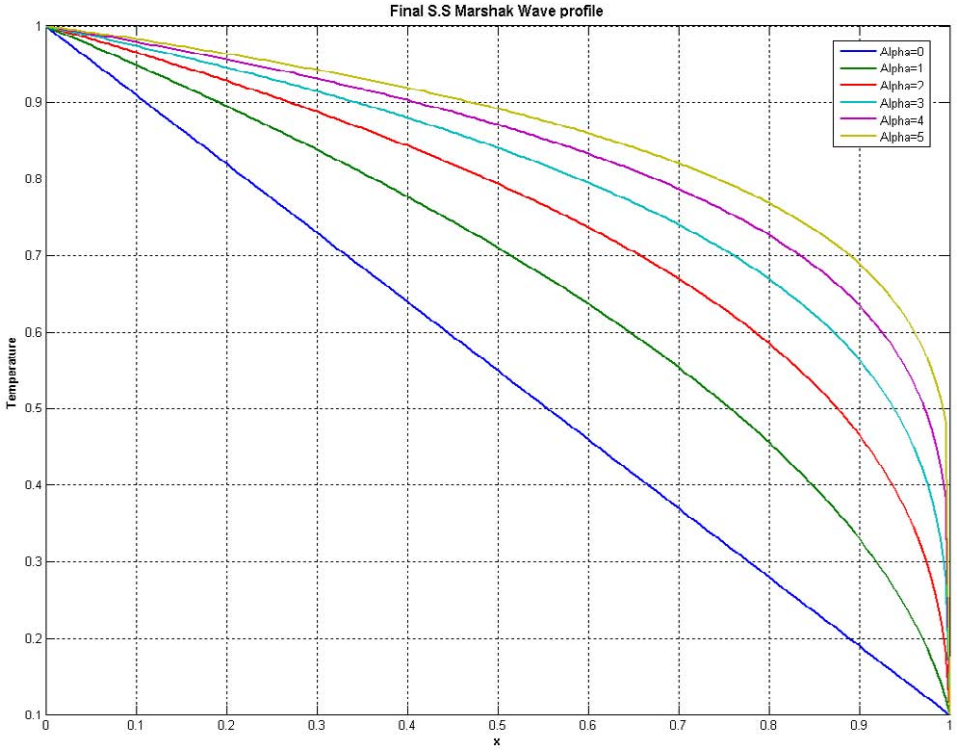


Figure 3: Temperature profile for the Marshak wave problem as a function of the nonlinearity index at T=100 secs.

4.1.7 Preconditioning Techniques

Several types of preconditioners have been tested to gain insight into the physics-based preconditioning techniques; these techniques that have been proven to be quite efficient (see [5, 6]). Few preconditioners that are trivial to compute and solve, such as a Point Jacobi or a Gauss Seidel preconditioner, can be quite devastating in terms of performance for certain kinds of problems. Understanding such subtleties in terms of the physics will provide us with a good intuition to create and implement better physics-based preconditioners.

Note that the implementation of the preconditioners is also done in a matrix-free fashion, wherein only the action of the preconditioner on a vector is performed when required to solve the preconditioned system. This considerably reduces the memory footprint required to solve large nonlinear problems, i.e., when number of unknowns is large.

4.1.7.1 Point-Jacobi Preconditioner A simple Point-Jacobi is the traditional numerical preconditioner that accounts for only the local variation in the system. There are no linearization assumptions involved in such a preconditioner but the important basis of such a preconditioner is that influence from surrounding nodes are small thereby assuming weak coupling in space. Numerically, it can visualized as the diagonal part of the actual Jacobian obtained through a finite difference perturbation method, hence requiring the evaluation of 1 nonlinear residual function (Eq.(14)) per node.

4.1.7.2 Physics-Based Preconditioner An efficient preconditioner can be obtained by linearizing (8) at about the last Newton's iteration. If the point in the state space about the last Newton's iteration is represented as (*), then the semi-discrete form of the linearized equation can be written as

$$\frac{\delta T}{\Delta t} - \theta \frac{\partial}{\partial x}\left(k(T^*)\frac{\partial}{\partial x}(\delta T)\right) = -r_{\text{cond}}(T^*) \tag{15}$$

where θ is the time discretization parameter, $k(T) = T^\alpha$, $\delta T = T - T^*$ is the change in solution from the last Newton update and r_{cond} is the nonlinear residual Eqn (14) evaluated at T^*.

There are several optimizations that can be made in order to avoid the recomputation of the preconditioner at every linear iteration and do it only once per Newton's iteration. For example, the diffusion coefficient can be computed and stored in a vector that can be reused every time the action of the preconditioner on a vector is required (i.e., re-use is possible in the context of the linear solve embedded between each Newton's iteration). Furthermore, if the order of spatial discretization is known apriori, then the local FEM mass matrix can be hard-coded to achieve the performance boost. These steps ensure that creation of the preconditioner itself is not expensive enough to prohibit its usage for complex problems.

4.1.7.3 Inconsistent Physics-Based Preconditioner Preconditioners should be cheap to form and to solve or else it defeats the purpose of solving the original linear system without a preconditioner. With that philosophy in mind, let us consider a physics-based preconditioner that is inconsistent with the original discretization of the nonlinear system, i.e., in this case, use a different discretization for the preconditioner that is not the same as the original spatial discretization. Such a decision leads to a gain in computational time for evaluation of the stiffness matrix in FEM discretization since an easily evaluatable function can be created to reduce the number of shape functions calls per element. A comparison to a consistent preconditioner will be provided and the potential gain from such a treatment will be discussed based on the results from the Marshak problem.

4.1.7.4 Preconditioner Efficiency Different kinds of preconditioners, introduced before, were tried out on the Marshak wave problem to analyze the efficiency of each of the techniques. Although, CPU time measurement would be more practical, due to reasons that this experiment has been conducted in a 1-D problem and that optimizations in the implementation have not been performed, an intuition can be developed by looking at the gain in terms of number of iterations.

Fig. 4 shows the total number of GMRES iterations/Newton Iteration vs the nonlinearity index of the problem with an adaptive time stepping scheme based on the dynamical time scale of the problem. Time adaptivity was included in order to test the efficiency of the preconditioner when larger time steps are taken which is the motivation behind the current research. Based on the simulation, we can conclude which kind of preconditioners will best perform for a given nonlinear problem.

From the results, it is clear that for nonlinear problems, the Point-Jacobi preconditioner is inefficient. This is also intuitive from the physics stand point since as the problem becomes more diffusive, resolving only the local variations does not reduce the other dominant modes arising from coupling with adjacent nodes. Hence, preconditioners that resolve the node-node coupling more effectively, might yield better result in such diffusive problems. This is one of the motivations behind the physics-based preconditioners introduced in the

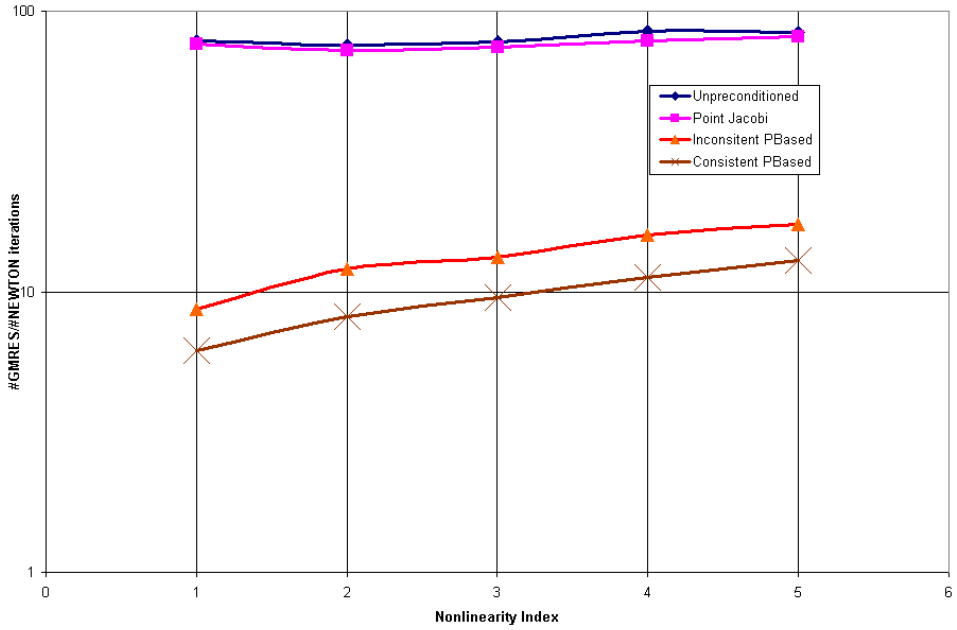


Figure 4: Plot of efficiency for different kinds of preconditioners: both physics-based and numerical

context of nonlinear conduction problem. On the other hand, the linearized, physics based preconditioners perform quite well and reduce the total iterations required to find the problem solution by an average factor of 10.

While comparing the consistent and inconsistent physics-based preconditioner performance, we can definitely see that the consistent preconditioner performs better than the inconsistent preconditioner by a factor of 1.5. This could prove important, especially for large problems, since the preconditioner efficiency dictates the storage requirements for the GMRES iteration (number of Krylov vectors to store obtain the solution). Hence, the cost of the total computation could be quite large, even with an inconsistent preconditioner. But since it is hard to gauge the percentage of computation time increase in the formation of the stiffness matrix itself in both cases, it is sufficient to say that a consistent preconditioner should be effective for larger problems and an inconsistent preconditioner could yield better gains for small nonlinear problems. Further efficiency studies should be performed for higher dimensional problems based on the intuition gained for the current nonlinear conduction problem in 1-D.

4.1.8 Conclusions

The nonlinear diffusion problem is solved using a spatially and temporally consistent JFNK technique and the performance gain from using several types of preconditioners was analyzed. From the sample runs for problems with high nonlinearity, a consistent linearized physics based preconditioning technique offered the best reduction in the number of total iterations needed. Although, solving this kind of physics preconditioner is just as expensive as solving the original system in this case (both a tridiagonal linear systems); for a higher dimensional problem, with larger bandwidth, the linearized form can be used to create an easily invertible system. Since it is not the intention of this report to find the efficiency of the physics-based preconditioning technique for diffusion problems but rather use it as a test case to understand and develop intuition on the method itself, efficiency analysis will be left for future work in higher dimensional problems.

One important advantage of using a consistent physics-based preconditioner is that it can be used as

a solver also, as long as the time step chosen is such that the linearization is accurate enough to provide nonlinear convergence. Such an adaptive procedure will improve the total cost of solution computation since only one Picard iteration is required and the GMRES linear solve is completely avoided. Such a scheme opens up the possibility to adaptively precondition the linear GMRES iterations when using large Δt and use the preconditioner as a solver with Picard iterations when using a Δt that is small enough such that the linearization is an accurate enough assumption.

4.2 Reaction-Advection: Nonlinear Euler Equation

The general nonlinear Euler equations for unsteady fluid flow can be written in the conservative form as follows (for generality, a right-hand-side term is included, this will be necessary for manufactured solutions):

$$\frac{\partial U(x)}{\partial t} + \frac{\partial}{\partial x} F(U) = S(U) \tag{16}$$

where

$$U = \begin{bmatrix} \rho : Density \\ \rho v : Momentum \\ E : Total\ energy \end{bmatrix}; \qquad F(U) = \begin{bmatrix} \rho v \\ \rho v^2 + P \\ vE + vP \end{bmatrix}$$

and the Pressure P is given by the closure relation, the Equation of State (EOS), in a linearized form as

$$P = \frac{\partial P}{\partial \rho}\rho + \frac{\partial P}{\partial E}E \tag{17}$$

The spatial discretization is performed using the Finite Volume method with a first order Upwind differencing scheme (UDS) or a second order Central differencing scheme (CDS) with use of Local Lax Friedrich flux when needed. Higher order spatial discretization using Piecewise Parabolic method (PPM) and Weighted Essential Non-Oscillatory (WENO) schemes can be used in the future to obtain higher orders of spatial accuracy if necessary.

4.2.1 Shock Tube Problem

The well known Sod shock tube problem was used as a test problem and the results were compared and plotted for different spatial discretization methods. In addition to UDS and CDS discretization, a Local-Lax-Friedrich flux (LLF) definition [7] was used with CDS to obtain a more consistent form and to eliminate the spatial oscillations that are observed with CDS at points of sharp discontinuity. This can clearly be seen in the plot of the density profile for the sod problem shown in Fig. 5.

The result matches theoretical predictions that the second order CDS is oscillatory when resolving sudden discontinuities. The oscillations observed at the shock interface are eliminated by using LLF+CDS but the smoothened solution obtained with LLF flux is more diffusive than the UDS solution. The additional viscosity added through the LLF flux is sufficient to eliminate all spurious oscillations at the price of some diffusion in the solution near sharp changes.

The obtained results agree with expected theoretical exact solution from Godunov based scheme and hence prove the validity of the implemented FVM code to solve the Euler equations reliably. Let us now look at a manufactured solution in order to observe the convergence rates in space and time and the preconditioners efficiency.

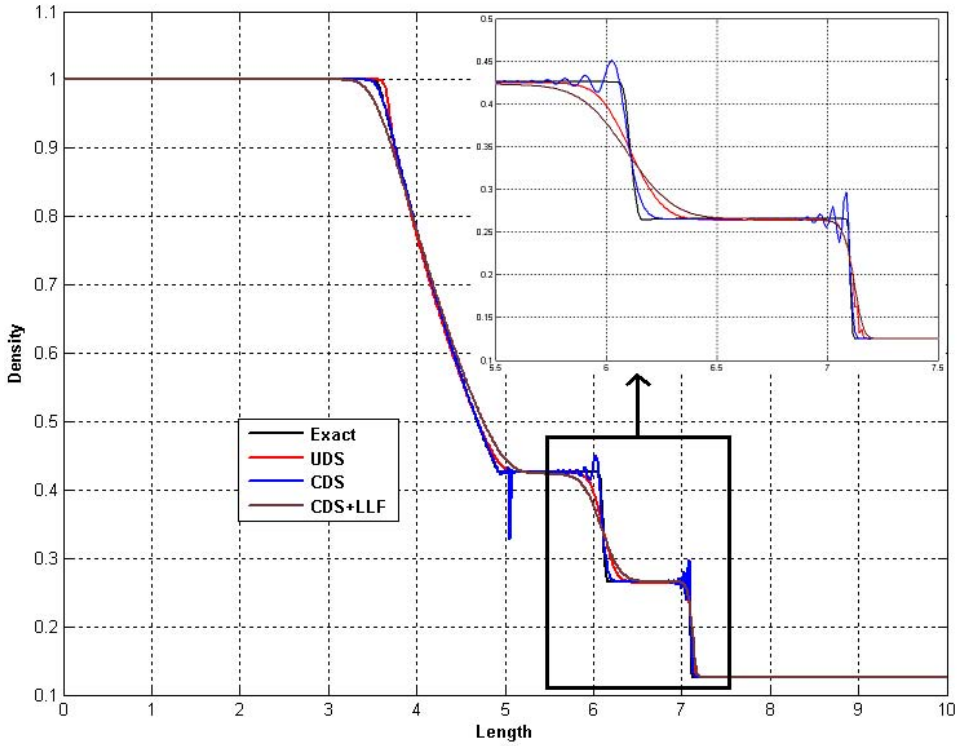


Figure 5: Density profile at $t = 1.2$ sec for the Sod Shock tube problem with different kinds of spatial discretization methods

4.2.2 Manufactured Solution (MMS1)

Using MMS, profiles for the state variables are assumed and accordingly forcing functions for the continuity, momentum and energy equations are found. The equations used are shown below.

$$\rho = \rho_{\min} + (\rho_{\max} - \rho_{\min}) \sec h(\frac{x - \varpi t}{\delta}) \tag{18}$$

$$v = v_{\min} + (v_{\max} - v_{\min}) \sec h(\frac{x - \varpi t}{\delta}) \tag{19}$$

$$E = E_{\min} + (E_{\max} - E_{\min}) \tanh(\frac{x - \varpi t}{\delta}) \tag{20}$$

Using these exact solutions, the source function were found and the solution procedure was implemented to check the order of convergence in space and time. Also, constants for $\frac{\partial P}{\partial \rho}$ and $\frac{\partial P}{\partial E}$ are assumed such that a low Mach flow is simulated. The mach number Ma was found out using

$$Ma = \left(\frac{\max(v(x_i))}{\sqrt{\frac{\partial P}{\partial \rho}}} \right), \; x_i \in [-L/2; L/2]$$

Let us now look at the convergence results for the nonlinear Euler equations using JFNK method.

4.2.3 Spatial Accuracy

As mentioned before, spatial discretization using FVM with UDS and CDS should theoretically yield $O(\Delta x)$ and $O(\Delta x^2)$ respectively. This was tested in the current setting and the results for the convergence order are plotted below with a fine constant Δt to eliminate temporal discretization errors.

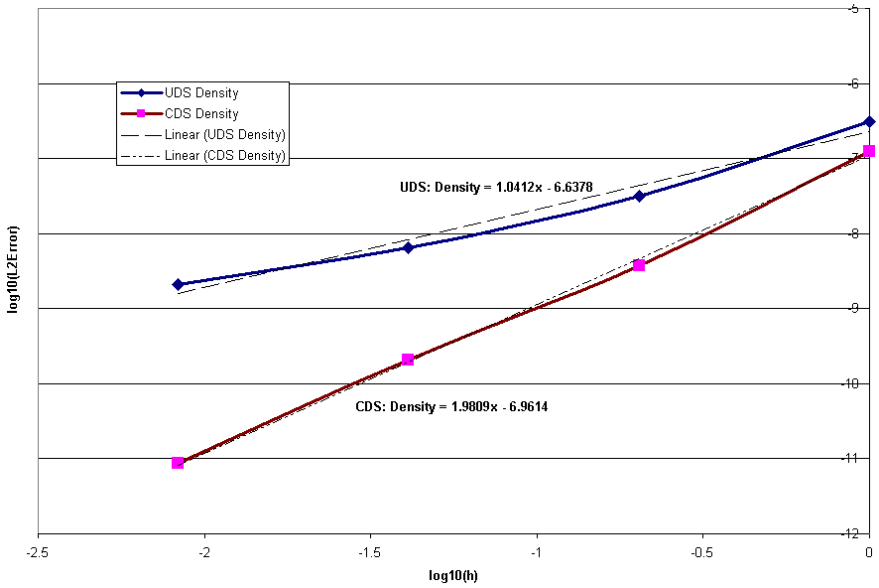


Figure 6: Spatial order of convergence with a fine $\Delta t = 1E-4$ with CN scheme for temporal discretization

The orders of convergence obtained for the differencing schemes tested are as expected. Hence, the code is consistent in the treatment of the spatially discretized terms.

4.2.4 Temporal Accuracy

Using the θ time discretization, the B.E and C.N with $\theta = 1.0$ and $\theta = 0.5$ were used with constant time stepping to test the temporal order of accuracy. Plot for the temporal order of convergence for B.E and C.N schemes are shown in Figs. 7 and 8. It is clear from the two plots that all the state variables density, momentum and energy are $O(\Delta t)$ for Backward Euler and $O(\Delta t^2)$ for Crank Nicholson scheme. These results prove that the code is consistent in the treatment of the temporal derivatives for the Euler equations.

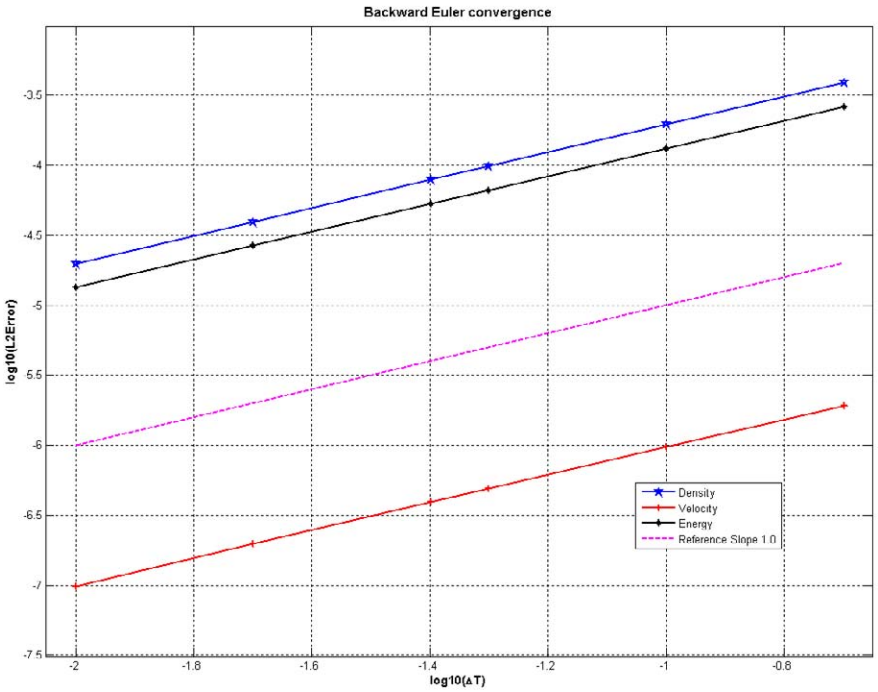


Figure 7: Temporal order of convergence for B.E discretization for all the state variables with $N = 300$ and reference $\Delta t = 1E-3$

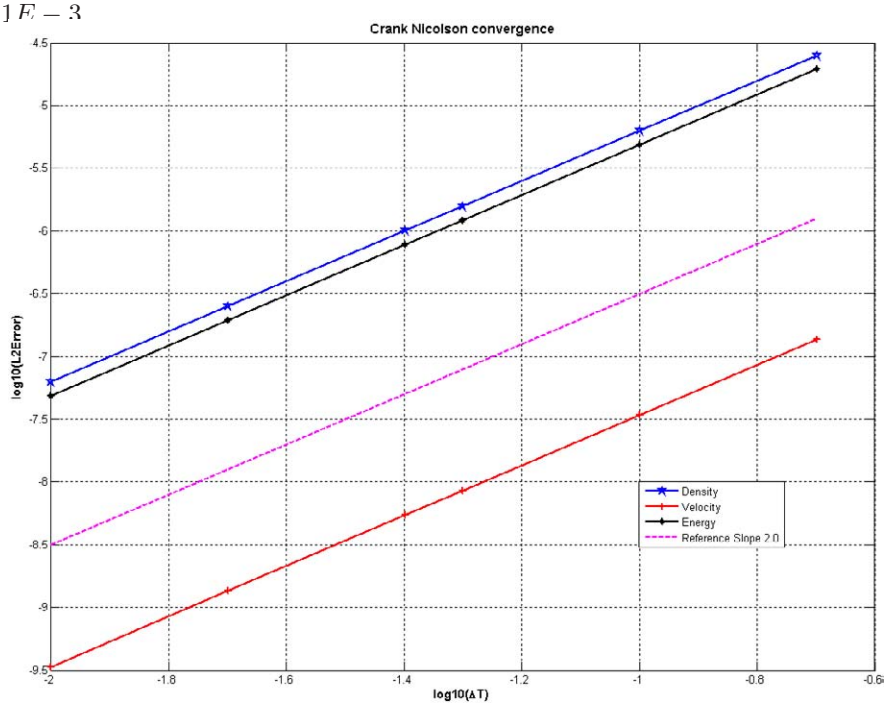


Figure 8: Temporal order of convergence for C.N discretization for all the state variables with $N = 300$ and reference $\Delta t = 1E-3$

Deficiency in adaptive time stepping based on dynamical time scale: Initial trial runs were performed with an adaptive time stepping procedure based on the **dynamical** time to compute the reference solution and to determine the convergence order of time discretization with constant time stepping. The results from these runs were sub-par in terms of expected convergence order since the adaptation based on the dynamical time scale of the system did not yield accurate reference solution **without posteriori error analysis and correction**. The adaptive scheme satisfies the physical time scales and is consistent but because there is no information regarding the local and/or global error of the computed solution, the resulting solution at the end of transient might have levels of inaccuracy outside of the desired range. This is one of the inherent disadvantages of this adaptive strategy.

The safety factor (S) in Equation (2) can be controlled to obtain the desired accuracy using the adaptive dynamical time stepping procedure. To show the consistency of this scheme, the global error in the solution obtained using different S values in comparison to a reference solution calculated using constant time steps with fine $\Delta t = 5E-4$. is plotted and shown in Fig. 9.

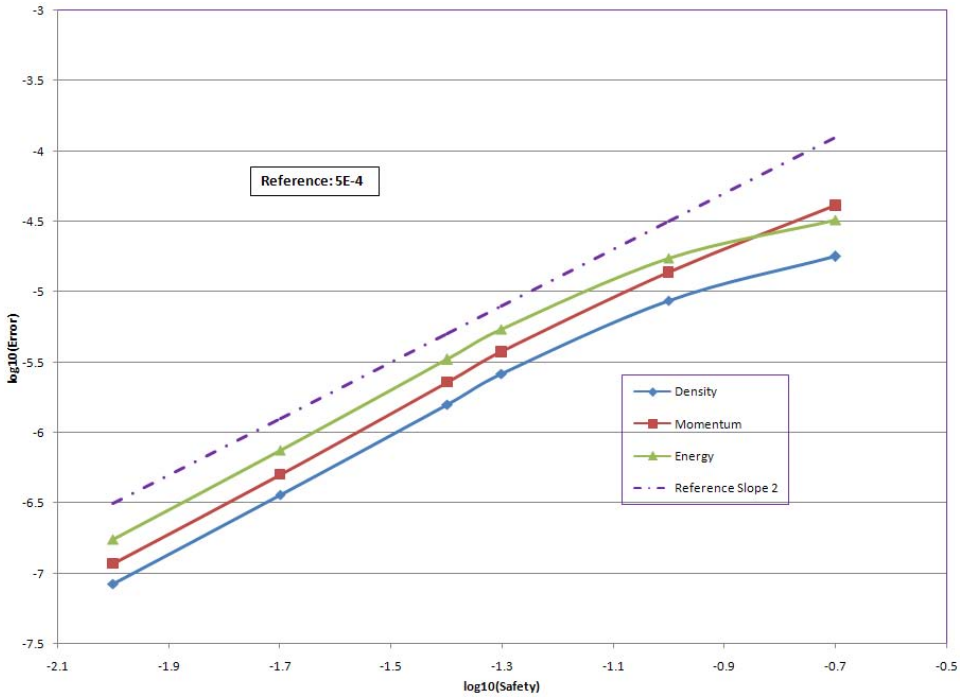


Figure 9: Temporal adaptive convergence order for C.N discretization with reference $\Delta t = 5E-4$

It is quite clear that the convergence order of the adaptive strategy for CN scheme is O(S^2) for all the state variables. But such a figure needs to be calculated for every new problem to obtain a solution with given accuracy since the dynamical time scale for each problem is different. Hence, a safety factor cannot be chosen without error analysis for a given problem and re-simulation of the transient. Also, it is obvious that every time a different S is chosen, the entire sequence of adaptive time steps is homogeneously scaled up or down (depending whether S decreases or increases). Hence, in order to obtain a better accuracy in a localized region of the transient, time steps for the whole calculation are rescaled by S. This is another drawback of this procedure in comparison with a LTE-based adaptation. This points out the inherent drawback of the dynamical time stepping technique and stresses the necessity to move toward a hybrid time stepping strategy that incorporates the LTE in the dynamical time stepping procedure in order to obtain more reliable and

accurate solutions.

4.2.5 Preconditioning Techniques

As in the nonlinear diffusion problem, several kinds of preconditioners were tested to analyze the efficiency in reducing the number of linear iterations. Physics based preconditioners that concentrate on eliminating the behavior of certain terms reduce the spectral radius of the linear system being solved. Such preconditioners have been tested for the nonlinear Euler equations by several researchers [8, 9]. Using techniques like these, the Implicit Continuous Eulerian (ICE) physics based preconditioner was derived and used to produce an efficient algorithm to solve the problem at hand. Detailed derivation and the discussion on the theory behind ICE is presented next.

4.2.5.1 Physics-Based Preconditioner: ICE For simplicity, let us redefine the momentum variable as M. Then the semi-discrete form of the equations, which are essentially the nonlinear residual for the continuity, momentum and energy equations, can be written as:

$r_C(n+1)$:

$$\frac{\rho^{n+1}-\rho^n}{\Delta t} + \theta\partial_x\left(M\right) + (1-\theta)\partial_x\left(M\right) = \theta S_C(\rho,M,E)\rfloor^{n+1} + (1-\theta)S_C(\rho,M,E)\rfloor^n \tag{21}$$

$r_M(n+1)$:

$$\frac{M^{n+1}-M^n}{\Delta t} + \theta(\partial_x\left(\frac{M^2}{\rho}\right) + \partial_x P) + (1-\theta)(\partial_x\left(\frac{M^2}{\rho}\right) + \partial_x P) = \theta S_M(\rho,M,E)\rfloor^{n+1} + (1-\theta)S_M(\rho,M,E)\rfloor^n \tag{22}$$

$r_E(n+1)$:

$$\frac{E^{n+1}-E^n}{\Delta t} + \theta\partial_x\left(v(E+P)\right) + (1-\theta)\partial_x\left(v(E+P)\right) = \theta S_E(\rho,M,E)\rfloor^{n+1} + (1-\theta)S_E(\rho,M,E)\rfloor^n \tag{23}$$

where S_C, S_M and S_E are the continuity, momentum and energy source terms arising from the forcing functions of the manufactured solution (MMS1).

Equations (21)-(23) are the nonlinear residual functions about the point $(n+1)$.

Let us choose a linearization point (*) which typically is the last Newton iteration, about which a change in the state variable can be defined. This can then be given as

$$\delta\rho = \rho^{n+1} - \rho^*$$
$$\delta M = M^{n+1} - M^*$$
$$\delta E = E^{n+1} - E^*$$

Substituting these new variables in (21)-(23), the following conservations equations for the delta form of the state variables are obtained.

$$\frac{\delta\rho}{\Delta t} + \theta\partial_x\left(\delta M\right) = -r_C(*) \tag{24}$$

$$\frac{\delta M}{\Delta t} + \theta\partial_x\delta P = -r_M(*) \tag{25}$$

$$\frac{\delta E}{\Delta t} + \theta\partial_x\left(\delta M\left(\frac{E+P}{\rho}\right)^*\right) = -r_E(*) \tag{26}$$

Notice that the advection terms in the Momentum and Energy conservation equations have been linearized about the point (*). Also notice that the source terms S_C, S_M and S_E have been linearized about (*) and is part of the residual function $r(*)$ for each conservation equation.

Rearranging the Momentum equation above, we get

$$\delta M = -\theta \Delta t \partial_x \delta P - \Delta t r_M(*) \tag{27}$$

This expression can then be substituted in the continuity and the energy equation to obtain a system of equations in $\delta\rho, \delta E$.

$$\delta\rho = -\theta \Delta t \partial_x \left(\delta M\right) - \Delta t r_C(*) \tag{28}$$

$$\delta E = -\theta \Delta t \partial_x \left(\delta M \left(\frac{E+P}{\rho}\right)^*\right) - \Delta t r_E(*) \tag{29}$$

Now using the linearized EOS introduced in Equation (17), we can then substitute

$$\delta\rho = \frac{1}{\frac{\partial P}{\partial \rho}} \left(\delta P - \frac{\partial P}{\partial E}\delta E\right) \tag{30}$$

Substituting the above equation in (28), we get

$$\frac{\partial P}{\partial E}\delta E = \delta P + \frac{\partial P}{\partial \rho}\left(\theta \Delta t \partial_x \left(\delta M\right) + \Delta t r_C(*)\right) \tag{31}$$

Rearranging the above equation and substituting (29) for δE, we get the semi-discrete form of the pressure Poisson equation, given as

$$\begin{aligned}\delta P = \frac{\partial P}{\partial E}\left(-\theta \Delta t \partial_x \left(\delta M \left(\frac{E+P}{\rho}\right)^*\right) - \Delta t r_E(*)\right) \\ - \frac{\partial P}{\partial \rho}\left(\theta \Delta t \partial_x \left(\delta M\right) + \Delta t r_C(*)\right)\end{aligned} \tag{32}$$

where δM can be obtained from (27). Equation (32) is the equivalent representation of the ICE system with linearized advection terms in momentum and energy equations while treating the pressure waves implicitly. Since the pressure waves are resolved, an ICE solve results in eliminating all the dominant eigenmodes occurring due to the pressure wave, i.e., sound speed in the medium. Hence, the resulting system has a smaller spectral radius, especially for low mach flows where the spread between the eigenvalues in the original nonlinear Euler system is the quite large.

This system shown in (32) can be solved for δP and back-substituted to obtain δM from (27), δE from (29), $\delta\rho$ from (28) respectively. It is important to note that the new system expressed as an elliptic pressure equation is exactly same as the original semi-discrete ICE linearized Euler equations. It is quite clear that by doing the algebraic manipulation shown in (30) for EOS and substitution of (27) into the continuity and energy equations, an equivalent Gaussian elimination on a system of size $4N$ has $(\delta\rho, \delta E, \delta M, \delta P)$ been performed analytically to convert it to an block upper triangular form that can be solved by back substitution. Hence it is important to note that solving the original ICE system and the elliptic pressure equation yield the exact same result as long as the spatial discretization of the PDE's are consistent in both cases.

The gain in computational time when using ICE as a preconditioner and as a solver by itself will be discussed in the following sections. Now let us consider the advantages and disadvantages of the ICE preconditioner of size N introduced earlier (denoted hereafter as N-ICE).

Pros

1. The elliptic pressure matrix in the fully discrete form is clearly only $N \times N$ while the original ICE system we started with was a $3N \times 3N$ hyperbolic system. The gain in terms of reduction in the size of the system without any approximation and loss of accuracy makes this method very valuable.

2. Cost savings in terms of forming and solving the modified system is significantly lower leading to decrease in memory and CPU requirements per calculation.

3. Advantages of linearized ICE equations are used in multi-dimensional, two-phase flows systems involving at least a linear systems of size $9N$, is quite obvious. Even in this 9-equation case, a single pressure equation can be obtained by performing a similar procedure as above to obtain a matrix of only size N which can be computed and solved efficiently.

Cons

1. It is difficult to maintain the consistency of the fully discrete ICE system w.r.t. the original discretization of the nonlinear system due to the evaluation of derivatives of residuals about the linearized point in computing the right hand side. Care is needed if a consistent preconditioner is to be created from the N-ICE system.

4.2.6 Preconditioner Efficiency

To evaluate the efficiency of preconditioners for the nonlinear Euler equations, we will consider 3 different kinds of preconditioning techniques. They are

1. Consistent-Block-Jacobi (CBJ): This technique is essentially a point Jacobi preconditioner per physic block and includes the coupling between the state variables for each node. Hence, it can be considered to be a spatially inconsistent physics-based preconditioning technique that treats all quantities implicitly and neglects the node-to-node spatial coupling of the state variables.

2. Consistent N-ICE: This is the physics preconditioner introduced earlier where a system of size N is solved to eliminate the modes arising from pressure waves

3. Inconsistent N-ICE: This is essentially the ICE preconditioner but is inconsistent in temporal discretization. For example, if C.N is used to solve the problem, the preconditioner will utilize a lower order method in time, say B.E by solving the size N system as before.

The motivation behind the introduction of inconsistent preconditioners (1) and (3), is to find out the effect of consistency in preconditioning techniques. Similar to the nonlinear diffusion problem, the preconditioner efficiency can then be measured and we can look at how each preconditioner reduces the total iterations as a function of the Mach number (Ma). Since Ma determines how close the velocity of flow is, to the speed of sound in the medium, it is critical to look at the the best preconditioner for low Mach flows like in the case of flow in nuclear reactors where our primary motivations lie.

The efficiency of each of these preconditioners was measured using an adaptive time stepping scheme based on the dynamical time scale for the problem using MMS1 forcing functions discretized into 300 finite volumes with C.D.S spatial and C.N temporal discretization.

We can see a uniform trend from the plot in Fig. 10 that as the Mach number decreases, the total number of GMRES iterations required to converge the nonlinear problem to a given accuracy, increases. The important observation from this plot is that preconditioners with consistent temporal discretization (1) and

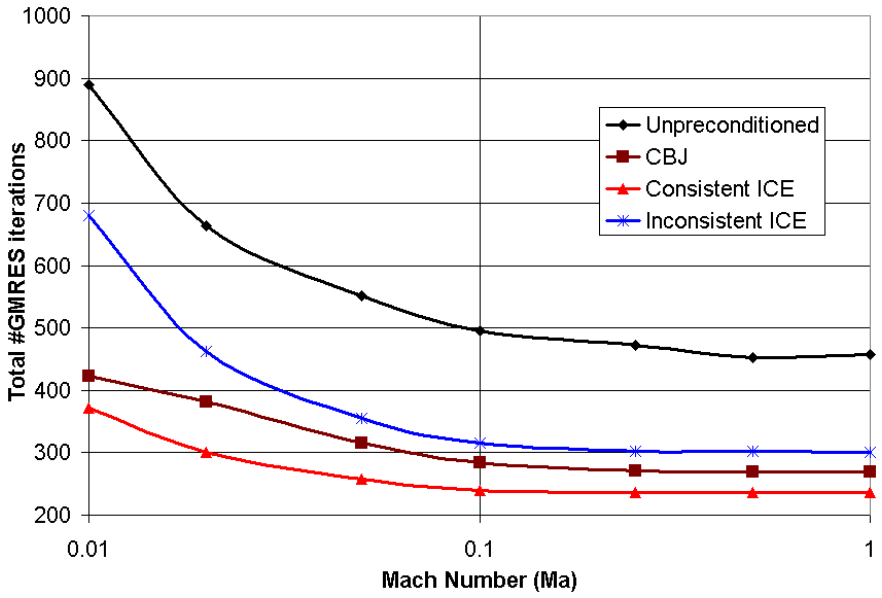


Figure 10: Effect of Mach number (Ma) on number of total linear iterations to show increase in stiffness as a function of Ma

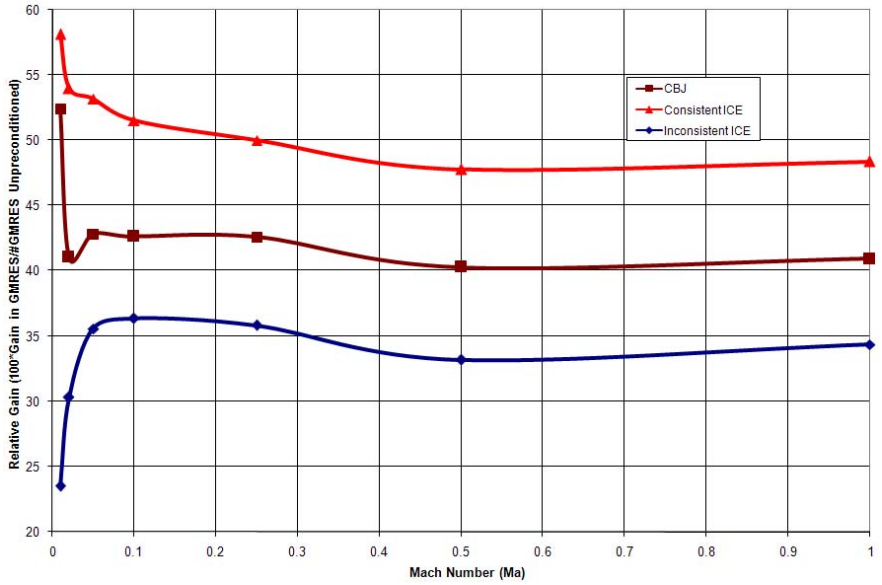


Figure 11: Preconditioner efficiency for Euler equations using different techniques: CBJ, $Consistent\ N-ICE$, $Inconsistent\ N-ICE$

(2) perform comparably much better in the low Mach limit as long as good time adaptive techniques are used to resolve the behavior of the transient.

Fig. 11 shows the percentage gain from different kinds of preconditioners wrt the Mach number of the problem. The formula for the gain is given as

$$\%Gain = 100\left(1 - \frac{\#Total\ Preconditioned\ GMRES\ Iterations}{\#Total\ UnPreconditioned\ GMRES\ Iterations}\right) \tag{33}$$

From the preconditioner performance results, it is quite clear that the efficiency of the ICE preconditioner goes up as Mach number reduces significantly from 1.0, the sonic limit. The Consistent-Block-Jacobi preconditioner provides around 45% gain in the total number of GMRES iterations while the consistent ICE provides close to 60% for low Mach flows.

Although the formation of the CBJ preconditioner takes only 3 perturbations/node (one per state variable), the bandwidth, even in a 1-D problem, is $2N$ and hence the linear solve for this preconditioner tends to be quite expensive. On the other hand, the ICE preconditioner solves a size N system that is consistent with the original spatial and temporal discretization and has a bandwidth of at most 5 for CDS. Hence, in terms of cost and efficiency, the ICE preconditioner yields considerable improvement over the other types of preconditioned systems for low Mach flow. Also, the temporally inconsistent ICE preconditioner provides on average 30% gain for low Mach flows which is only half as much from a consistent ICE preconditioner. Hence from the results, one can conclude that consistency in spatial and temporal discretization might be the key for the success of ICE preconditioner, especially for low Mach flows. Since all results presented here are for a simple 1-D problem, further studies relating to efficiency with respect to consistency on higher dimensional problems might be necessary to determine superiority of consistent $N - ICE$ preconditioner.

Previously, the effect of Mach number on the preconditioner efficiency was discussed but a critical observation regarding efficiency with respect to time step size could not be ascertained. For this purpose, for Ma=0.05, an adaptive transient based on 1/100th of dynamical time scale of the problem was simulated and the minimum dynamical time scale was measured. Also several constant time stepping transients using different physics based techniques as a solver with Picard iterations for nonlinear resolution was tried. The results from these runs are plotted in Fig. 12.

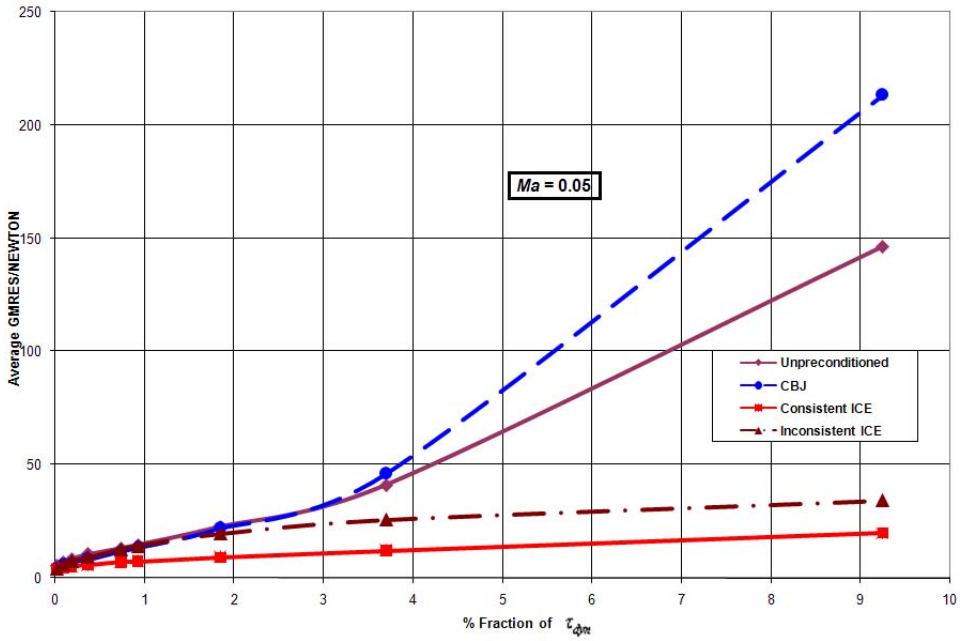


Figure 12: Effect of Δt on efficiency of different preconditioners for Ma=0.05

Fig. 12 shows the total #GMRES iterations as a function of Δt utilized. Since the dynamical time scale is not constant throughout the transient, the average value is used in calculating %Fraction of τ_{dyn}. It is obvious that the spatially inconsistent CBJ scheme fails to be a good preconditioner and performs worse than an unpreconditioned solver for bigger time steps i.e., it pushes the solver away from the true solution by amplifying modes neglected by the preconditioner. On the other hand, the ICE schemes, consistent and inconsistent temporally provide an average gain by a factor of 4 in terms of reducing the total iterations. The performance of the temporally inconsistent ICE comparable to the consistent ICE preconditioner is interesting for bigger time steps but this might be due to the fact that B.E scheme used in this preconditioner is probably as accurate as C.N scheme in the non-asymptotic regime. Further analysis on the behavior of this inconsistent ICE preconditioner can be performed for different Mach numbers and using, say, an explicit temporal discretization. Such discussions are not included in this report and can be simulated in the future to gain intuitive understanding of inconsistent preconditioning techniques.

5 Conclusions

The work pursued at Texas A&M University and at INL during the summer internship, and, the insight gained into the analysis of JFNK techniques, nonlinear diffusion physics, fluid flow physics, and flexible software engineering concepts have been quite enlightening and will be fruitful in understanding coupled nonlinear multi-physics phenomena better. Few observations noted on different tasks that were worked on during this contract duration are given below.

5.1 JFNK

1. JFNK framework is quite efficient at tackling even highly nonlinear problems involving diffusion, reaction and advection physics;
2. JFNK framework dictates minimal requirements from a coding perspective and only 3 main routines are necessary to implement a consistent technique in place
   (a) Calculating the nonlinear residual,
   (b) Action of Jacobian on a vector (Jv) and,
   (c) Action of the Preconditioner on a vector ($P^{-1}v$);
3. In the future, implementation using the PETSc library is envisioned; the nonlinear solver would then handle using PETSc, for which a better part of the existing framework exists and can be integrated easily with the PETSc/SNES solver.

5.2 Physics-Based Preconditioning

1. Efficient physics-based preconditioners can significantly reduce the cost in linear iteration;
2. Simplified physics models that linearize the weakest physics can yield a better guess to the correct solution than using purely numerical approaches like using a Block-Jacobi solver;
3. For nonlinear fluid flow problems, especially with low Mach numbers, N-ICE preconditioning (ICE system of size N) yields considerable reduction in cost with a gain of almost 55% in the number of iterations and by reducing a system of size $3N$ to a single elliptic pressure matrix of size N;
4. Whenever possible, preconditioners that are consistent with the original nonlinear discretization should be used since they provide the flexibility to use Picard iterations and have proved to be good candidate all the simulations tested for the 1-D problem; further studies on higher dimensions will be helpful in confirming these preliminary conclusions.
5. It should be remembered that most inconsistent preconditioners as solvers do converge in the fine time step size limit and as long as stability limits introduced due to the approximation are respected, they might provide cheap alternatives to solve the problem with desired accuracy.

6 Future Work

Current nonlinear multi-physics coupling techniques involve heavily in using an Operator-Split approach rather than Implicit treatment of the coupling terms. Based on the new techniques learnt in the context of solution for highly nonlinear problems, a list of intended future work topics are listed below.

1. Experiments with different state variables (non conservative form) to compare condition number of JP^{-1} to the current state variable choice, with ICE as preconditioner for low Mach flows should yield interesting results. Since the focus of the current research is in the realm of nuclear reactor analysis, low Mach flows are most likely to occur and hence, an optimal set of state variables suitable for this purpose can be determined for this purpose.
2. Techniques to adaptively choose consistent or inconsistent preconditioners in order to increase computational savings can be implemented based on the time step size selected. Such methods require good understanding of the approximations that go into these preconditioners and the limits at which these approximations are reasonably accurate.

3. The nonlinear diffusion and fluid flow equations can be coupled together to simulate a complex multi-physics problem involving several different length and time scales. Such efforts can improve the intuition and understanding in creating efficient physics-based preconditioners to reduce computational time without sacrificing accuracy of the solution.

4. Analysis on the difference between dynamical time scale based adaptive schemes and LTE based schemes need to be performed in order to determine the optimal hybrid adaptive scheme that respects the different time scales and numerical error in the solution.

5. Lessons learnt from (1)-(4) need to be used in the future to create a fully implicit, coupled reactor physics code that can efficiently handle nonlinear multi-physics transient problems arising in nuclear reactor safety and analysis studies.

A brief description of the proposed code mentioned in (5) will be covered in the next section.

7 KARMA: C(K)ode for Accident and Reactor Modeling Analysis

KARMA is a fully implicit, coupled, multi-physics transient analysis code that can be used to analyze reactor accidents and modeling problems. The plug-in architecture employed in KARMA will make it easy to adapt it to couple several physics components and the framework can be used to seamlessly integrate the existing consistent numerics models with the new physics models that can be created.

7.1 Why another code ?

Conventional coupling paradigms currently used to couple different physics components in reactor analysis problems can be inconsistent in their treatment of the nonlinear terms due to the naïve coupling and time stepping strategies. This leads to usage of small time steps to maintain stability and accuracy requirements thereby increasing the overall computational time. These inconsistencies can be overcome by using improved time approximations for the nonlinear operator to regain the lost accuracy and restore the consistency of conventional schemes.

Traditional multi-physics coupled codes for reactor analysis problems often employ validated and verified, efficient mono-physics codes with either PVM or MPI architecture to achieve the loose coupling with an Operator Split (OS) methodology. Such "divide and conquer" methods provide flexibility in the usage of standard industrial codes and avoid replicating man years of development and testing. Examples of such existing coupled codes to analyze reactor transients are PARCS/TRACE and NESTLE/RELAP where an explicit coupling of the physics codes is performed by exchanging the solution fields from each physics at every time step as only boundary condition to the other physics. Such methods have been proven [1] to be only $O(h)$ implying the need for considerably small time steps to obtain an accurate solution.

Past research on the effect of the OS methods in terms of accuracy as compared to fully implicit coupling methods have been analyzed and documented [10]. Mousseau et al. demonstrated that coupled benchmark problems in unsteady radiation diffusion using a consistent and accurate numerical scheme based on Jacobian-Free Newton Krylov (JFNK) framework is more effective to resolve the coupling than traditional OS methods. Such a scheme will then preserve the higher orders of accuracy in time integration in each nonlinear physics and the fully coupled solution. Also, since the length and time scales vary by orders of magnitude in such problems, an adaptive methodology in space and time is required to efficiently resolve the solution evolution.

The current design of KARMA was put forth in order to satisfy the above necessities and to produce a strongly coupled, implicit framework to handle multi-physics, multi-scale problems in general. In the process of rigourous testing, validation and verification of KARMA, accidents and reactor analysis problems will be tested against benchmarked results to prove the increased efficiency and accuracy from the new effort.

7.2 The KARMA Framework

KARMA will be entirely written in C++ making use of the advanced features, such as abstraction, encapsulation and inheritance, to create loosely coupled objects that allow seamless integration of new physics models as and when created. A prime concern then would be to achieve high level of efficiency while still maintaining the object oriented philosophy in mind. Careful planning in this aspect of the computational domain was done and a decision to use well tested parallel data structures provided by libraries like PETSc [11] was made. This also reduces the overhead in designing, implementing, well tested parallel communication and linear algebra routines and thereby eliminating the possibility of bugs in these basic operations.

A broad look at how the KARMA framework works with the different components is shown in Fig. 13.

The overview plot shown in Fig. 13 clearly indicates the global interaction of KARMA library with an User Interface (UI) which is the primary driver interacting with the end user i.e., in this case, a reactor designer or analyst. KARMA library itself makes use of the PETSc parallel library for the linear algebra data structures such as Matrix, Vectors, Distributed arrays and several PETSc modules that handle linear solvers, nonlinear solvers, numerical preconditioners and optionally even temporal integration. The details on the capability and functionality of PETSc used in KARMA will not be covered in this report.

A more comprehensive look in to the different packages and components in KARMA will be covered in the next section.

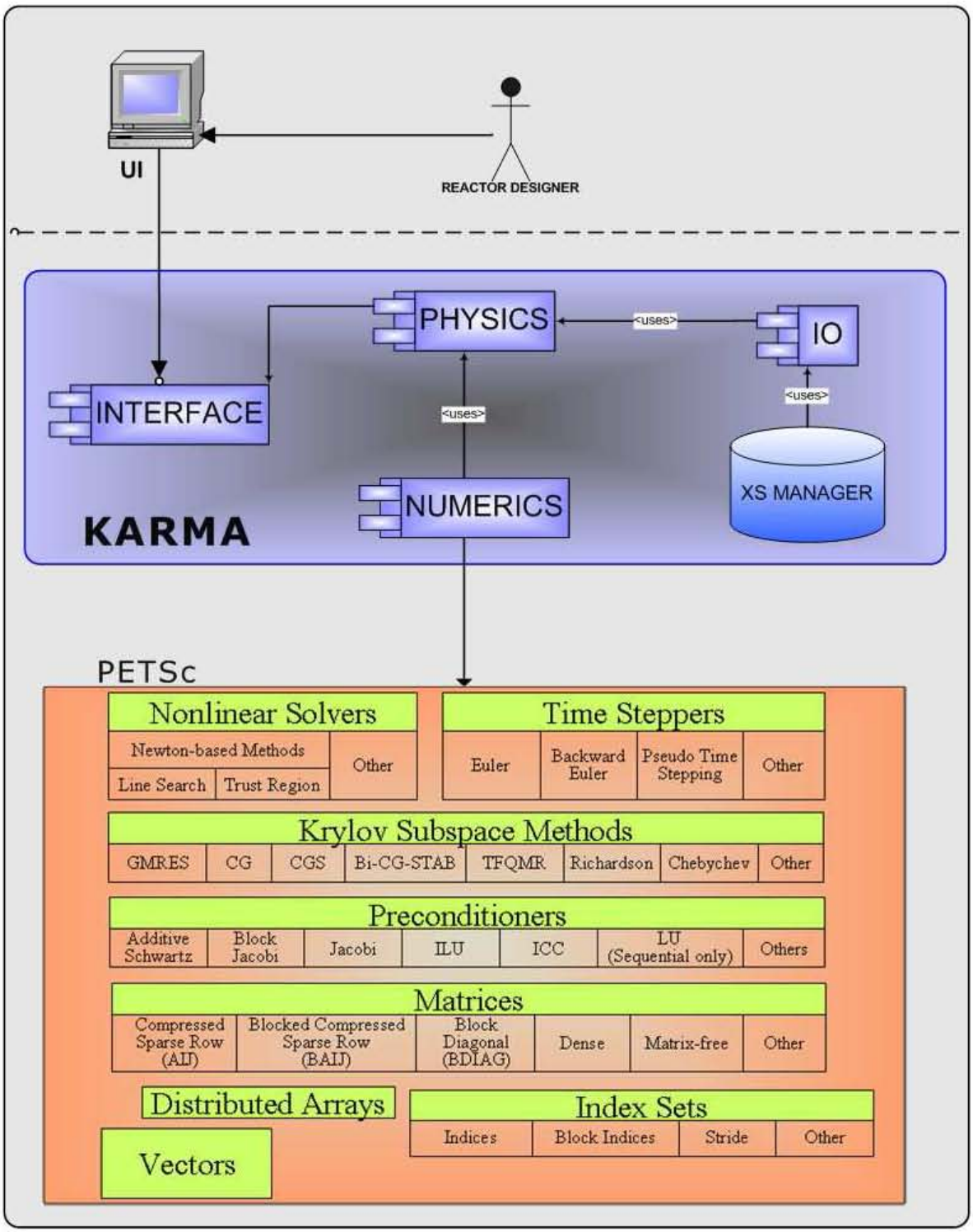


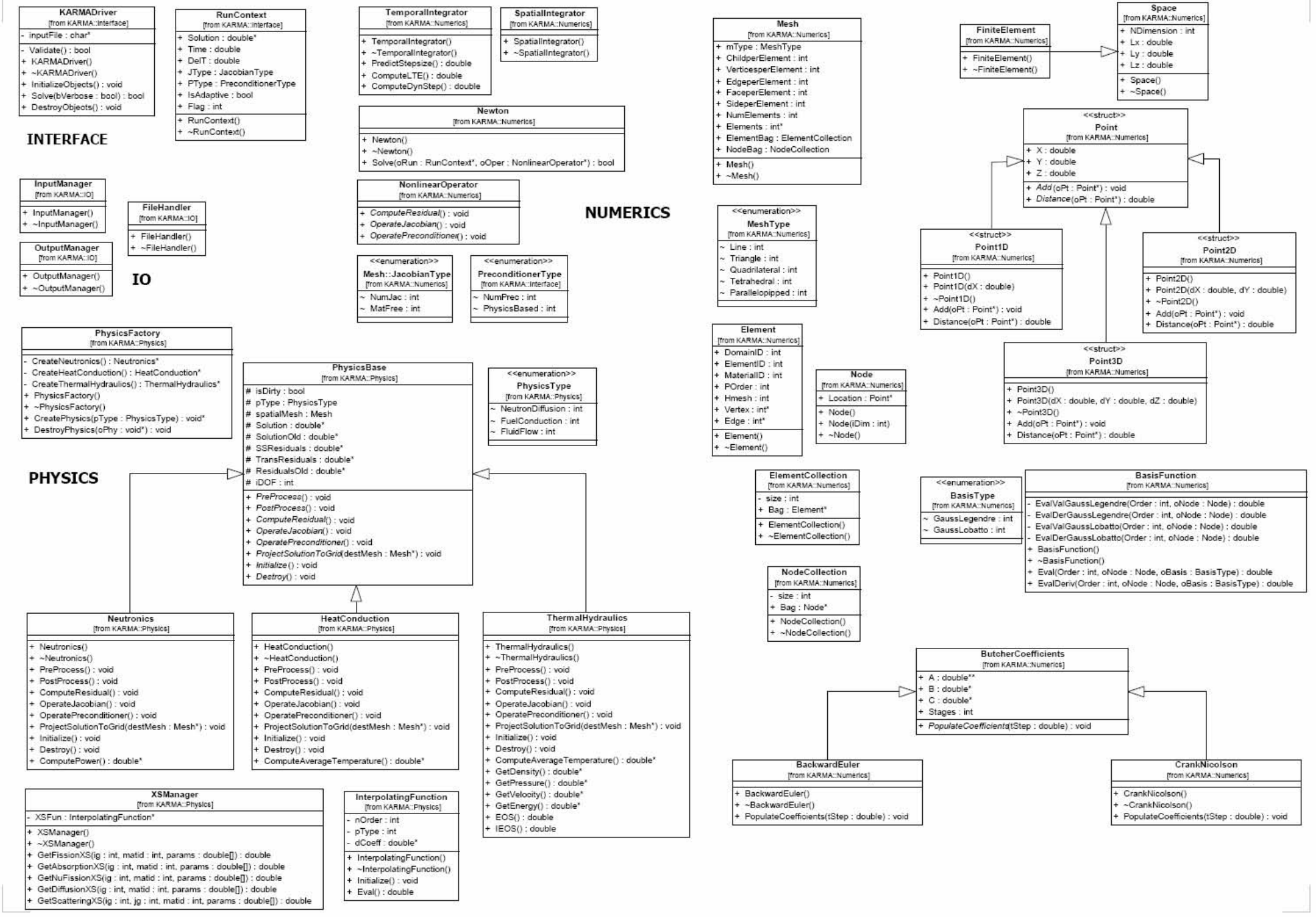
Figure 13: Overall KARMA framework diagram showing interaction of different packages with user and external libraries

7.3 KARMA code structure

A descriptive way of showing how the different classes work together and the relation between different objects can be seen clearly from a class diagram. The KARMA class diagram showing these relations clearly is shown in Fig. 14. It should be noted that the test framework to test individual routines is not shown here and that the existing class structures are tentative and are only instructional on the final data structure representation.

A brief look at each of the packages, the functions, the objects they contain are described in the following section.

Fig. 14 (next page): Class diagram for KARMA showing data structures to be used in different packages.


KARMADriver
[from KARMA::Interface]
- inputFile : char*
- Validate() : bool
+ KARMADriver()
+ ~KARMADriver()
+ InitializeObjects() : void
+ Solve(bVerbose : bool) : bool
+ DestroyObjects() : void
RunContext
[from KARMA::Interface]
+ Solution : double*
+ Time : double
+ DelT : double
+ JType : JacobianType
+ PType : PreconditionerType
+ IsAdaptive : bool
+ Flag : int
+ RunContext()
+ ~RunContext()
INTERFACE
InputManager
[from KARMA::IO]
+ InputManager()
+ ~InputManager()
FileHandler
[from KARMA::IO]
+ FileHandler()
+ ~FileHandler()
OutputManager
[from KARMA::IO]
+ OutputManager()
+ ~OutputManager()
IO
TemporalIntegrator
[from KARMA::Numerics]
+ TemporalIntegrator()
+ ~TemporalIntegrator()
+ PredictStepsize() : double
+ ComputeLTE() : double
+ ComputeDynStep() : double
SpatialIntegrator
[from KARMA::Numerics]
+ SpatialIntegrator()
+ ~SpatialIntegrator()
Newton
[from KARMA::Numerics]
+ Newton()
+ ~Newton()
+ Solve(oRun : RunContext*, oOper : NonlinearOperator*) : bool
NonlinearOperator
[from KARMA::Numerics]
+ ComputeResidual() : void
+ OperateJacobian() : void
+ OperatePreconditioner() : void
NUMERICS
<<enumeration>>
Mesh::JacobianType
[from KARMA::Numerics]
~ NumJac : int
~ MatFree : int
<<enumeration>>
PreconditionerType
[from KARMA::Interface]
~ NumPrec : int
~ PhysicsBased : int
PhysicsFactory
[from KARMA::Physics]
- CreateNeutronics() : Neutronics*
- CreateHeatConduction() : HeatConduction*
- CreateThermalHydraulics() : ThermalHydraulics*
+ PhysicsFactory()
+ ~PhysicsFactory()
+ CreatePhysics(pType : PhysicsType) : void*
+ DestroyPhysics(oPhy : void*) : void
PHYSICS
PhysicsBase
[from KARMA::Physics]
isDirty : bool
pType : PhysicsType
spatialMesh : Mesh
Solution : double*
SolutionOld : double*
SSResiduals : double*
TransResiduals : double*
ResidualsOld : double*
iDOF : int
+ PreProcess() : void
+ PostProcess() : void
+ ComputeResidual() : void
+ OperateJacobian() : void
+ OperatePreconditioner() : void
+ ProjectSolutionToGrid(destMesh : Mesh*) : void
+ Initialize() : void
+ Destroy() : void
<<enumeration>>
PhysicsType
[from KARMA::Physics]
~ NeutronDiffusion : int
~ FuelConduction : int
~ FluidFlow : int
Neutronics
[from KARMA::Physics]
+ Neutronics()
+ ~Neutronics()
+ PreProcess() : void
+ PostProcess() : void
+ ComputeResidual() : void
+ OperateJacobian() : void
+ OperatePreconditioner() : void
+ ProjectSolutionToGrid(destMesh : Mesh*) : void
+ Initialize() : void
+ Destroy() : void
+ ComputePower() : double*
HeatConduction
[from KARMA::Physics]
+ HeatConduction()
+ ~HeatConduction()
+ PreProcess() : void
+ PostProcess() : void
+ ComputeResidual() : void
+ OperateJacobian() : void
+ OperatePreconditioner() : void
+ ProjectSolutionToGrid(destMesh : Mesh*) : void
+ Initialize() : void
+ Destroy() : void
+ ComputeAverageTemperature() : double*
ThermalHydraulics
[from KARMA::Physics]
+ ThermalHydraulics()
+ ~ThermalHydraulics()
+ PreProcess() : void
+ PostProcess() : void
+ ComputeResidual() : void
+ OperateJacobian() : void
+ OperatePreconditioner() : void
+ ProjectSolutionToGrid(destMesh : Mesh*) : void
+ Initialize() : void
+ Destroy() : void
+ ComputeAverageTemperature() : double*
+ GetDensity() : double*
+ GetPressure() : double*
+ GetVelocity() : double*
+ GetEnergy() : double*
+ EOS() : double
+ IEOS() : double
XSManager
[from KARMA::Physics]
- XSFun : InterpolatingFunction*
+ XSManager()
+ ~XSManager()
+ GetFissionXS(ig : int, matid : int, params : double[]) : double
+ GetAbsorptionXS(ig : int, matid : int, params : double[]) : double
+ GetNuFissionXS(ig : int, matid : int, params : double[]) : double
+ GetDiffusionXS(ig : int, matid : int, params : double[]) : double
+ GetScatteringXS(ig : int, jg : int, matid : int, params : double[]) : double
InterpolatingFunction
[from KARMA::Physics]
- nOrder : int
- pType : int
- dCoeff : double*
+ InterpolatingFunction()
+ ~InterpolatingFunction()
+ Initialize() : void
+ Eval() : double
Mesh
[from KARMA::Numerics]
+ mType : MeshType
+ ChildperElement : int
+ VertioesperElement : int
+ EdgeperElement : int
+ FaceperElement : int
+ SideperElement : int
+ NumElements : int
+ Elements : int*
+ ElementBag : ElementCollection
+ NodeBag : NodeCollection
+ Mesh()
+ ~Mesh()
<<enumeration>>
MeshType
[from KARMA::Numerics]
~ Line : int
~ Triangle : int
~ Quadrilateral : int
~ Tetrahedral : int
~ Parallelopipped : int
Element
[from KARMA::Numerics]
+ DomainID : int
+ ElementID : int
+ MaterialID : int
+ POrder : int
+ Hmesh : int
+ Vertex : int*
+ Edge : int*
+ Element()
+ ~Element()
Node
[from KARMA::Numerics]
+ Location : Point*
+ Node()
+ Node(iDim : int)
+ ~Node()
ElementCollection
[from KARMA::Numerics]
- size : int
+ Bag : Element*
+ ElementCollection()
+ ~ElementCollection()
NodeCollection
[from KARMA::Numerics]
- size : int
+ Bag : Node*
+ NodeCollection()
+ ~NodeCollection()
FiniteElement
[from KARMA::Numerics]
+ FiniteElement()
+ ~FiniteElement()
Space
[from KARMA::Numerics]
+ NDimension : int
+ Lx : double
+ Ly : double
+ Lz : double
+ Space()
+ ~Space()
<<struct>>
Point
[from KARMA::Numerics]
+ X : double
+ Y : double
+ Z : double
+ Add(oPt : Point*) : void
+ Distance(oPt : Point*) : double
<<struct>>
Point1D
[from KARMA::Numerics]
+ Point1D()
+ Point1D(dX : double)
+ ~Point1D()
+ Add(oPt : Point*) : void
+ Distance(oPt : Point*) : double
<<struct>>
Point2D
[from KARMA::Numerics]
+ Point2D()
+ Point2D(dX : double, dY : double)
+ ~Point2D()
+ Add(oPt : Point*) : void
+ Distance(oPt : Point*) : double
<<struct>>
Point3D
[from KARMA::Numerics]
+ Point3D()
+ Point3D(dX : double, dY : double, dZ : double)
+ ~Point3D()
+ Add(oPt : Point*) : void
+ Distance(oPt : Point*) : double
<<enumeration>>
BasisType
[from KARMA::Numerics]
~ GaussLegendre : int
~ GaussLobatto : int
BasisFunction
[from KARMA::Numerics]
- EvalValGaussLegendre(Order : int, oNode : Node) : double
- EvalDerGaussLegendre(Order : int, oNode : Node) : double
- EvalValGaussLobatto(Order : int, oNode : Node) : double
- EvalDerGaussLobatto(Order : int, oNode : Node) : double
+ BasisFunction()
+ ~BasisFunction()
+ Eval(Order : int, oNode : Node, oBasis : BasisType) : double
+ EvalDeriv(Order : int, oNode : Node, oBasis : BasisType) : double
ButcherCoefficients
[from KARMA::Numerics]
+ A : double**
+ B : double*
+ C : double*
+ Stages : int
+ PopulateCoefficients(tStep : double) : void
BackwardEuler
[from KARMA::Numerics]
+ BackwardEuler()
+ ~BackwardEuler()
+ PopulateCoefficients(tStep : double) : void
CrankNicolson
[from KARMA::Numerics]
+ CrankNicolson()
+ ~CrankNicolson()
+ PopulateCoefficients(tStep : double) : void

7.4 Class list

A list of all the classes currently shown in the class diagram of Fig. 14 along with a very brief description about the function of the class is given in the following sections.

7.4.1 INTERFACE

The INTERFACE package decouples other packages in KARMA from the outside world. This provides the flexibility to an end user that whether a 3-D neutron diffusion model is used or a simple 0-D point kinetics model is used, the way the library interacts with the UI would not change. The INTERFACE will also provide specific drivers to simulate reactor accident scenarios and transients that can be optimized to choose specific physics models to improve computation time.

7.4.1.1 KARMADriver Primary interface driver for all KARMA routines ; Run parameters and numerical method specifics are bubbled down from the user to KARMA by the driver.

7.4.1.2 RunContext This class holds the current context of execution; It is important in a loosely coupled, plug-in architecture, for parts of the code to know the dependencies, time level, spatial discretization details and other run params in order to integrate with different parts of the code seamlessly and be consistent in handling their functions.

7.4.2 IO

The geometry, run parameters and user inputs will be supplied to KARMA through a XML file which has the advantages of being user and machine readable. The IO package also includes a cross-section, material properties handler which abstracts the physics models from the details whether a table is used to interpolate or if a user defined closed form function is used to obtain the information.

7.4.2.1 FileHandler This class takes care of the input and output stream functionality to handle files. It does not really understand the format of the file but remembers the reference counts so that there are no unclosed files at end of execution.

7.4.2.2 InputManager This class will use the FileHandler to read the input deck file and supply the INTERFACE module with the relevant information. This can be visualized as being a parser that can understand the specific format of the file it is required to read. For example, input parameters could be specified in a XML file in which case, InputManager contains information to locate required data for execution.

7.4.2.3 OutputManager This class will also use the FileHandler to write user specified data into specific formats such as VTK or mat files in order to aid the end user in visualization of the data generated. This procedure can be used to create real-time monitoring of solution evolution in a transient by communicating with software like VISIT or ParaView.

7.4.3 PHYSICS

The PHYSICS package contains the different models used to describe the physics in terms of nonlinear PDE's. The objects in the package are dedicated to specific physics namely Neutronics, Heat conduction and Thermalhydraulic fluid flow. Several simple and detailed models for each physics can be created and based on the problem being solved, a decision to use a specific physic model can be made. The PDE

description is translated into a numerical form by means of discretization in both space and time and these objects are used from the NUMERICS package.

7.4.3.1 PhysicsFactory The Factory pattern used here has PhysicsFactory class at its root. This class handles the initialization and destruction of Physics models. Any problem which needs a certain combination of Physics models will have to request the factory and validations on the compatibility of the models will be performed before the model is handed back to the requestor.

7.4.3.2 PhysicsBase This is the base class for all Physics models. All Physics specific classes should inherit from this class and implement the virtual methods. Also this class contains information about the Numerical mesh in space and time being used to solve this specific physics.

7.4.3.3 Neutronics This class handles the neutron transport behavior in the reactor core. Based on the number of energy and delayed groups, dynamically the PDEs to be solved will be changed. A class can be derived from this base class to create specific approximate models such as 3D Multi-group diffusion (MGD) or 2D neutron transport or the such. In the current implementation, only a 3D MGD model will be available to describe this physics. Parameterized cross-sections will be obtained from the IO::XSManager based on the supplied fuel temperature, moderator density and other state variables.

7.4.3.4 HeatConduction This class describes the nonlinear heat conduction in the fuel. Since this physics is strongly nonlinear (conductivity is a dependent function of the temperature), a newton based iteration can be used to solve this single physics efficiently. The power density and the fluid flow state variables are supplied from the other physics.

7.4.3.5 Thermalhydraulics This base class describes the fluid flow model in the reactor core. Like Neutronics model, it will serve as the base for say a 1D, single-phase model or a 1D two-phase model or maybe even an advanced model to capture the cross flow mixing in the core and the turbulent phenomena. Typically, for each phase, the Thermalhydraulic model will contain the mass, momentum and energy conservation equations and some kind of closure models to make the PDE system consistent.

7.4.3.6 XSManager This class handles the generation of cross-section data based on parameterized data supplied to it. Parameterized data might include fuel temperature, moderator density, moderator temperature, boron concentration, burnup etc.

7.4.4 NUMERICS

The NUMERICS package handles all the spatial and temporal discretization routines and objects. These objects describe the spatial domain, the geometry of the system, the discrete formulation of the problem and time adaptation. Since the NUMERICS package involves the usage of several low level data structures, this will also be the primary package handling PETSc structures such as Vectors and Matrices to store the values efficiently.

7.4.4.1 TemporalIntegrator Primary class that handles how the time step is chosen based on different criteria ; Options to use certain types of adaptive time stepping taking into account also the dynamical time scale of the system will be available. This class will implement routines to handle the actual nonlinear newton solve over the different physics models for each stage as given in a Butcher Tableau for the chosen method of time integration.

7.4.4.2 SpatialIntegrator Driver class that has the knowledge to use a spatially discretize domain information to obtain the required basis functions, transformation from reference to local and several other routines to aid in constructing the spatial matrix that arise from discretization.

7.4.4.3 Newton This is the main class that handles the nonlinear Newton iteration. It requires the nonlinear object to implement or contain the NonlinearOperator object so that the newton iteration procedure is completely independent of the type of object being solved for.

7.4.4.4 NonlinearOperator This class will contain virtual methods that need to be implemented by each of the nonlinear physics models. The Newton class will look for the implementation of the NonlinearOperator by an object and call those to perform the actual Newton iteration procedure. This class implements the following routine: ComputeResidual, OperateJacobian, OperatePreconditioner. It is easy to see how this fits in the Jacobian Free Newton Krylov framework to compute the solution of a nonlinear problem using matrix free technique

7.4.4.5 InterpolatingFunction This is more of a utility class that can interpolate a given set of data and find the best polynomial fit by Least Squares with minimum residual. This can be a multi-dimensional fit (more than 1 state variable) and will primarily be used by the XSManager class to obtain interpolated cross-section data by a table look-up. Alternately, this class can be assigned the fit coefficients and be used to obtain the interpolant based on that.

7.4.4.6 Point This is a simple abstract base class that defines a point in space.

7.4.4.7 Point1D This class derives from Point and defines a point in 1-D space. (x)

7.4.4.8 Point2D This class derives from Point and defines a point in 2-D space. (x, y)

7.4.4.9 Point3D This class derives from Point and defines a point in 3-D space. (x, y, z)

7.4.4.10 Space A generic object that describes the spatial domain being solved. Space contains the actual domain information and is unaware of the discretization being used to solve the problem.

7.4.4.11 FiniteElement This class derives from Space and discretizes the domain into several Elements. This is also a driver class that has the knowledge to discretize a given domain into discretized space based on FEM discretization, for a fixed step size (h) or adaptivity in space, for a fixed basis polynomial order (p) or variable order.

7.4.4.12 Mesh This is the final discretized description of a given domain. Each of the discretization method will contain at least one Mesh object and the solution or the degrees of freedom will be calculated at the Nodes as defined by this object.

7.4.4.13 Element This is a local Element object that contains information on the different vertices, faces, edges and some attributes specifying if it is a boundary element.

7.4.4.14 Node These are essentially objects that contain information about the degree of freedom and correspond to discrete points in space for a given dimension.

7.4.4.15 ElementCollection A collection of several Element objects.

7.4.4.16 NodeCollection A collection of several Node objects.

7.4.4.17 BasisFunction A BasisFunction object that has the knowledge to evaluate value and derivative for a given basis function type of say order p, at a particular point. The FiniteElement assembly procedure will call this object repeatedly to find the local contribution to a node from all the other nodes in the element.

7.4.4.18 ButcherCoefficients Container for A, B and C coefficients as defined by Butcher. This class acts as an abstract base and several types of numerical discretization schemes can be represented using this method. A clear way of representing multi-step methods might be a little complicated but all Runge-Kutta schemes, both Implicit and Explicit are usually represented in this way and hence this class creates a generic way of representing all such time discretization methods.

7.4.4.19 BackwardEuler It derives from ButcherCoefficients and is an implicit, first-order Backward Euler time discretization scheme.

7.4.4.20 CrankNicholson It derives from ButcherCoefficients and is an implicit, second-order Crank Nicholson time discretization scheme. This scheme is not L-stable and has bad damping properties leading to spurious oscillation when the time step used is larger than its L-stability limit.

7.5 Planned models

7.5.0.21 Physics model 1) 3-D *cartesian* Multi-group neutron diffusion model with homogenized assembly level cross-sections obtained from lattice physics code like CASMO.

2) Multi 2-D, $r-z$ nonlinear heat conduction in one typical fuel pin cell per assembly coupled to average heat generation from fission power and to heat convection to coolant at the fuel surface

3) Multi 1-D channel single-phase fluid flow model coupled to heat addition from fuel; This coupling term appears as a reaction term in the energy conservation equation.

7.5.0.22 Discretization Model 1) FEM discretization of the 3-D space with *prism* type finite elements for the Neutronics calculation

2) FEM discretization of the 2-D $r-z$ space in the fuel pin for the Heat Conduction calculation

3) FVM discretization of the 1-D channel in the axial direction for the single-phase fluid flow calculation

7.5.1 Features

1) Capability to model the steady state conditions in a reactor by means of a pseudo-transient procedure

2) Analyze rod-ejection accidents in the core and simulate the transient evolution

3) Analyze SCRAM scenarios during a power trip and analyze the transient solution

4) When the two-phase fluid model is in place, accidents like LOCA can also be simulated

8 References

1) Vijay S Mahadevan, "Nonlinearly consistent schemes for coupled problems in reactor analysis", Master of Science thesis document (Dec 2006)

2) Saad, Y. and Schultz, M. "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems." SIAM J. Sci. Statist. Comput. 7, 856-869, 1986

3) Knoll D. A., Keyes D. E., "Jacobian-free Newton-Krylov methods: a survey of approaches and applications", Journal of Computational Physics, 193, 2, 357-397, (2004)

4) Ober C., Shadid J., "Studies on the accuracy of time integration methods for radiation-diffusion equations", Journal of Computational Physics, 195, 743-772, (2004)

5) Mousseau,V.A., Knoll, D.A., Rider W.J., "Physics-Based Preconditioning and the Newton-Krylov Method for Non-equilibrium Radiation Diffusion", Journal of computational physics, 160, 2, 743-765, (Feb 2000)

6) Mousseau V.A, "Implicitly balanced solution of the two-phase flow equations coupled to nonlinear heat conduction", Journal of computational physics, 200, 1, 104-132, (Oct 2004)

7) Leveque R. J., "Finite Volume methods for Nonlinear Scalar conservation laws", Cambridge University Press

8) F.H. Harlow, A.A. Amsden, A numerical fluid dynamics calculation for all flow speeds, J. Comput. Phys. 8 (1971) 197

9) Martineau, R. C. and Berry, R. A. 2004. The pressure-corrected ICE finite element method for compressible flows on unstructured meshes. J. Comput. Phys. 198, 2 (Aug. 2004), 659-685

10) Knoll D.A., Chacon L., Margolin L.G., Mousseau V.A., "On balanced approximations for time integration of multiple time scale systems", Journal of Computational Physics, 185, 2, 583-611 (Mar. 2003)

11) Portable, Extensible Toolkit for Scientific Computation (PETSc), Weblink:
`http://www-unix.mcs.anl.gov/petsc/petsc-as/documentation/index.html`