SANDIA REPORT

SAND2011-2195
Unlimited Release
Printed March, 2011

A Theory Manual for Multi-physics Code Coupling in LIME

Version 1.0

Roger Pawlowski, Roscoe Bartlett, Noel Belcourt, Russell Hooper, Rod Schmidt

Sandia National Laboratories

SAND2011-2195
Unlimited Release
Printed March, 2011

A Theory Manual for Multi-physics Code Coupling in LIME

Version 1.0

Roger Pawlowski, Roscoe Bartlett, Noel Belcourt,
Russell Hooper, and Rod Schmidt
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185

Abstract

The Lightweight Integrating Multi-physics Environment (LIME) is a software package for creating multi-physics simulation codes. Its primary application space is when computer codes are currently available to solve different parts of a multi-physics problem and now need to be coupled with other such codes. In this report we define a common domain language for discussing multi-physics coupling and describe the basic theory associated with multi-physics coupling algorithms that are to be supported in LIME. We provide an assessment of coupling techniques for both steady-state and time dependent coupled systems. Example couplings are also demonstrated.

Acknowledgment

We thank Eric Phipps (Sandia National Laboratories) for providing the basis for the example couplings chapter [22].

This work was funded by the Consortium for Advanced Simulation of Light Water Reactors (CASL), a Department of Energy Innovation Hub [1].

Contents

List of Figures

Chapter 1

Introduction

Many problems of great scientific and national interest involve complex systems with numerous, sometimes disparate, interconnected components that involve many different physical processes that interact or couple in a variety of ways. Examples include the design, safety analysis and licensing of current and future nuclear reactors, development of renewable energy technologies, vulnerability analysis of water and power supplies, understanding of complex biological networks, and many others. As our ability to simulate such systems using advanced computational tools has improved, our appreciation for the challenges associated with predictive simulation has also deepened. Often these systems strongly couple many different physical processes exhibiting phenomena at a diverse set of length and/or time scales. These interacting, nonlinear, multiple time-scale physical mechanisms can balance to produce steady-state behavior, nearly balance to evolve a solution on a dynamical time scale that is long relative to the component time-scales, or can be dominated by just a few fast modes. These characteristics make the scalable, robust, accurate, and efficient computational solution of these systems over relevant dynamical time scales of interest (or to steady-state solutions) extremely challenging.

The Lightweight Integrating Multi-physics Environment (LIME) is a software package being developed to help create multi-physics simulation codes. Its primary application space is when computer codes are currently available to model and solve different parts of a multi-physics problem and now need to be coupled with other such codes. Thus a sound understanding of the mathematical theory associated with multi-physics coupling is of fundamental importance for potential users of LIME.

Because multi-physics coupling issues arise in many different scientific and engineering arenas, they often involve multi-disciplinary teams. Our experience is that, without a common vocabulary and associated definitions, application development teams tend to talk past one another, wasting significant time and effort trying to describe basic algorithms. Thus the first purpose of this report is to define a common set of terms and associated definitions for use as we describe multi-physics coupling. The second objective is to provide a concise description and discussion of the mathematical theory associated with multi-physics coupling algorithms. For many problems of interest, tremendous differences in performance behavior can be observed when different coupling approaches are applied. Understanding these differences, their characteristics, and the theory behind the different methods will enable potential users of LIME to make informed choices about how to couple their physics codes when leveraging the LIME package.

When discussing multi-physics simulation, one must distinguish between coupled physics and coupled physics codes. So it is important to note that we will use the word "coupled" in two related but different contexts; one with respect to computer codes, and one with respect to physics. In principle, a single physics code can be written to simulate any coupled multi-physics system, i.e. a single physics code is not restricted to a single-physics. Of interest to LIME is the ability for multiple such physics codes to be coupled in some fashion to simulate a multi-physics system. In this case each physics code separately approximates the solution to one or more sets of physics in the overall coupled multi-physics system. This document describes strategies for coupling multiple physics codes together to solve coupled multi-physics problems.

This report is purposefully not specific to software implementation. While the algorithms we target are associated with the LIME software suite [26], this document will only describe the general theory. A companion report provides an introduction to LIME, its design and its use.

This document is intentionally kept short so that users and developers alike can quickly grasp the domain language.

The organization of this report is as follows. In the next section, we describe the basic nomenclature used in multi-physics coupling. In chapter 2, a rigorous mathematical model of multi-physics coupling is presented. This includes a section on what multi-physics coupling strategies are available depending on how much information (i.e. how invasive the interaction) each physics application code can expose. The final chapter shows examples for steady-state and transient couplings.

1.1 Basic Definitions

Here we define a set of key terms used when describing multi-physics coupling.

- **Coupled Physics:** When the solution of one physics equation (set) is dependent upon the solution of another different physics equation (set).
- **Coupled System:** Effectively synonymous with "coupled physics", but used here to emphasize the different components that make up a coupled physics problem (each associated with different physical processes). Section 3.1 defines coupled systems with a **shared spatial domain**, with **interfacial coupling**, and with what we call **network systems**.
- **Multi-Physics:** A problem of interest modeled using more than one set of physical processes (and associated equation sets). Here we are only interested in cases where these physical processes are coupled.

- **Degree of Physics Coupling:** The degree of physics coupling is the measure of how much one physics set influences another physics set. If a change to a solution variable in one physics set produces a comparable response in another physics set, then the coupling is said to be “strong”. Conversely, if a change in the solution of one physics set results in a negligible change in another physics set, then the coupling is said to be “weak”. “Strong” and “weak” refer to the interdependence among the models (and associated equation sets).

- **Directional Coupling:** In the spirit of the degree of coupling, coupling also has an associated direction. “Two-way coupling” is where two physics sets that are coupled together each directly depend on the solution of the other set. In “one-way” or “forward” coupling, one physics set depends on another but not vice-versa. This has implications for coupling solution algorithms, e.g. one physics can be solved independent of the other physics.

- **Physics Code:** Any computer code that solves a particular set of physics (or model) equations. The terms “application”, or “physics application” will also be used here to mean the same thing. A single physics code can be further classified as **single-physics** in that it solves only one set of physics equations (e.g. heat conduction) or **multi-physics** in that it internally solves more than one set of physics equations (e.g. Navier-Stokes and energy conservation [4]).

- **Coupled Physics Codes:** Two or more physics codes that have been coupled together to simulate a multi-physics system. Information is transferred between computer codes during the solution process so that there is feedback between the components.

- **Monolithic vs. Partitioned:** Refers to algorithms used to effect a solution to a multi-physics system comprising two or more physics codes. These terms come from work done in the area of fluid/structure interaction [7, 15]. In this community, solution strategies are classified as either **monolithic** or **partitioned**. *Monolithic* refers to using a single solver for the complete set of coupled physics equations with time stepping performed synchronously at each time step. Monolithic approaches to coupling typically employ fully implicit Newton schemes. *Partitioned* methods refer to using the individual solvers of each physics application in asynchronous time advancement schemes. Partitioned solvers tend to be blocked systems that involve explicit time integration and/or operator splitting. It is critical to note that different partitioning methods will produce different numerical solutions for a given set of transient multi-physics equations for a given selection of time steps, no matter how tight the linear and nonlinear tolerances are set in the individual (single-physics) solution algorithms.

- **Numerical Solution Strategy:** A general term that refers to the approach used to obtain a solution to the combined physics equation sets. There are many different facets to this topic and a host of related terms are used to describe different types or different aspects of numerical solution strategies. Of particular interest here are terms that describe specific strategies for solving coupled systems of algebraic equations, such as “Newton” methods, “fixed-point” or “Picard” iteration, “coupled block techniques”, "nonlinear elimination", "Multi-level Newton" methods, and "Jacobian-Free Newton-Krylov (JFNK)" methods. It is critical to note that any of these methods will produce the same numerical solution if the various solution tolerances are made tight enough and the overall solution strategy is monolithic. Only the rates of convergence and the cost per iteration vary between the different solution methods.

- **Consistency:** A consistent solution is one where all quantities of interest are evaluated using the same global state. Accurate determination of solution convergence requires consistency. Consistency is automatic for fully coupled implicit methods but must be enforced as an additional step for numerical solution strategies that lag or asynchronously update pieces of the overall solution.

- **Conservation:** Most equation sets used as a basis for computing the state variables in physics codes are derived as statements about the conservation of some important quantity, e.g. mass, momentum, or energy. Here we are particularly interested in the transfer of these conserved quantities between coupled physics codes. Different ways of transferring data between coupled physics codes can preserve or corrupt these quantities of interest which in turn affect the stability and accuracy of the coupling algorithm [17]. For example, for heat transfer across a fluid/structure interface, the total energy leaving the fluid domain should be equal to the total energy entering the solid structure. This aspect of conservation should not be confused with the conservation properties of discretization schemes used for solving the partial differential equations within the physics codes themselves.

- **State Variables:** Refer to the solution variables of a physics code. Also called the unknowns or degrees of freedom (DOFs) that the physics application is solving for. The union of state variables from all physics codes represents the solution state of the coupled multi-physics system. We sometimes refer to this as the composite solution.

- **Residual or Constraint Function:** Refer to the set of equations that are solved to compute the state variables. The term constraint equation comes from the optimization community where the equations are used to constrain the objective function. The term residual comes from the nonlinear solution community and represents a set of equations driven toward zero through nonlinear iterations to approach a solution.

- **Response Function:** Refers to a quantity of interest that a simulation is used to predict. The response function can be the state variables themselves but is more commonly a post-processed set of values derived from the state variables and other independent application parameters (e.g. flux, lift, max stress, etc.).

- **Transfer Function:** Refers to a function (or set of equations) used to map data (state, parameter and response function data) from one application into a form that is accepted as input for another application. Transfer functions perform the process of communication between different single-physics codes.

Chapter 2

General Formulation for Multi-physics Coupling

This chapter provides a rigorous general mathematical description of a multi-physics coupled problem. Appendix A defines the relevant mathematical notation taken from set theory. The first section describes a single-physics application and the second section expands on this to describe an arbitrary set of physics applications coupled together. Chapter 3 will simplify this model and give concrete examples of coupling strategies and solution techniques. If you find yourself getting bogged down in the math in this chapter, we recommend you move on to Chapter 3 to get a feel for the math from the simplified examples.

2.1 Single-physics Application

We start by defining a general expression for the mathematical equations in any nonlinear single-physics application.

$$f(\dot{x}, x, \{p_l\}, t) = 0 \tag{2.1}$$

where

$x \in \mathbb{R}^{n_x}$ is the vector of state variables (unknowns being solved for),
$\dot{x} = \partial x/\partial t \in \mathbb{R}^{n_x}$ is the vector of derivatives of the state variables with respect to time,
$\{p_l\} = \{p_0, p_1, \ldots, p_{N_p-1}\}$ is the set of N_p independent parameter sub-vectors,
$t \in [t_0, t_f] \in \mathbb{R}^1$ is the time ranging from initial time t_0 to final time t_f,

and where we will call

$$f(\dot{x}, x, \{p_l\}, t) : \mathbb{R}^{\left(2n_x + \left(\sum_{l=0}^{N_p-1} n_{p_l}\right) + 1\right)} \rightarrow \mathbb{R}^{n_x}$$

the "residual" or "constraint" equations of the physics code.

Equation 2.1 describes a set of n_x nonlinear equations, f, that are used to solve for n_x state variables x. Users can set independent parameters listed here in the form of N_p subvectors $\{p_l\}$ in the simulation where n_{p_l} is the size of parameter subvector p_l. The simulation could be transient, requiring information on the time and time derivative, t and $\dot{x}$, respectively, or it could be steady-state.

For a steady-state simulation, equation 2.1 simplifies to

$$f(x, \{p_l\}) = 0. \tag{2.2}$$

In many situations the end use of a simulation is to compute one or more quantities of interest, g_j, here called a response function. For N_g response functions we can express the associated set of response-functions as a vector:

$$g_j(\dot{x}, x, \{p_l\}, t) = 0, \text{ for } j = 0, \ldots, N_g - 1 \tag{2.3}$$

where

$$g_j(\dot{x}, x, \{p_l\}, t) : \mathbb{R}^{\left(2n_x + \left(\sum_{l=0}^{N_p-1} n_{p_l}\right)+1\right)} \to \mathbb{R}^{n_{g_j}} \text{ is the } j^{\text{th}} \text{ response function.}$$

For example, suppose we want to find the total heat flux through a boundary Γ of a concrete slab Ω. The physics of heat transfer in a solid is described by a partial differential equation for energy conservation using a Fourier closure model [4] for the heat flux, $q = -k\nabla T$, where k is the thermal conductivity. After discretizing the partial differential equation over domain Ω, we construct a set of n_x nonlinear equations, $f(x)$, in terms of n_x temperatures (i.e. the state variables) defined on Ω. After solving the equations for the temperatures, a post processing step computes our quantity of interest, g, which is the total heat flux through surface Γ by integrating the heat flux over the surface

$$g = \int_\Gamma n \cdot (-k\nabla x). \tag{2.4}$$

Typically, the response functions are calculated as a post processing step, but they can also be added as extra state variables and computed as part of the solve. We will address this more in the following sections.

2.2 Extension to Multi-physics

We now proceed to describe a coupled set of N_f single-physics applications. Here the independent parameters for a single-physics application may now be dependent on state variables and parameters from other applications. To express this mathematically in a sufficiently general way, the parameter vector from a single-physics application, $\{p_l\}$, is now split into (a) parameters that are dependent on other application data, $\{z_k\}$, and (b) the remaining independent parameters, redefined now as $\{p_l\}$. In the following, $\{z_{i,k}\}$ is the set of data (N_{z_i} sub-vectors) that is transferred to application i.

The constraint (or residual) equations for physics application i are now defined as

$$f_i(\dot{x}_i, x_i, \{z_{i,k}\}, \{p_{i,l}\}, t) = 0, \text{ for } i = 0, \ldots, N_f - 1, \tag{2.5}$$

where

$x_i \in \mathbb{R}^{n_{x_i}}$ is vector of state variables for application i,
$\dot{x}_i \in \mathbb{R}^{n_{x_i}}$ is vector of derivatives of the state variables for application i with respect to time,
$\{z_{i,k}\} = \{z_{i,0}, z_{i,1}, \ldots, z_{i,N_{z_i}-1}\}$ is the set of N_{z_i} coupling parameter sub-vectors for application i,
$\{p_{i,l}\} = \{p_{i,0}, p_{i,1}, \ldots, p_{i,N_{p_i}-1}\}$ is the set of independent parameter sub-vectors for application i,
$t \in [t_0, t_f] \in \mathbb{R}^1$ is time ranging from initial time t_0 to final time t_f, and
$f_i(\dot{x}_i, x_i, \{z_{i,k}\}, \{p_{i,l}\}, t) : \mathbb{R}^{\left(2n_{x_i} + \left(\sum_{k=0}^{N_{z_i}-1} n_{z_{i,k}}\right) + \left(\sum_{l=0}^{N_{p_i}-1} n_{p_{i,l}}\right) + 1\right)} \rightarrow \mathbb{R}^{n_{x_i}}$ is the constraint equations for application i.

There now exists an extra set of requirements for transferring the parameter information, $z_{i,k}$, between codes. These transfer functions can be simple post-processing that uses solution and parameter values from the other applications or it can be something as complex as a complete nonlinear solve. We define the inter-application transfer functions, $r_{i,k}$ for physics application i, as

$$z_{i,k} = r_{i,k}(\{x_m\}, \{p_{m,n}\}), \text{ for } i = 0, \ldots, N_f - 1,\ k = 0, \ldots, N_{z_i} - 1, \tag{2.6}$$

where

$\{z_{i,k}\} = \{z_{i,0}, z_{i,1}, \ldots, z_{i,N_{z_i}-1}\}$ is the set of N_{z_i} coupling parameter sub-vectors for physics application i,
$\{x_m\} = \{x_0, x_1, \ldots, x_{N_f-1}\}$ is the set of state variables for all applications,
$\{p_{m,n}\} = \{p_{0,0}, \ldots, p_{0,N_{p_0}-1}, \ldots, p_{N_f-1,0}, \ldots, p_{N_f-1,N_{pn}-1}\}$ is the set of all coupling parameter sub-vectors for all applications, and
$r_{i,k}(\{x_m\}, \{p_{m,n}\}) : \mathbb{R}^{\left(\sum_{m=0}^{N_f-1}\left(n_{x_m} + \sum_{n=0}^{N_{p_m}-1} n_{p_{m,n}}\right)\right)} \rightarrow \mathbb{R}^{n_{z_{i,k}}}$ is the transfer function for application i for coupling parameter k.

In essence, for physics i we will have a number of transfer operators equal to N_{z_i}. Each transfer operator $r_{i,k}(\ldots)$ evaluates single parameter sub-vector $z_{i,k}$. Strategies on how to aggregate the evaluation of the transfer operators are usually dictated by software design constraints, data transfer requirements for the multiple code couplings supported, and efficiency considerations.

Some brief comments about transfer functions:

- In terms of software implementation, there are many ways to handle transfers of information between codes. The transfer functions, r, are only a mathematical concept that explicitly demonstrates the transfers. Software implementations could implement this directly (we chose to show transfer in this fashion because it translates readily to code). An alternative and equally valid way to transfer information is to directly write the transfer functions into the residual operator f. We typically do this when the code already does the post-processing during the solution step. Yet another way to implement transfers is to treat the transfer operator as an entirely new single-physics application by implementing an additional f. In this case, simple transfer mappings, r, are used to copy the data between residual functions.

  In our experience, we usually reserve the transfer operators r for simple mappings and for anything that requires a linear or nonlinear solver, we implement the transfer as an additional physics application f.

- In practice, the sets for $\{x_m\}$ and $\{z_{m,n}\}$ for a particular transfer function r are very sparse. They usually take the solution and parameters from one physics application and generate values for another physics application.

- The union of all transfer operators defines an implicit dependency graph between the coupled physics applications. Explicitly exposing this information can help determine nonzero sensitivity blocks for implicit solution techniques.

- We could extend the transfer operators r to be dependent on time ($\dot{x}_m$ and t) as well. We have chosen to avoid that for now as a number of complications arise.

We now define response functions for the multi-physics case. The j^{th} response function associated with the multi-physics system is defined as

$$g_j(\{\dot{x}_i\}, \{x_i\}, \{z_{i,k}\}, \{p_{i,l}\}, t) = 0, \text{ for } j = 0, \ldots, N_g - 1 \tag{2.7}$$

where $\{x_i\}$, $\{z_{i,k}\}$, $\{p_{i,l}\}$, and t are defined in (2.5), (2.6), and

$\{\dot{x}_i\} = \{\dot{x}_0, \dot{x}_1, \ldots, \dot{x}_{N_f-1}\}$ is the set of state variable time derivatives for all applications,

$g_j(\{\dot{x}_i\}, \{x_i\}, \{z_{i,k}\}, \{p_{i,l}\}, t) : \mathbb{R}^{\left(\sum_{i=0}^{N_f-1}\left(2n_{x_i}+\left(\sum_{k=0}^{N_{z_i}-1} n_{z_{i,k}}\right)+\left(\sum_{l=0}^{N_{p_i}-1} n_{p_{i,l}}\right)\right)+1\right)} \rightarrow \mathbb{R}^{n_{g_j}}$ is response function j.

Note that in many cases, response functions, g_j for a multi-physics system can correspond exactly to the input parameters $z_{m,n}$ of one or more of the individual applications in the multi-physics system. While we have explicitly differentiated between the response functions from one or more codes and the inputs to another code, in practice they can be the same.

The fully coupled set of physics applications can now be expressed with the following set of equations

$$\hat{f}(\hat{\dot{x}}, \hat{x}, \hat{p}, t) = 0, \tag{2.8}$$

where

$$\begin{aligned}
\hat{\dot{x}} &= \left[\dot{x}_0, \ldots, \dot{x}_i, \ldots, \dot{x}_{N_f-1}\right], \\
\hat{x} &= \left[x_0, \ldots, x_i, \ldots, x_{N_f-1}\right], \\
\hat{p} &= \left[p_{0,0}, \ldots, p_{0,N_{p_0}-1}, \ldots, p_{i,0}, \ldots, p_{i,N_{p_i}-1}, \ldots, p_{N_f-1,0}, \ldots, p_{N_f-1,N_{p_{N_f-1}}}\right], \\
\hat{f} &= \begin{bmatrix} f_0(\dot{x}_0, x_0, \{r_{0,k}(\{x_m\}, \{p_{m,n}\})\}, \{p_{0,l}\}, t)) \\ \vdots \\ f_i(\dot{x}_i, x_i, \{r_{i,k}(\{x_m\}, \{p_{m,n}\})\}, \{p_{i,l}\}, t)) \\ \vdots \\ f_{N_f-1}(\dot{x}_{N_f-1}, x_{N_f-1}, \{r_{N_f-1,k}(\{x_m\}, \{p_{m,n}\})\}, \{p_{N_f-1,l}\}, t)) \end{bmatrix}.
\end{aligned} \tag{2.9}$$

We call these two equations the *General Model* for multi-physics coupling. This is the most flexible model for multi-physics coupling supported in LIME and allows for any implicit strongly coupled solution method for both transient and steady-state problems. Simplifications of this model can be applied for weak couplings and/or explicit and semi-implicit/operator split time stepping strategies.

If application codes cannot write to the general interface, the available solution methods are limited by what the application can accept from and expose to the coupling driver. For example if the application cannot accept $\dot{x}$, then the multi-physics coupling algorithm can no longer drive the time integration algorithm and requires the internal application to use its own internal time integrator. In this case, the general model reduces to $\tilde{f}(\tilde{x}, \tilde{p}) = 0$. The driver can still pass in a time step size to take as a parameter, but the time integration is now internal to the application.

Note that in the general model, the internal coupling parameters/transfer functions are not exposed to the solvers. These are internal parameters and functions that handle the coupling.

2.3 Solution Strategies for the General Model

This section describes the basic algorithms used to solve the general model assuming that it is composed of nonlinear equations. The goal is not to present an exhaustive list of solution strategies, but to describe the typical requirements that a physics code must support for a particular solution method. For a thorough analysis of nonlinear solution techniques see [6, 11, 19]. The information in this section should be used to help determine the best coupling strategy for a particular set of applications or to select the most practical approach given current legacy software constraints. Specific examples of coupling methods are given in section 3.2.

The equations that compose the general model can be generated from any number of sources. For example, it could be a set of ordinary differential equations (ODEs), a discretized set of partial differential equations (PDEs), a set of algebraic equations, or a set of differential algebraic equations (DAEs). Once the spatial and temporal discretizations are applied, equation 2.8 (for a single time step) reduces to a simpler set of nonlinear equations expressed entirely in terms of the state variables and the independent parameters.

$$\bar{f}(\bar{x}, \bar{p}) = 0$$

where $\bar{x} \in \mathbb{R}^{n_{\bar{x}}}$ is the state vector consisting of the union of all application state vectors that are exposed through the general model interface, $\bar{p} \in \mathbb{R}^{n_{\bar{p}}}$ is the independent parameter vector consisting of the union of all application parameter vectors that are exposed through the general model interface, and $\bar{f} : \mathbb{R}^{n_{\bar{x}}+n_{\bar{p}}} \to \mathbb{R}^{n_{\bar{x}}}$ is the residual of the nonlinear equations.

The first solution method we mention is a stationary iterative method whose variants are known by multiple names: Picard iteration, nonlinear Richardson iteration, successive substitution or fixed-point iteration. Here we do not discuss differences among these variants and will refer to them collectively as Picard iteration methods. The general procedure is shown in Algorithm 1, where $G : \mathbb{R}^{n_{\bar{x}}} \to \mathbb{R}^{n_{\bar{x}}}$ is what we call a contraction mapping on $\bar{x}$. The

Algorithm 1 Picard iteration for a single-physics.

```
Require: Initial guesses $\bar{x}_0^{(0)}$:
  $k = 0$
  while not converged do
    $k = k + 1$
    $\bar{x}^{(k)} = G(\bar{x}^{(k-1)})$
  end while
```

contraction mapping is only a function of the previous iterate value. k is the iteration count in the algorithm. The algorithm is generally simple to implement and the mapping is usually very cheap to perform, typically only requiring an evaluation of the constraint equations, $\bar{f}(\bar{x})$. In practice this method is usually very robust, although there are requirements for stability of the algorithm [14, 19] that can be violated in some settings. The major drawback is that the convergence rate is q-linear in the norm [6]. Thus it may take many iterations to converge to a stringent tolerance.

The primary alternative to the Picard iteration class of methods are various methods based on Newton's method. The basic algorithm for a Newton method is described by Algorithm 2, where $\bar{J}(\bar{x}^{(k)})$ is the Jacobian matrix, $\bar{J} = \frac{\partial \bar{f}}{\partial \bar{x}}$, evaluated using the state variables $\bar{x}^{(k)}$ at iterate k. The advantages of Newton methods include a q-quadratic convergence rate in the error norm and the stability of the algorithm [11]. Although robustness issues can arise, these can be addressed by leveraging globalization techniques such as line search and trust region methods [5, 6, 18, 21].

The primary drawback to Newton's method is the cost and difficulty of computing the full Jacobian matrix. In the context of LIME and the general model expressed above, we

Algorithm 2 Newton's method.

```
Require: Initial guesses $\bar{x}_0^{(0)}$:
  $k = 0$
  while not converged do
    $k = k + 1$
    $\bar{x}^{(k)} = \bar{x}^{(k-1)} - \bar{J}^{-1}(\bar{x}^{(k-1)})\bar{f}(\bar{x}^{(k-1)})$
  end while
```

note that the Jacobian matrix requires model sensitivities for each application with respect to the state variables for each other application, as well as its own

$$\bar{J} = \begin{bmatrix} \bar{J}_{0,0} & \dots & \dots & \dots & \bar{J}_{0,N_f-1} \\ \vdots & \ddots & \vdots & & \vdots \\ \bar{J}_{i,0} & \dots & \bar{J}_{i,i} & \dots & \bar{J}_{i,N_f-1} \\ \vdots & & \vdots & \ddots & \vdots \\ \bar{J}_{N_f-1,0} & \dots & \dots & \dots & \bar{J}_{N_f-1,N_f-1} \end{bmatrix} \tag{2.10}$$

where

$$\begin{aligned} \bar{J}_{i,i} &= \frac{\partial \bar{f}_i}{\partial \bar{x}_j} = \frac{\partial f_i}{\partial z_{i,k}} \in \mathbb{R}^{n_{x_i} \times n_{x_i}}, \text{ for } i = 0 \dots N_f - 1 \\ \bar{J}_{i,j} &= \frac{\partial \bar{f}_i}{\partial \bar{x}_j} = \sum_{k=0}^{N_{z_i}-1} \frac{\partial f_i}{\partial z_{i,k}} \frac{\partial r_{i,k}}{\partial x_j} \in \mathbb{R}^{n_{x_i} \times n_{x_j}}, \text{ for } i = 0 \dots N_f - 1,\ j = 0 \dots N_f - 1,\ i \neq j, \end{aligned}$$

and the diagonals $\bar{J}_{i,i}$ are typically nonsingular matrices (but may not always be) but the non-diagonal blocks $\bar{J}_{i,j}$ $(i \neq j)$ are typically rectangular matrices.

Note that even if each individual application were to implement a Newton-based solve, this would only supply the Jacobian diagonal blocks $\bar{J}_{i,i}$ in equation 2.10. The off-diagonal sensitivities provided by the transfer functions would still be missing. One way to address the off-diagonal blocks is to leverage an approximate Newton-based method called the Newton-Krylov approach. When using Newton-Krylov solvers [12] we avoid the cost of constructing an explicit Jacobian. Newton-Krylov solvers build up an approximation to the solution of the Newton system by applying Jacobian-vector products to construct a Krylov subspace, $\mathcal{K}(\bar{J}, v) \equiv span\{v, \bar{J}v, \bar{J}^2 v, \dots\}$. By requiring only the Jacobian-vector products, $\bar{J}v$, to solve the Newton system, the Jacobian need not be explicitly formed. While an explicit Jacobian matrix could be used, the Jacobian-vector product can be computed to machine precision using automatic differentiation [3, 9] or approximated by directional differences using only residual evaluations

$$\bar{J}v = \frac{\bar{f}(\bar{x} + \epsilon v) - f(\bar{x})}{\epsilon}. \tag{2.11}$$

Here, $v \in \mathbb{R}^{n_{\bar{x}}}$ is a Krylov vector and $\epsilon \in \mathbb{R}^1$ is a perturbation parameter. This method eliminates the burdensome and error-prone procedure of hand coding an analytic Jacobian and reduces the runtime memory footprint since the Jacobian is not explicitly stored. Example applications using Jacobian-free Newton-Krylov can be found in [10] and [12].

2.4 Classification of Code Coupling Capabilities

This section ties the *general model* described in the previous section (equations 2.5, 2.6, and 2.7) to actual application couplings. The goal here is to provide a common language that can be used to describe how a particular application code can interact with a multi-physics coupling framework. Each coupling classification can support a specific set of algorithms based on how much information can be passed between the application and coupling driver. As more information is exposed between the application and coupling driver a wider variety of coupling algorithms can be applied. The various levels of classification are now listed progressing from least invasive to most invasive (where p is an aggregation of coupling variables z to other physics as well as other general non-coupling parameters p used to drive optimization, UQ and other parameter studies).

- **Time-step (or Steady-state) coupling elimination model:** Elimination of the entire state only exposing information for coupling in the form of a response function, $p \rightarrow g(p)$. This is how standard "Black-Box" optimization software interfaces to applications [2].

- **Time-step (or Steady-state) state elimination model:** Elimination of time-step nonlinear system $p \rightarrow \hat{x}(p)$. This method does not assume $\frac{\partial \hat{x}}{\partial p}$ is available. The coupling equation/function to other physics is externally defined. Could be coupled to other physics with nonlinear Gauss-Seidel (i.e. Picard iteration) [14]. Works for implicit and explicit internal time integration methods (internally implemented in the closed application code).

- **Time-step (or Steady-state) fully implicit model:** Exposes the time-step (or steady-state) nonlinear system $\hat{f}(\hat{x}, p) = 0$. We assume that $\frac{\partial \hat{f}}{\partial \hat{x}}$ exists and we have an operator (does not have to be explicitly formed) and a preconditioner for it. Allows full Newton (and variations thereof) to be used in time-step. If transient, the time step size is passed in as a parameter, but the time derivative cannot be supplied by the coupling driver. Therefore, only an internally defined time integrator can be used with the application for transient simulations. This works for implicit time integration methods only.

- **Transient explicitly defined ODE model:** The transient model equations are exposed explicitly as a set of ordinary differential equations (ODEs) of the form $\dot{x} = f(x, p, t)$ where the user evaluates f given the values of x, t, and p from the multi-physics driver. The multi-physics driver code performs the time integration using any explicit or implicit time integration strategy available. The basic call-back routine minimally requires the computation of f. If implicit methods are used, the Jacobian, $W = \alpha I + \beta \frac{\partial f}{\partial x}$, must also be supplied as well as its preconditioner (in operator form).

Name	Required Inputs	Required Outputs	Optional Outputs	Time Integration Control
Coupling elimination model	p	g or z		Internal
State elimination model	p	x		Internal
Fully implicit time-step model	x, p	f	W, M	Internal
Transient explicitly defined ODE model	x, p, t	f	W, M	External
Transient fully implicit DAE model	$\dot{x}, x, p, t$	f	W, M	External or Internal

Table 2.1. Classifications of code couplings. Table progresses from least invasive (top) to most invasive interface (bottom).

- **Transient fully implicit DAE model:** A fully implicit differential algebraic equation (DAE) model (the general model) of the form $f(\dot{x}, x, t, p) = 0$ is supported by the application code. Here we assume that the code can supply f and W where $W = \alpha\frac{\partial f}{\partial \dot{x}} + \beta\frac{\partial f}{\partial x}$ and a preconditioner M for use in the resulting Newton iterations. This allows for any implicit or explicit solution method. This is the absolute most flexible code integration model. The coupling driver can choose any external time integration technique and any nonlinear solution technique since it can provide $\dot{x}$, x, and t to the application. The application provides the necessary state sensitivity information for a Newton-based approach.

The levels of invasiveness are summarized in Table 2.1 by classification. Note that applications can support multiple levels of invasiveness (the software implementation of the ModelEvaluator concept - see LIME users guide - was designed to support all levels). Typically, applications start with the coupling elimination model and then expand down the chain as needed to balance convergence robustness with computational efficiency.

Chapter 3

Example Couplings and Solution Strategies

This chapter discusses several examples of coupled systems and the solution techniques that can be applied in each case. The first section shows simple couplings where the time integration is handled by the application and only the coupled nonlinear equations for a given time-step are solved.

3.1 Examples of Mathematical Models for Different Types of Coupling

Broadly speaking there are three types of coupled systems that are widely studied through the scientific community: multi-physics systems within a shared spatial domain, interfacially coupled systems, and network systems. Of course real physical systems may consist of a combination of more than one of these types. Our description of the coupled systems in this section is for *illustration purposes only* and is but one of a myriad of possibilities. A very simplified model is used to demonstrate shared spatial domains, a more complex model is used to demonstrate interfacially coupled systems and a fairly complex model is used to demonstrate network systems. In this way we can progress from simple examples to more complex couplings.

The examples in this section are limited to either steady-state problems or transient problems where a suitable implicit time discretization method has been chosen. In the latter case, the methods discussed here are then applicable to the implicit nonlinear equations produced for each time step. For simplicity in introducing multi-physics coupling, we will assume the transient terms are already discretized.

3.1.1 Shared Spacial Domain Model Example

The first class of coupled multi-physics systems discussed here consist of multiple interacting physics in a **shared spatial domain**. A familiar example is fluid flow and heat transfer (see

Figure 3.1(a)), where several conservation laws (i.e. mass, momentum and energy) simultaneously govern the evolution of the state variables (e.g. pressure, velocity, and temperature). Because the equations for conservation of momentum and mass are so strongly coupled, fluid flow problems are almost universally solved within one code, even though several distinct equation sets are being solved. However, this is not a requirement and there are other problems of interest where having separate application codes for the different physics is the norm.

Systems of this type are often modeled by a set of coupled, possibly nonlinear, partial differential equations over a given domain. Although each application shares the same physical domain, they could potentially have a completely different spatial discretization. After discretization in space, a generic two-component multi-physics system can be represented by the coupled nonlinear equations

$$\begin{aligned}
f_0(x_0, z_{0,0}) &= 0 \\
f_1(x_1, z_{1,0}) &= 0 \\
z_{0,0} &= r_{0,0}(x_1) = x_1 \\
z_{1,0} &= r_{1,0}(x_0) = x_0 \\
&\Rightarrow \\
\bar{f}(\bar{x}) = \begin{bmatrix} f_0(x_0, r_{0,0}(x_1)) \\ f_1(x_1, r_{1,0}(x_0)) \end{bmatrix} &= \begin{bmatrix} f_0(x_0, x_1) \\ f_1(x_1, x_0) \end{bmatrix} = 0.
\end{aligned} \tag{3.1}$$

Here $x_0 \in \mathbb{R}^{n_{x_0}}$ and $x_1 \in \mathbb{R}^{n_{x_1}}$ represent the discretized solution variables for each physical system, and $f_0 : \mathbb{R}^{n_{x_0}+n_{x_1}} \rightarrow \mathbb{R}^{n_{x_0}}$ and $f_1 : \mathbb{R}^{n_{x_1}+n_{x_0}} \rightarrow \mathbb{R}^{n_{x_1}}$ represent the corresponding discretized residuals. Here we have assumed that the equations have the same spatial discretization, and thus the transfer functions, $r_{i,0}(\ldots)$, are trivial transfers (direct copies) of the solution (state variables) from one application to another ($i.e.$ $r_{i,0}(x_j) = x_j$). Here n_{x_0} and n_{x_1} are the number of state variables for physics 0 and physics 1 respectively.

3.1.2 Interface Coupling Model Example

A second type of coupling is **interfacial coupling**. Here we have two or more physical domains each containing different physical processes, but which share a common interfacial surface. (Note that coupling through separate surfaces at a distance - e.g. radiation heat transfer, can be considered a variation of this type of coupling, but is not discussed here). The physical processes are independent in each domain apart from the interaction at the interface. One example is the modeling of a reentry vehicle as it travels through the atmosphere (see Figure 3.1(b)). The flight of the vehicle through the atmosphere creates a pressure load on the shell of the vehicle which in turn affects the structural dynamics of the interior of the vehicle. Here the two domains are the fluid exterior to the vehicle (compressible, turbulent fluid flow) and the interior of the vehicle (structural dynamics) coupled through the shell of the vehicle (interface).

In systems of this type the physics in each domain is typically modeled as a set of partial differential equations that are coupled through boundary conditions. Although the applications share a common interfacial surface, the interface could have different spatial discretizations for each. After spatial discretization, a generic two-component interfacially coupled system can be represented as

$$\begin{aligned} f_0(x_0, z_{0,0}) &= 0, \\ f_1(x_1, z_{1,0}) &= 0, \\ z_{0,0} &= r_{0,0}(x_1) \neq x_1, \\ z_{1,0} &= r_{1,0}(x_0) \neq x_0, \\ &\Rightarrow \\ \bar{f}(\bar{x}) = \begin{bmatrix} f_0(x_0, r_{0,0}(x_1)) \\ f_1(x_1, r_{1,0}(x_0)) \end{bmatrix} &= 0. \end{aligned} \tag{3.2}$$

Here the transfer functions $r_{0,0}(\ldots) : \mathbb{R}^{n_{x_1}} \to \mathbb{R}^{n_{z_{0,0}}}$ and $r_{1,0}(\ldots) : \mathbb{R}^{n_{x_0}} \to \mathbb{R}^{n_{z_{1,0}}}$ represent the interfaces between systems 0 and 1 and typically map their argument to a much lower dimensional space, i.e., $n_{z_{1,0}} \ll n_{x_0}$ and $n_{z_{0,0}} \ll n_{x_1}$. In this case the discrete residuals $f_0(\ldots) : \mathbb{R}^{n_{x_0}+n_{z_{0,0}}} \to \mathbb{R}^{n_{x_0}}$ and $f_1(\ldots) : \mathbb{R}^{n_{x_1}+n_{z_{1,0}}} \to \mathbb{R}^{n_{x_1}}$ only depend on the other solution variables through the transfer functions $r_{0,0}$ and $r_{1,0}$.

3.1.3 Network Systems Model Example

The third type of coupling considered here is that of **network systems**. In this case a collection (possibly a large number) of separate domains are coupled together through a series of low-dimensional (typically lower fidelity) interactions. This is a remarkably common type of problem that appears in many different technical settings when modeling large or complex engineering systems. One example of particular relevance to the authors is the systems-level analysis of an operating nuclear power plant. As conceptually illustrated in Figure 3.1(c), the overall plant can be modeled as consisting of many distinct components (pumps, reactors, turbines, heat exchangers, etc.) that are coupled together through a series of connections (or vertices). The state of each unit evolves according to the particular nature of its own physics, but also according to the mass and energy balances at its connections. These connections in turn are assumed to satisfy a small number of consistency constraints, such as overall mass, momentum, and energy conservation.

The fidelity of the components in a network system can vary, from high-fidelity single and multi-physics models to low fidelity compact models. Thus the overall mathematical structure of such systems can take many forms, but have the unifying feature that the network model must be discrete in space, i.e., either a system of ordinary or differential-algebraic equations (for time-dependent problems) or a system of algebraic equations (for steady problems). The models for each system component could be a single or multi-physics system of partial differential equations or a set of algebraic/ordinary-differential/differential-algebraic equations. To mathematically represent such a system, consider the simple network schematic described by Figure 3.2, consisting of three vertices (or nodes) connecting together two components or devices.

Component 0 will be assigned as physics application 0 and component 1 will be assigned as physics application 1. The network vertices are used to couple the components together by enforcing conservation between components. For example, vertex 1 could enforce that the total energy leaving component 0 is equal to the total energy entering component 1. For demonstration purposes assume that the equation sets at vertices 0, 1, and 2 are non-trivial and therefore are each assigned as an individual physics application. Vertex 0 is assigned as physics application 2, vertex 1 is assigned as physics application 3, and vertex 2 assigned as physics application 4. Each vertex in the network is thus associated with its own set of state variables (x_2, x_3, and x_4). In a plant balance, these might represent, for example, species mass flow rates (or concentrations), pressure, and temperature (or enthalpy).

Each component in the network will have dependencies on the state variables of the vertices connected to that particular component. For example, the constraint equations for component 0, f_0 are dependent on the state variables of the vertices 0 and 1 (e.g. $f_0(x_0, z_{0,0}, z_{0,1})$). Similarly, each vertex equation set will have dependencies on the components connected to that vertex. For example, vertex 0 will have a dependency on component 0 (i.e. $f_2(x_2, r_{2,0}(x_0))$). In terms of the general model, the coupled system of constraint equations, is represented mathematically as

$$\begin{aligned}
&f_0(x_0, z_{0,0}, z_{0,1}) = 0,\\
&f_1(x_1, z_{1,0}, z_{1,1}) = 0,\\
&f_2(x_2, z_{2,0}) = 0,\\
&f_3(x_3, z_{3,0}, z_{3,1}) = 0,\\
&f_4(x_4, z_{4,0}) = 0.
\end{aligned} \tag{3.3}$$

The coupling parameters are determined from the network connectivity as described above. One extra assumption is made in that the transfer functions for the components are usually trivial and that they map directly into the state variables at the vertices (e.g. $r_{0,1}(x_2) = x_2$). No simplifying assumptions are made for transfer functions from used in the vertex equation sets. This results in the following set of transfer functions

$$\begin{aligned}
&z_{0,0} = r_{0,0}(x_2) = x_2,\\
&z_{0,1} = r_{0,1}(x_3) = x_3,\\
&z_{1,0} = r_{1,0}(x_3) = x_3,\\
&z_{1,1} = r_{1,1}(x_4) = x_4,\\
&z_{2,0} = r_{2,0}(x_0),\\
&z_{3,0} = r_{3,0}(x_0),\\
&z_{3,1} = r_{3,1}(x_1),\\
&z_{4,0} = r_{4,0}(x_1).
\end{aligned} \tag{3.4}$$

Inserting the coupling parameter equations 3.4 into the general model 3.3 yields the final set of equations

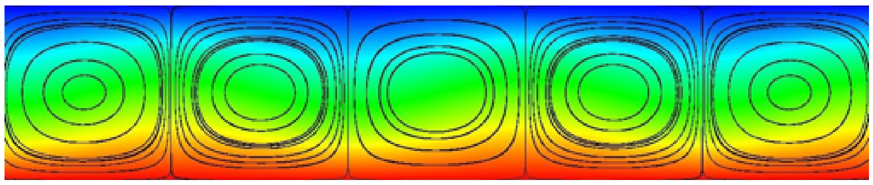
$$\begin{aligned}
&f_0(x_0, x_2, x_3) = 0,\\
&f_1(x_1, x_3, x_4) = 0,\\
&f_2(x_2, r_{2,0}(x_0)) = 0,\\
&f_3(x_3, r_{3,0}(x_0), r_{3,1}(x_1)) = 0,\\
&f_4(x_4, r_{4,0}(x_1)) = 0.
\end{aligned} \tag{3.5}$$

These equations could, for example, represent high-fidelity PDE models of process operations equipment which determine for example the outlet species mass flow rates as a function of supplied inlet mass flow rates supplied as boundary conditions.

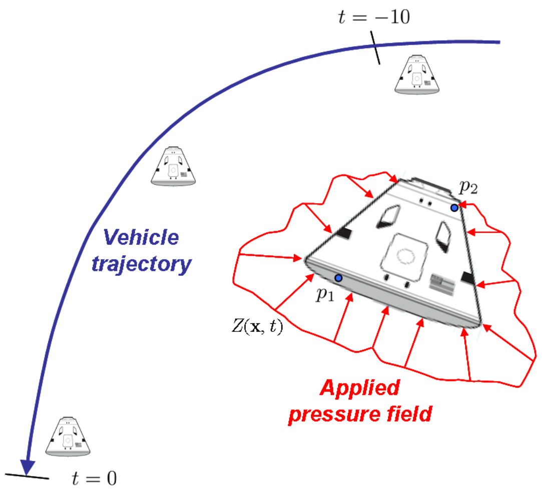
3.1.4 Summary of Coupling Model Examples

The above models demonstrate how to map some generic multi-physics couplings (shared domain, interfacial, and network) to the general model. We stress that these are only examples and that the mapping to the general model is one of many choices we could have used. The mapping is very problem specific. Each coupling is unique.

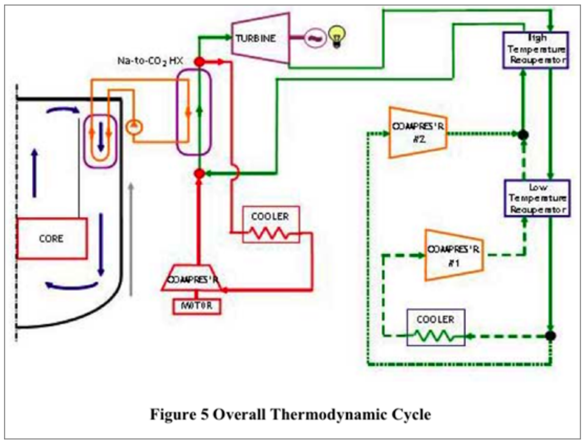
Often in computational settings, all three types of coupled systems may occur in a given coupled system simulation. Therefore it is important to leverage nonlinear solution algorithms that are optimized for each type of coupled system and strength of coupling. An overview of available approaches is provided in the following section.



(a) Multi-physics coupling in a shared spatial domain



(b) Interfacial coupling



TURBINE
Na-to-CO$_2$ HX
High Temperature Recuperator
COMPRES'R #2
Low Temperature Recuperator
CORE
COOLER
COMPRES'R #1
COMPRES'R
MOTOR
COOLER
Figure 5 Overall Thermodynamic Cycle

(c) Network coupling

Figure 3.1. Three types of coupled systems. (a) Multi-physics coupling of two-dimensional fluid flow (black streamlines) and temperature (color gradient). (b) Interfacial coupling between turbulent air flow and structural dynamics through the shell of the vehicle (reprinted with permission from [8]). (c) Network coupling of a nuclear plant simulation. Each unit operation (turbine, pump, reactor core, etc...) is either a low-fidelity lumped parameter model or a high-fidelity PDE simulation.

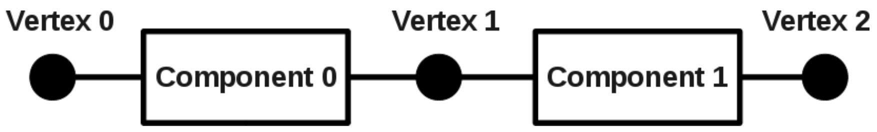


Figure 3.2. A simple two component network example.

3.2 Various Solution Strategies Demonstrated on Model Examples

When discussing solution strategies for the types of coupled systems presented in the previous section, one must distinguish between coupled physics and coupled simulation codes. In principle, a single simulation code could be written to simulate any of these types of coupled multi-physics systems. However, an important alternative is for multiple simulation codes to be coupled in some fashion to simulate the multi-physics system. In this case each simulation code separately approximates the solution to one or more sets of physics in the coupled system. This is the situation that is of particular relevance to LIME. When developing a new multi-physics application there is an important advantage of this approach: it enables one to leverage existing simulation code technology that may represent many man-years of development work. Furthermore, in many cases domain-specific knowledge is built into existing simulation codes for specialized physics that may be difficult or impossible to generalize when incorporating new physics. However, there are problems that can arise with this approach. For example, it may be difficult to extract the relevant derivative information needed for Newton-based solution algorithms (which, as mentioned in section 2.3, have convergence properties that are important for many problems). For this reason, numerous nonlinear solution strategies that try to obtain Newton-like efficiency and robustness (while minimizing the amount of derivative information necessary) have been studied [10]. In this section we provide a brief overview of some of these methods that are more strongly suited to the types of coupled systems discussed previously.

3.2.1 Forward Coupling

A significant simplification occurs for coupled systems when the system is only forwardly coupled, i.e., the two-component shared-domain multi-physics system (3.1) becomes

$$\begin{aligned} f_0(x_0) &= 0, \\ f_1(x_1, x_0) &= 0, \end{aligned} \tag{3.6}$$

while the two-component interfacially coupled system (3.2) becomes

$$\begin{aligned} f_0(x_0) &= 0, \\ f_1(x_1, r_{1,0}(x_0)) &= 0. \end{aligned} \tag{3.7}$$

In both cases, the solution to the first application x_0 can be solved for independently and substituted into the remaining equations. This allows any nonlinear solver method to be separately applied to the first and remaining systems with no solver communication between them. Of course, this setting is clearly not generic, but forms the basis of the Picard method when treating the fully coupled system.

3.2.2 Picard Iteration

Picard iteration, also known as successive substitution, is a simple numerical method for approximating the solution to a fully coupled system (3.1), (3.2) or (3.5). It works by solving each component in the coupled system for its solution variables, treating the other variables as fixed quantities. This is repeated in a round-robin fashion until some measure of convergence is achieved (typically when the size of the change is small for the solution variables from iteration to iteration, and/or the size of the residual of each component evaluated at the most current solution values). For example, an algorithm for applying this technique to the two-component interfacially coupled system (3.2) is displayed in Algorithm 3. As with

Algorithm 3 Picard iteration for the two-component interfacially coupled system (3.2)

```
Require: Initial guesses $x_0^{(0)}$ and $x_1^{(0)}$ for $x_0$ and $x_1$:
  k = 0
  while not converged do
    k = k+1
    Solve $f_0(x_0^{(k)}, r_{0,0}(x_1^{(k-1)})) = 0$ for $x_0^{(k)}$
    Solve $f_1(x_1^{(k)}, r_{1,0}(x_0^{(k)})) = 0$ for $x_1^{(k)}$
  end while
```

forward coupling, any solver method can be used for each of the single-component solves in Algorithm 3. The problem with this approach is that the convergence rate is slow (linear), and needs additional requirements on f_0 and f_1 to converge [10]. Picard methods are attractive when the different components are only weakly coupled or when there is a very good initial guess and/or only an approximate solution is needed. For example, a Picard method would tend to work well to solve for the implicit time-step update in a transient predictor/corrector method (where the explicit predictor gives a good initial guess and the implicit corrector system only needs to be converged a little to guarantee stability).

3.2.3 Newton's Method

Due to the slow rate of convergence and lack of robustness of Picard iteration, Newton's method is often preferred for simulating coupled systems. However its implementation is significantly more complicated and potentially costly. At each iteration, it requires solving the following Newton systems:

$$\begin{bmatrix} \frac{\partial f_0}{\partial x_0} & \frac{\partial f_0}{\partial x_1} \\ \frac{\partial f_1}{\partial x_0} & \frac{\partial f_1}{\partial x_1} \end{bmatrix} \begin{bmatrix} \Delta x_0^{(k)} \\ \Delta x_1^{(k)} \end{bmatrix} = - \begin{bmatrix} f_0(x_0^{(k-1)}, x_1^{(k-1)}) \\ f_1(x_1^{(k-1)}, x_0^{(k-1)}) \end{bmatrix} \tag{3.8}$$

for shared-domain multi-physics coupling,

$$\begin{bmatrix} \frac{\partial f_0}{\partial x_0} & \frac{\partial f_0}{\partial z_{0,0}} \frac{\partial r_{0,0}}{\partial x_1} \\ \frac{\partial f_1}{\partial z_{1,0}} \frac{\partial r_{1,0}}{\partial x_0} & \frac{\partial f_1}{\partial x_1} \end{bmatrix} \begin{bmatrix} \Delta x_0^{(k)} \\ \Delta x_1^{(k)} \end{bmatrix} = - \begin{bmatrix} f_0(x_0^{(k-1)}, r_{0,0}(x_1^{(k-1)})) \\ f_1(x_1^{(k-1)}, r_{1,0}(x_0^{(k-1)})) \end{bmatrix} \tag{3.9}$$

for interfacial coupling, and

$$\begin{bmatrix} \frac{\partial f_0}{\partial x_0} & 0 & \frac{\partial f_0}{\partial x_{2,0}} & \frac{\partial f_0}{\partial x_{2,1}} & 0 \\ 0 & \frac{\partial f_1}{\partial x_1} & 0 & \frac{\partial f_1}{\partial x_{2,1}} & \frac{\partial f_1}{\partial x_{2,2}} \\ \frac{\partial f_2}{\partial z_{2,0}}\frac{\partial r_{2,0}}{\partial x_0} & 0 & \frac{\partial f_2}{\partial x_2} & 0 & 0 \\ \frac{\partial f_3}{\partial z_{3,0}}\frac{\partial r_{3,0}}{\partial x_0} & \frac{\partial f_3}{\partial z_{3,1}}\frac{\partial r_{3,1}}{\partial x_1} & 0 & \frac{\partial f_3}{\partial x_3} & 0 \\ 0 & \frac{\partial f_4}{\partial z_{4,0}}\frac{\partial r_{4,0}}{\partial x_1} & 0 & 0 & \frac{\partial f_4}{\partial x_4} \end{bmatrix} \begin{bmatrix} \Delta x_0^{(k)} \\ \Delta x_1^{(k)} \\ \Delta x_2^{(k)} \\ \Delta x_3^{(k)} \\ \Delta x_4^{(k)} \end{bmatrix} = - \begin{bmatrix} f_0(x_0^{(k-1)}, x_2^{(k-1)}, x_3^{(k-1)}) \\ f_1(x_1^{(k-1)}, x_3^{(k-1)}, x_4^{(k-1)}) \\ f_2(x_2^{(k-1)}, r_{2,0}(x_0^{(k-1)})) \\ f_3(x_3^{(k-1)}, r_{3,0}(x_0^{(k-1)}), r_{3,1}(x_1^{(k-1)})) \\ f_4(x_4^{(k-1)}, r_{4,0}(x_1^{(k-1)})) \end{bmatrix} \tag{3.10}$$

for network coupling, where $x_0^{(k)} = x_0^{(k-1)} + \Delta x_0^{(k)}$, and so on. Numerous methods are available for estimating the cross-physics derivatives appearing in these systems [10].

3.2.4 Nonlinear Elimination

The disadvantage of the full Newton approach is that it requires forming and solving the fully-coupled Newton systems (3.8), (3.9), and (3.10), making it very difficult to use nonlinear solvers, linear solvers, and preconditioners that are specialized to each system component. An alternative approach that maintains the quadratic convergence of Newton's method, but allows for greater flexibility in the choice of solver for each system component is nonlinear elimination. Nonlinear elimination has been used in circuit simulation [16, 23] (also called the "Two-level Newton" technique in the circuit community), aerostructures [29], and chemically reacting flows [28]. A theoretical analysis with convergence proofs can be found in [13, 27].

The nonlinear elimination approach works by eliminating solution variables from each system component, relying on the Implicit Function Theorem. For example, the equation $f_0(x_0, z_{0,0}(x_1)) = 0$ for an interfacially coupled system can be thought of as an implicit equation defining x_0 as a function of x_1, which can be numerically evaluated using any appropriate nonlinear solver method. We then apply a nonlinear solver method to the reduced system

$$f_1(x_1, z_{1,0}(x_0(x_1))) = 0. \tag{3.11}$$

Applying Newton's method to this system requires computation of the sensitivity $\partial x_0/\partial x_1$, which by the Implicit Function Theorem is

$$\frac{\partial x_0}{\partial x_1} = -\left(\frac{\partial f_0}{\partial x_0}\right)^{-1} \frac{\partial f_0}{\partial z_{0,0}} \frac{\partial r_{0,0}}{\partial x_1}. \tag{3.12}$$

Note this linear system involves the same matrix as Newton's method applied to the system $f_0(x_0, z_{0,0}(x_1)) = 0$ with x_1 held fixed, with $n_{z_{0,0}}$ right-hand-sides. Clearly this approach is only effective when $n_{z_{0,0}}$ is reasonably small, and therefore is typically not appropriate for shared-domain multi-physics problems such as (3.1). The Newton system required for Newton's method applied to the reduced system (3.11) is

$$\left(\frac{\partial f_1}{\partial z_{1,0}}\frac{\partial r_{1,0}}{\partial x_0}\frac{\partial x_0}{\partial x_1}+\frac{\partial f_1}{\partial x_1}\right)\Delta x_1^{(k)} = -f_1(x_1^{(k-1)}, z_{1,0}(x_0^{(k)})). \tag{3.13}$$

The complete algorithm for a Newton-based nonlinear elimination method for interfacial coupling is displayed in Algorithm 4. Note that one could avoid the sensitivity computations

Algorithm 4 Newton-based nonlinear elimination for two-component interfacial coupling.

```
Require: Initial guesses $x_0^{(0)}$ and $x_1^{(0)}$ for $x_0$ and $x_1$
  k = 0
  while not converged do
    k = k+1
    Solve $f_0(x_0^{(k)}, r_{0,0}(x_1^{(k-1)})) = 0$ for $x_0^{(k)}$
    Compute $\frac{\partial x_0^{(k)}}{\partial x_1^{(k-1)}} = -\frac{\partial f_0}{\partial x_0}^{-1}\frac{\partial f_0}{\partial r_{0,0}}\frac{\partial r_{0,0}}{\partial x_1^{(k-1)}}$
    Solve $\left(\frac{\partial f_1}{\partial r_{1,0}}\frac{\partial r_{1,0}}{\partial x_0^{(k)}}\frac{\partial x_0^{(k)}}{\partial x_1^{(k-1)}}+\frac{\partial f_1}{\partial x_1^{(k-1)}}\right)\Delta x_1^{(k)} = -f_1(x_1^{(k-1)}, r_{1,0}(x_0^{(k)}))$
    $x_1^{(k)} = x_1^{(k-1)} + \Delta x_1^{(k)}$
  end while
```

of 3.12 by applying a Jacobian-free Newton-Krylov method [12] to the outer solve 3.13.

Nonlinear elimination is also effective for network problems such as (3.5). In this example, both components will be eliminated (i.e. both x_0 and x_1 are eliminated), leaving just the network vertex solve. More precisely, the equations $f_0(x_0, x_2, x_3) = 0$ and $f_1(x_1, x_3, x_4) = 0$ implicitly defines x_0 as a function of x_2, x_3 and x_1 as a function of x_3, x_4. This results in the following reduced system to solve with Newton's method

$$\begin{aligned} &f_2(x_2, z_{2,0}(x_0(x_2, x_3))) = 0,\\ &f_3(x_3, z_{3,0}(x_0(x_2, x_3)), z_{3,1}(x_1(x_3, x_4))) = 0,\\ &f_4(x_4, z_{4,0}(x_1(x_3, x_4))) = 0. \end{aligned} \tag{3.14}$$

The resulting Newton system at the kth iterate is

$$\begin{bmatrix} \frac{\partial f_2}{\partial x_2^{(k-1)}} + \frac{\partial f_2}{\partial z_{2,0}^{(k)}}\frac{\partial z_{2,0}^{(k)}}{\partial x_2^{(k-1)}} & \frac{\partial f_2}{\partial z_{2,0}^{(k)}}\frac{\partial z_{2,0}^{(k)}}{\partial x_3^{(k-1)}} & 0 \\ \frac{\partial f_3}{\partial z_{3,0}^{(k)}}\frac{\partial z_{3,0}^{(k)}}{\partial x_2^{(k-1)}} & \frac{\partial f_3}{\partial x_3^{(k-1)}} + \frac{\partial f_3}{\partial z_{3,0}^{(k)}}\frac{\partial z_{3,0}^{(k)}}{\partial x_3^{(k-1)}} + \frac{\partial f_3}{\partial z_{3,1}^{(k)}}\frac{\partial z_{3,1}^{(k)}}{\partial x_3^{(k-1)}} & \frac{\partial f_3}{\partial z_{3,1}^{(k)}}\frac{\partial z_{3,1}^{(k)}}{\partial x_4^{(k-1)}} \\ 0 & \frac{\partial f_4}{\partial z_{4,0}^{(k)}}\frac{\partial z_{4,0}^{(k)}}{\partial x_3^{(k-1)}} & \frac{\partial f_4}{\partial x_4^{(k-1)}} + \frac{\partial f_4}{\partial z_{4,0}^{(k)}}\frac{\partial z_{4,0}^{(k)}}{\partial x_4^{(k-1)}} \end{bmatrix} \begin{bmatrix} \Delta x_2^{(k)} \\ \Delta x_3^{(k)} \\ \Delta x_4^{(k)} \end{bmatrix} = \\ - \begin{bmatrix} f_2(x_2^{(k-1)}, z_{2,0}^{(k)}) \\ f_3(x_3^{(k-1)}, z_{3,0}^{(k)}, z_{3,1}^{(k)}) \\ f_4(x_4^{(k-1)}, z_{4,0}^{(k)}) \end{bmatrix}, \tag{3.15}$$

where the derivatives of the transfer functions, $\frac{\partial z}{\partial x}$, are needed. Using the transfer function relations 3.4 along with the implicit function theorem, the required derivatives are given by

$$\begin{aligned}
\frac{\partial z_{2,0}}{\partial x_2} &= -\frac{\partial r_{2,0}}{\partial x_0}\left(\frac{\partial f_0}{\partial x_0}\right)^{-1}\frac{\partial f_0}{\partial x_2}, \\
\frac{\partial z_{2,0}}{\partial x_3} &= -\frac{\partial r_{2,0}}{\partial x_0}\left(\frac{\partial f_0}{\partial x_0}\right)^{-1}\frac{\partial f_0}{\partial x_3}, \\
\frac{\partial z_{3,0}}{\partial x_2} &= -\frac{\partial r_{3,0}}{\partial x_0}\left(\frac{\partial f_0}{\partial x_0}\right)^{-1}\frac{\partial f_0}{\partial x_2}, \\
\frac{\partial z_{3,0}}{\partial x_3} &= -\frac{\partial r_{3,0}}{\partial x_0}\left(\frac{\partial f_0}{\partial x_0}\right)^{-1}\frac{\partial f_0}{\partial x_3}, \\
\frac{\partial z_{3,1}}{\partial x_3} &= -\frac{\partial r_{3,1}}{\partial x_1}\left(\frac{\partial f_1}{\partial x_1}\right)^{-1}\frac{\partial f_1}{\partial x_3}, \\
\frac{\partial z_{3,1}}{\partial x_4} &= -\frac{\partial r_{3,1}}{\partial x_1}\left(\frac{\partial f_1}{\partial x_1}\right)^{-1}\frac{\partial f_1}{\partial x_4}, \\
\frac{\partial z_{4,0}}{\partial x_3} &= -\frac{\partial r_{4,0}}{\partial x_1}\left(\frac{\partial f_1}{\partial x_1}\right)^{-1}\frac{\partial f_1}{\partial x_3}, \\
\frac{\partial z_{4,0}}{\partial x_4} &= -\frac{\partial r_{4,0}}{\partial x_1}\left(\frac{\partial f_1}{\partial x_1}\right)^{-1}\frac{\partial f_1}{\partial x_4}.
\end{aligned} \tag{3.16}$$

Note that z generally is a vector not a scalar. This implies that multiple inversions of the application Jacobian are necessary to generate these derivative blocks. Codes could leverage multivector support for inverting multiple right-hand-sides simultaneously. The complete algorithm for a Newton-based nonlinear elimination method for network coupling is displayed in Algorithm 5.

The above example shows only a single type of nonlinear elimination. Given a group of multi-physics applications to couple, many variations and combinations of nonlinear elimination and other solution strategies can be employed to achieve a convergent system. The approximate block Newton methods analyzed in [28] are a good example.

Algorithm 5 Newton-based nonlinear elimination for two-component network coupling.

```
Require: Initial guesses $x_0^{(0)}$, $x_1^{(0)}$, $x_2^{(0)}$, $x_3^{(0)}$, and $x_4^{(0)}$ for $x_0$, $x_1$, $x_2$, $x_3$, and $x_4$.
  k = 0
  while not converged do
    k = k+1
    Solve $f_0(x_0^{(k)}, x_2^{(k-1)}, x_3^{(k-1)}) = 0$ for $x_0^{(k)}$
    Solve $f_1(x_1^{(k)}, x_3^{(k-1)}, x_4^{(k-1)}) = 0$ for $x_1^{(k)}$
    Evaluate $z_{2,0}^{(k)} = r_{2,0}(x_0^{(k)})$
    Evaluate $z_{3,0}^{(k)} = r_{3,0}(x_0^{(k)})$
    Evaluate $z_{3,1}^{(k)} = r_{3,1}(x_1^{(k)})$
    Evaluate $z_{4,0}^{(k)} = r_{4,0}(x_1^{(k)})$
    Compute $\partial z^{(k)}/\partial x^{(k-1)}$ via (3.16)
    Solve Newton system (3.15) for $\Delta x_2^{(k)}$, $\Delta x_3^{(k)}$, and $\Delta x_4^{(k)}$
    $x_2^{(k)} = x_2^{(k-1)} + \Delta x_2^{(k)}$
    $x_3^{(k)} = x_3^{(k-1)} + \Delta x_3^{(k)}$
    $x_4^{(k)} = x_4^{(k-1)} + \Delta x_4^{(k)}$
  end while
```

Chapter 4

Stability of Multi-physics Coupling Algorithms

Coupling multiple physics applications can result in unexpected convergence behavior. Even if the individual applications use stable discretizations and stable and robust solution techniques, there is no guarantee that the coupled system, even when using something as simple as Picard iteration, will be stable. The use of operator splitting methods is a classic example of this. It has recently been shown that even simple 1D PDE models can exhibit complex numerical instabilities [24, 25]. In fact such phenomena are not limited to operator splitting. Something as simple as a steady-state two equation diffusion/reaction model with interfacial coupling can exhibit instabilities when coupled [20]. In light of this, before performing a multi-physics coupling, the coupling algorithm should be clearly defined and analyzed for stability.

References

[1] CASL website. Online. http://www.casl.gov. iv

[2] B. M. Adams, W. J. Bonhoff, K. R. Dalbey, J. P. Eddy, M. S. Eldred, D. M. Gay, K. Haskell, P. D. Hough, and L. P. Swiler. Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 5.0 user's manual. Sandia Technical Report SAND2010-2183, Sandia National Laboratories, December 2009. 12

[3] R. A. Bartlett, D. M. Gay, and E. T. Phipps. Automatic differentiation of C++ codes for large-scale scientific computing. In V. N. Alexandrov, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, editors, *Computational Science ICCS 2006*, volume 3994 of *Lecture Notes in Computer Science*, pages 525–532. Springer, 2006. 11

[4] R. B. Bird, W. E. Stewart, and B. N. Lightfoot. *Transport Phenomena*. John Wiley & Sons, 2 edition, 2007. 3, 6

[5] A. R. Conn, N. I. M. Gould, and P. L. Toint. *Trust-Region Methods*. MPS-SIAM Series on Optimization. SIAM, 1987. 10

[6] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Series in Automatic Computation. Prentice-Hall, Englewood Cliffs, NJ, 1983. 9, 10

[7] C. A. Felippa, K. C. Park, and C. Farhat. Partitioned analysis of coupled mechanical systems. *Computer Methods in Applied Mechanics and Engineering*, 190:3247–3270, 2001. 3

[8] R. V. Field. Stochastic models: Theory and simulation. Technical Report SAND2008-1365, Sandia National Laboratories, 2008. 20

[9] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, PA, 2000. ISBN 0–89871–451–6. 11

[10] R. Hooper, M. Hopkins, R. Pawlowski, B. Carnes, and H. K. Moffat. Final report on ldrd project: Coupling strategies for multi-physics applications. Technical Report SAND2007-7146, Sandia National Laboratories, 2008. 12, 21, 22, 23

[11] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*, volume 16 of *Frontiers in Applied Mathematics*. SIAM, Philadelphia, PA, 1995. 9, 10

[12] D. A. Knoll and D. E. Keyes. Jacobian-free Newton–Krylov methods: a survey of approaches and applications. *J. Comput. Phys.*, 193:357–397, 2004. 11, 12, 24

[13] P. J. Lanzkron, D. J. Rose, and J. T. Wilkes. An analysis of approximate nonlinear elimination. *SIAM J. Sci. Comput.*, 17(2):538–559, 1996. 23

[14] H. G. Matthies and J. Steindorf. Partitioned strong coupling algorithms for fluid-structure interaction. *Computers & Structures*, 81(8-11):805–812, 2003. 10, 12

[15] H. G. Matthies, R. Niekamp, and J. Steindorf. Algorithms for strong coupling procedures. *Computer Methods in Applied Mechanics and Engineering*, 195:2028–2049, 2006. 3

[16] K. Mayaram and D. O. Pederson. Coupling algorithms for mixed-level circuit and device simulation. *IEEE Transactions on Computer Aided Design*, 11(8):1003–1012, 1992. 23

[17] C. Michler, E. H. van Brummelen, S. J. Hulshoff, and R. de Borst. The relevance of conservation for stability and accuracy of numerical methods for fluid-structure interaction. *Computer Methods in Applied Mechanics and Engineering*, 192:4195–4215, 2003. 4

[18] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer-Verlag, New York, 1999. 10

[19] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, New York, 1970. 9, 10

[20] A. Y. A. Pandy and J. Derby. Fixed-point convergence of modular, steady-state heat transfer models coupling multiple scales and phenomena for melt-crystal growth . *IJNME*, 67(12):1768–1789, 2006. 27

[21] R. P. Pawlowski, J. N. Shadid, J. P. Simonis, and H. F. Walker. Globalization techniques for newton-krylov methods and applications to the fully coupled solution of the navier-stokes equations. *SIAM Review*, 48(4):700–721, December 2006. 10

[22] E. Phipps, M. Eldred, A. Salinger, and C. Webster. LDRD final report: Capabilities for uncertainty in predictive science. Sandia Technical Report SAND2008-6527, Sandia National Laboratories, P.O. Box 5800, Albuquerque, New Mexico 87185, September 2008. iv

[23] N. B. G. Rabbat, A. L. Sangiovanni-Vincentelli, and H. Y. Hsieh. Multilevel newton algorithm with macromodeling and latency for the analysis of large-scale nonlinear circuits in the time domain. *IEEE Transactions on Circuits and Systems*, 1979. 23

[24] D. L. Ropp and J. N. Shadid. Stability of operator splitting methods for systems with indefinite operators: reaction-diffusion systems. *Journal of Computational Physics*, 203: 449–466, 2005. 27

[25] D. L. Ropp and J. N. Shadid. Stability of operator splitting methods for systems with indefinite operators: Advection-diffusion-reaction systems. *Journal of Computational Physics*, 228:3508–3516, 2009. 27

[26] R. C. Schmidt, K. Belcourt, K. T. Clarno, R. Hooper, L. L. Humphries, A. L. Lorber, R. J. Pryor, and W. F. Spotz. Foundational development of an advanced nuclear reactor integrated safety code. Technical Report SAND2010-0878, Sandia National Laboratories, 2010. 2

[27] J. T. Wilkes. A new method for solving systems of nonlinear equations in circuit simulation. Technical report, CiteSeer [http://cs1.ist.psu.edu/cgi-bin/oai.cgi] (United States), 1994. URL `http://citeseer.ist.psu.edu/129347.html`. 23

[28] A. Yeckel, L. Lun, and J. Derby. An approximate block newton method for coupled iterations of nonlinear solvers: Theory and conjugate heat transfer applications. *Journal of Computational Physics*, 228:8566–8588, 2009. 23, 25

[29] D. P. Young, W. P. Huffman, R. G. Melvin, C. L. Hilmes, and F. T. Johnson. Nonlinear elimination in aerodynamic analysis and design. In L. T. Biegler, O. Ghattas, and M. Heinkenschloss, editors, *Large-scale PDE-Constrained Optimization*, volume 30 of *Lecture Notes in Computational Science A*, pages 17–43. Springer-Verlag, 2003. 23

Appendix A

Mathematical Notation

Symbol	Definition	Example
$:=$	definition. $x := y$ means that x is defined to be another name for y.	$cosh(x) := \frac{e^x+e^{-x}}{2}$
$\{a_i\}$	set of objects. A list of elements defining a set.	$\{a_i\}$ is the set consisting of the elements $a_0, a_1, \ldots$
$\in$	set membership. $a \in S$ means a is an element of the set S.	Given the set $S := \{1, 2, 3\}$, $2 \in S$.
$\mathbb{R}$	the set of real numbers.	$\mathbb{R}^5$ is a set of 5 real numbers $\mathbb{R}^{n_x}$ is a set of n_x real numbers where n_x is a positive integer $\mathbb{R}^{(n_x+n_x)}$ is a set of $2n_x$ real numbers where n_x is a positive integer $\mathbb{R}^{(n_x \times n_y)}$ is a set of n_x times n_y real numbers with positive integers n_x, n_y
$\mathbb{N}$	the set of natural numbers, $\{0, 1, 2, 3, ...\}$.	
$(\ )$	function application. $f(x)$ means the value of the function f at the element x.	If $f(x) := x^2$, then $f(2) = 2^2 = 4$.
$\{\ :\ \}$	set builder notation. $\{x : P(x)\}$ means the set of all x for which $P(x)$ is true.	$\{n \in \mathbb{N} : n^2 < 20\} = \{1, 2, 3, 4\}$
$[\ ]$	closed interval. $[a, b] = \{x \in \mathbb{R} : a \leq x \leq b\}$.	0 and 1/2 are in the interval [0,1].
$\rightarrow$	function arrow. $f : X \rightarrow Y$ means the function f maps the set X into the set Y	
$\Rightarrow$	material implication. $A \Rightarrow B$ means if A is true then B is also true; if A is false then nothing is said about B.	

Table A.1. Mathematical notation used in this manual.

DISTRIBUTION:

1	MS	1318	R. Bartlett, 1411
1	MS	0748	N. Belcourt, 6232
1	MS	1318	R. Hooper, 1414
1	MS	1318	R. Pawlowski, 1414
1	MS	1318	R. Schmidt, 1437
1	MS	0899	Technical Library, 9536 (electronic)

Sandia National Laboratories